Context as a Resource: A Service-Oriented Approach for Context-Awareness

Daniel Romero

Université Lille1 Sciences et Technologies

EDSPI École Doctorale des Sciences pour l'Ingénieur - Lille

Département de formation doctorale en informatique **École doctorale SPI Lille**
UFR IEEA

Context as a Resource: A Service-Oriented Approach for Context-Awareness

THÈSE

présentée et soutenue publiquement le 4 Juillet 2011

pour l'obtention du

Doctorat de l'université des Sciences et Technologies de Lille

(spécialité informatique)

par

Daniel Francisco Romero Acero

Composition du jury

Rapporteurs :	Yolande Berbers, *Katholieke Universiteit Leuven, Belgium* Noël de Palma, *Université de Grenoble, France*
Examinateurs :	Françoise André, *Université de Rennes I, France* Hausi A. Müller, *University of Victoria, Canada* Ernesto Exposito, *Institut National des Sciences Appliquées (INSA), France*
Directrice :	Laurence Duchien, *Université Lille I, France*
Co-directeur :	Lionel Seinturier, *Université Lille I, France*
Co-encadrant :	Romain Rouvoy, *Université Lille I, France*

Laboratoire d'Informatique Fondamentale de Lille – UMR USTL/CNRS 8022 – INRIA Lille Nord Europe

lifL cnrs INRIA

Mis en page avec la classe thloria.

Contents

List of Figures

List of Tables

Abstract

Nowadays, ubiquitous environments become part of our daily lives. At home, work, cars, hotels, supermarkets and others public spaces we find technologies (electronics and computational elements) that try to make our life simpler and easier in a transparent way. In recent years, the potential of these environments is more and more exploited with the advent and widespread usage of smartphones. This kind of devices enables the execution of applications that are able to adapt seamlessly to the current environment state. These applications, called context-aware applications, benefit from the context information and services that are present in ubiquitous environments to improve and change automatically their behavior. However, such adaptations require the integration of information regarding heterogeneity in terms of devices, execution platforms, and communication protocols as well the mobility of applications so that the different responsibilities of the adaptation can be distributed.

In order to face these issues, and considering the limitations of existing solutions, we provide two major contributions in this dissertation: *i)* **SPACES**, a middleware approach to integrate context information and *ii)* **Ubiquitous Feedback Control Loops** (Ubiquitous FCLs), as an approach to adapt context-aware applications. In particular, in SPACES we define a metamodel inspired on REST (REpresentational State Transfer) for fostering the exchange of context information as resources, which represents the keystone of our approach. Then, we define *SPACES Connectors* modularizing the different concepts and concerns identified by the metamodel. The connectors are designed by using *Component Based Software Engineering* (CBSE) principles and then they are incorporated into the *Service Component Architecture* (SCA) model to be used in different kinds of applications, not only context-aware applications.

With the SPACES definition we are able to state the second major contribution— *i.e.*, Ubiquitous FCLs. Inspired on concepts from *Autonomic Computing*, this kind of FCLs provides the flexibility required to integrate new participants in the adaptation process (*e.g.*, context-aware applications, services and legacy systems) by supporting mobility and the incorporation of new communication mechanisms when required. In the core of the Ubiquitous FCLs—*i.e.*, the decision making— we employ constraint programming techniques to optimize the selected configuration regarding aspects for providing a better user experience, such as the cost associated with the adaptation, the resources consumed or the QoS offered by the new configuration.

Finally, we validate our proposal with three case studies: *i)* a *Caching or Off-Loading* situation, where the application behavior is modified at runtime, *ii)* TRACK.ME, a platform for supporting tracking-based scientific experimentations and *iii)* DIGIHOME, a smarthome platform. These scenarios demonstrate the suitability of our approach when different kinds of devices, protocols and implementation technologies are involved in the adaptation process.

Résumé

Aujourd'hui, les environnements ubiquitaires font partie de notre vie quotidienne. À la maison, au travail, dans les voitures, dans les hôtels, les supermarchés et autres espaces publiques, nous rencontrons des technologies qui visent à rendre notre vie plus simple et plus facile d'une façon transparente. Durant ces dernières années, le potentiel de ces environnements a été de plus en plus exploité, notamment avec l'avènement et l'utilisation généralisée des smartphones. Ce type de dispositifs permet l'exécution d'applications qui ont la capacité de s'adapter parfaitement à l'état courant de l'environnement. De telles applications, appelées "applications sensibles au contexte", bénéficient de l'information du contexte et des services qui sont présents dans leur environnement pour améliorer et changer automatiquement leur comportement. Toutefois, ces adaptations nécessitent une intégration d'informations qui doit être effectuée en tenant compte de l'hétérogénéité en termes de dispositifs, de plateformes d'exécution, et de protocoles de communication ainsi que la mobilité des applications de sorte que les différentes responsabilités de l'adaptation peuvent être distribuées.

Pour faire face à ces défis, et compte tenu des limitations des solutions existantes, nous fournissons deux contributions majeures dans cette dissertation. Tout d'abord nous introduisons l'intergiciel **SPACES** comme une solution d'intégration des informations contextuelles et ensuite nous définissons le paradigme de "boucles de contrôle ubiquitaires" pour adapter les applications sensibles au contexte. En particulier, dans SPACES, nous définissons un méta-modèle inspiré du style architectural REST (REpresentational State Transfer) pour favoriser l'échange des informations contextuelles en tant que ressources, ce qui représente le fondement de notre approche. Ensuite, nous définissons les *connecteurs SPACES* pour modulariser les différents concepts et préoccupations identifiés par le méta-modèle. Ces connecteurs sont conçus en utilisant les principes de la *programmation orientée composant* et ils sont incorporés dans le modèle *Service Component Architecture* (SCA) pour être utilisé dans différents types d'applications, et ainsi indépendamment des applications sensibles au contexte.

Grâce à la définition de SPACES, nous sommes en mesure d'élaborer la seconde contribution de la dissertation—*i.e.*, les **boucles de contrôle ubiquitaires**. Inspiré par les concepts de l'*informatique autonome*, les boucles de contrôle offre la flexibilité nécessaire pour intégrer de nouveaux participants dans le processus d'adaptation (par exemple, des applications sensibles au contexte, des services et des systèmes existants) tout en fournissant un support pour la mobilité et l'intégration de nouveaux mécanismes de communication en cas de besoin. Dans le noyau des boucles de contrôle ubiquitaires—*i.e.*, la prise de décision— nous employons des techniques de programmation par contraintes pour optimiser la configuration courante de l'application en intégrant des critères qui garantissent une meilleure expérience à l'utilisateur final, tels que les coûts associés à l'adaptation, les ressources consommées ou encore la qualité de service offerte par la nouvelle configuration.

Enfin, nous validons notre proposition avec trois études de cas: Tout d'abord une *politique de Caching or Off-Loading*, dans laquelle le comportement de l'application est modifiée lors de l'exécution, ensuite TRACK.ME, une plateforme pour effectuer des expérimentations scientifiques et enfin DIGIHOME, une plateforme pour la création des maisons intelligentes. Ces scénarios démontrent la pertinence de notre approche lorsque différents types de dispositifs, des protocoles et des technologies de mise en œuvre sont impliqués dans le processus d'adaptation.

Chapter 1

Introduction

Contents

Motivation

Nowadays, computers are pervasive and resides in the periphery of our daily lives. This means that in different places we find technologies that make our lives simpler and we do not even notice. Cars that help people to reach the destination, smart houses that prevent domestic accidents, malls that provide customized advertisements on smartphones are some examples of this kind of places. These environments, called ubiquitous environments [Weiser, 1999, Weiss and Craiger, 2002], provide a great richness in terms of sensors giving information about the environment (*e.g.*, temperature, occupancy and light level), actuators that act on the environment (*e.g.*, closing the blinds, activating the alarm, and changing the temperature) and services (*e.g.*, for accessing multimedia content and querying the current magazine promotions). Therefore, computing power has became "invisible" and transparent for people.

In the last years, the potential of ubiquitous environments have been more exploited with the increasing usage of smartphones [Stefan Sidahmed, 2010]. These smart objects are powerful enough to interact with the environment and process information locally [Bellavista and Corradi, 2006]. In particular, smartphones execute applications that benefit from services and environmental information to improve their behavior [Schilit *et al.*, 1994]. This information, called **context information**, includes: *i)* physical conditions such as temperature, noise level and light level, *ii)* computational conditions, *e.g.*, battery level, bandwidth and available memory and, *iii)* social conditions, *i.e.*, where you are and who you are with [Schilit *et al.*, 1994, Chen and Kotz, 2000]. Consequently, the application exploiting this information are called context-aware applications. They require the information retrieval and processing for adapting them according to current environmental conditions. This context-aware applications search to provide a better user experience and thank to that they provide a competitive advantage in the business world [Beth Schultz, 2009]. Therefore, approaches enabling the adaptation of context-aware applications are required.

Software Adaptation. Nowadays, user and business requirements evolve constantly making modifications on software systems necessary. It would be possible to develop new systems from

scratch in order to satisfy the new requirements. However, in several cases, systems are complex, having different subsystems interconnected between them. Therefore, the development from scratch becomes an expensive task and unsuitable option. To deal with software changes, new ideas have emerged consisting in the *reuse* of systems parts by means of their *composition* and extension [McKinley *et al.*, 2004]. These ideas provide the foundations for software adaptation, which consists in the modification or extension of system structure and/or behavior improving its interaction with the environment [Kell, 2008]. For adapting applications and fostering reuse of their different parts, they have to satisfy two properties: *i)* provide a modularized architecture and *ii)* promote loose coupling between the different parts. Paradigms such as Component Based Design [Szyperski, 2002], Services Oriented Architectures (SOA) and Service Component Architecture (SCA) [OASIS Open CSA, 2007, Open SOA, 2007b] have emerged to achieve these properties. The former provides the notion of software components, which are software entities that can be implemented and deployed independently. Examples of component models include OpenCOM [Coulson *et al.*, 2008], OSGi Declarative Service [The OSGi Alliance, 2009], CORBA component model [Group, 2006a] and Fractal component model [Bruneton *et al.*, 2006b]. On the other hand, SOA aims to integrate applications that have been separately implemented, without rewriting them from scratch [Arsanjani, 2004]. Finally, SCA structures SOA applications by applying Component Based Design. The goal of this component model is to get the best of the two worlds—*i.e.*, the isolation of concerns and modularization from Component Based Design and the loose coupling and reuse fostered by SOA applications.

The adaptations achieved by means of the usage of these paradigms can be static or dynamic [McKinley *et al.*, 2004]. *Static adaptations* occur at development, compile, or load time, whereas *dynamic adaptations* take place at execution time. The dynamic adaptations can be triggered because of changes on social, physical or/and computational conditions— *i.e.*, regarding the context information. Context-aware Computing deals with this kind of adaptation.

Context-aware Computing. As stated in [Dey, 2001], context-awareness or context-aware computing consists in *"the use of context to provide relevant information and/or services to the user, where relevancy depends on the user's task."* 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"* [Dey, 2001]. From the previous definitions, we can see that there is an implicit notion of dynamic adaptation, where application functionality and/or behavior need to be modified according to the current context information. The applications that are tuned or changed regarding context are called *context-aware applications* [Cohen *et al.*, 2004] and the adaptation process taking place is the *context-based adaptation*. The context-based adaptation requires *context mediation* (also called *context integration* or *context exchange* in this dissertation). This context mediation consists in the collection and processing of usage data from different resources (*e.g.*, sensors, databases, network devices and Web Services), to compute pertinent indicators and deliver them to applications programs (*e.g.*, Web Services, enterprise applications and monitoring tools) [Coutaz *et al.*, 2005]. The realization of context-based adaptation requires to face several issues. First, the environments where context-applications are executed (*i.e.*, the already mentioned ubiquitous environments) constantly change. In particular, services and applications can join and leave at anytime. This makes necessary a flexible infrastructure [Zhu *et al.*, 2005] supporting volatility. Second, the responsibilities associated with adaptation can be distributed in order to benefit from the most powerful devices in the environment, requiring a clear isolation of these responsibilities. Finally, context mediation has to be done considering the diversity in terms of devices, development technologies, execution platforms and communication mechanisms. Thus, *heterogeneity*, *mobility* and *distribution of concerns* make the context-based adaptation a challenge.

Autonomic Computing. The previous issues associated with adaptation have to be faced in a transparent way for final users. In other words, the intervention from users in the adaptation process should be minimum or not required at all in the best case. The satisfaction of this feature

needs the design of applications able to make decisions by themselves. The *Autonomic Computing* [Ganek and Corbi, 2003] focuses in the development of this kind of applications. In particular, this computing paradigm conceives self-managing applications that are able to heal, protect and optimize by themselves. To achieve these properties, *Autonomic Computing* reifies the different phases that composes the adaptation process as *Feedback Control Loops*. These phases include the Monitoring and collection of information from the environment, the Analysis of such information and the determination of the goals to reach, the Planning of actions for achieving the goals, and the Execution of these actions.

The different responsibilities associated with Feedback Control Loop phases can be located at the application or middleware levels, or divided between both levels. In particular, the functionality required for retrieving and processing the context information can make part of the application itself. This means that the application developer focuses not only in the business logic but also in the context integration concern. On the other hand, all the adaptation responsibilities can rest on the middleware level. In this case, developers only have to provide the required flexibility points enabling the modifications on applications at runtime. An intermediary solution consists in dividing the responsibilities between the two levels. In this solution, the middleware deals with the distribution issues and makes the context available for applications adapt according to it. Regarding these alternatives of dealing with context-based adaptation, in this dissertation we choose to provide a middleware approach encapsulating the feedback control loops concerns. This selection is motivated because we consider that the development of context-aware applications has to focus in the business logic and the flexibility points enabling changes on the application. By isolating the feedback control loops associated with applications in the middleware level, we achieve this goal.

Chapter Organization

The rest of this introductory chapter is organized as follows. We start by presenting the problematic that we tackle (cf. Section 1.1) and then we discuss our research goals (cf. Section 1.2). We conclude the chapter with an overview of the dissertation's contributions (cf. Section 1.3) and a roadmap to assist the reader in browsing the document (cf. Section 1.4).

1.1 Understanding the Problem

In this dissertation, we deal with the context-based adaptation and the implicit context mediation associated with it. In particular, we have identified that approaches facing such kind of adaptation are confronted to the following issues:

1. **Hardware Heterogeneity**: In general, context-aware applications and services[1] run on devices (*e.g.*, laptops, servers, smartphones, and sensors) with different kinds of computational resources (*e.g.*, processor, memory and storage capabilities). These aspects restrict the way in which the different tasks associated with adaptation can be distributed. This means that most powerful devices will take responsibility for the processing load and the tiny devices will be limit to simpler tasks such as information provision.

2. **Software Heterogeneity**: The diversity is not only present at the hardware level. We also find heterogeneity in terms of operating systems (Mac OS X, Windows, Linux, Android, etc.), execution environments (Java, .NET framework, etc.) and development technologies (Java, C++, Smalltalk, etc.). This heterogeneity makes the interoperability between service providers and consumers difficult and therefore limits the benefits that can be obtained from ubiquitous environments in terms of computational resources.

[1] In this dissertation we make the distinction between applications and services. For us, a service is a specific functionality available in the environment (*e.g.*, actuators allowing the control of home appliances in smart houses such as blinds, TVs and air conditioner) and that can be remotely accessed by applications. Consequently, these applications, and specifically context-aware applications, benefit from environment in order to help users in the execution of specific tasks.

3. **Heterogeneity of Protocols**: In ubiquitous environments several standards are employed for exchanging information and locating available services. This means that different providers select the most suitable communication mechanisms according to their needs. No consensus exists about the protocols that should be applied in different situations. Context-aware applications that only interact via specific protocols limit the user satisfaction degree that they are able to provide. This restriction make a flexible mechanism allowing them to use different kinds of interaction solutions necessary.

4. **Heterogeneity of Services and Context Providers**: This heterogeneity results from the software and protocol heterogeneity. In other words, we find services implemented with different technologies and using several mechanisms to offer their functionalities. Furthermore, the services can provide complex APIs following a function-oriented approach that assumes that all clients apply the same model and promotes a high coupling between providers and consumers. In the particular case of context providers, the diversity means context information with different representations, syntax and semantic. Once again, this heterogeneity restricts the exploitation of resources in the environment.

5. **Mobility of Devices**: In ubiquitous environments the devices running context-aware applications and services can move from one location to another. In this way, the ancient distributed programming model that assumes a static infrastructure, in which the different elements are stable, is not suitable. Therefore, the approach used to build this kind of applications should promote loose coupling between applications and services as well as the possibility of dealing with spontaneous communications.

6. **Distribution of Adaptation Concerns**: The changing nature of ubiquitous environments and the heterogeneity in terms of devices make the distribution of the adaptation concerns necessary. This means that the context gathering and processing as well as the identification of required configurations and the execution of the adaptation itself can be spread between different entities. With this distribution, we need to face the different kinds of heterogeneity and the mobility already mentioned. Therefore, a global approach allowing adaptation of context-aware applications and considering these issues is required.

For dealing with the previous issues, today, we can find several middleware solutions [Bellavista *et al.*, 2003, Ranganathan *et al.*, 2004, Debaty *et al.*, 2005, Román *et al.*, 2002a] providing support for context-based adaptation. However, most of them are difficult to reuse in different situations because they are not easily configurable and customizable. In general, the existing solutions lack flexibility in terms of communications, context information representations and modularization of the adaptation concerns. Moreover, they are not suitable to work on devices with restricted capabilities (*e.g.*, smartphones) and fail to integrate existing legacy applications.

The approach introduced by this dissertation deals with the issues associated with context-based adaptation. In particular, we consider this kind of adaptation as a process, where the different participants are heterogenous in terms of capabilities and can integrate the process at runtime. More concretely, we provide a middleware solution integrating different entities, which is based on simple and well accepted standards and paradigms. Then, we incorporate this integration mechanism into the adaptation process, which we defined by exploiting concepts from Feed Back Control Loops [Hariri *et al.*, 2006a, Parashar and Hariri, 2005] and constraint programming techniques [Apt, 2003]. This results in the definition of *"Ubiquitous Feedback Control Loops"* (Ubiquitous FCLs), which present the flexibility required to easily integrate new participants in the adaptation process (*e.g.*, context-aware applications, services and legacy systems). Our Ubiquitous FCLs support mobility, the incorporation of new communication mechanisms when required and the optimization of the selected application configuration. In the following section we discuss the goals associated with our approach.

1.2 Goals of this dissertation

As explained in the previous section, the development of context-aware applications implies to consider several issues associated with heterogeneity, dynamism and mobility. Therefore, the main goal of this dissertation is to **provide an approach for context-based adaptation by integrating heterogenous entities and resources in ubiquitous environments**. To achieve this, we decompose this goal in the following subgoals:

1. **Consider what exists:** We will leverage our solution on existing standards and paradigms. This choice is motived because we want to propose a simple but at the same time complete solution that can be applied in different context-aware adaptation situations. In practice, we find several paradigms and technologies that have proved their suitability in the design and implementation of applications. Paradigms such as the component-based design, SOA architectures and the REpresentational State Transfer (REST) [Fielding, 2000] are widely applied for building reusable applications and services. In particular, we can exploit the concern isolation (*i.e.*, the separation between business logic and non-functional requirements) and modularization fostered by the Component Design, the loose coupling and reuse promoted by SOA and the data centric vision from REST. Consequently, standard protocols such as HTTP, UPnP [UPnP Forum, 2008] and SLP [Guttman *et al.*, 1999] are applied in the industry as well as in research. The usage of such protocols in communications fosters the integration with already existing services and enables a better exploitation of the environment. Therefore, we pretend to benefit from the advantages provided by these technologies and paradigms in order to conceive our approach.

2. **Provide a simple and flexible solution**: The cornerstone of our proposal are *simplicity* and *flexibility*. By simple we mean easy to understand and apply. The application of standards help us in this purpose. The flexibility refers to the possibility of modifying and extending the approach according to developer needs as well as its usage in different situations. The fulfillment of this property rests on the analysis and study of the different elements having a role in context-based adaptation and in the already mentioned usage of existing paradigms. Thus, our objective is to provide a solution easy to use and flexible enough to be extended when required.

3. **Identify relevant elements in context mediation**: The integration of context becomes a key issue when we consider the realization of context-based adaptation. Therefore, before building a suitable solution enabling such kind of adaptation, we need to identify the different concepts and elements impacting the exchange. The identification of these elements will allow us to conceive a mechanism for context mediation to deal with the typical adaptation distribution concerns in ubiquitous environments.

1.3 Contribution

In order to provide a better understanding of this dissertation, in this section we present an overview of the contributions. In short, our work focuses on the domain of ubiquitous computing for the purpose of making the adaptation of context-aware applications easier. Figure 1.1 summarizes our proposal. As observed, our two main contributions—*i.e.*, *SPACES* and *Ubiquitous FCLs* provide solutions for context mediation and context-based adaptation, respectively. Below we discuss these contributions.

1. **SPACES**: Our middleware approach fostering the notion of context in order to enable context integration in ubiquitous environments. SPACES makes this possible by defining:

   (a) **The SPACES Metamodel:** We define a metamodel that clearly states the relationships between relevant concepts and elements that make part of the context information exchange. In particular, this metamodel considers context information as resources

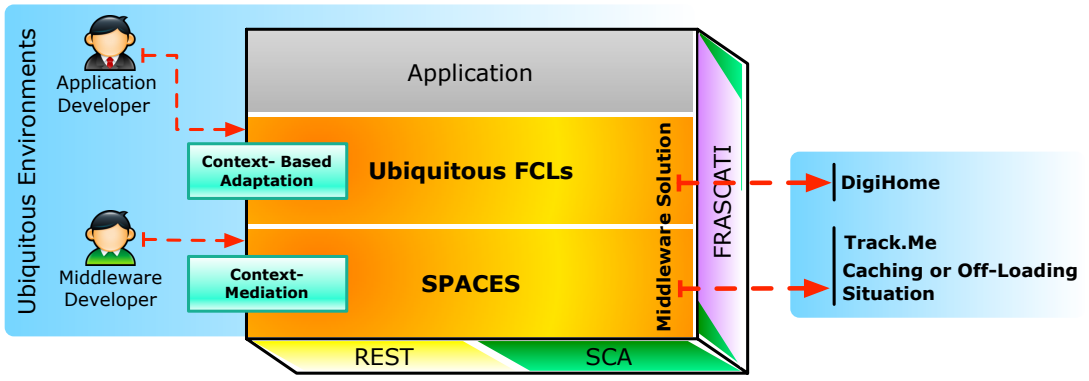


Figure 1.1: Approach for Building Context-Aware Solutions

that can be accessed by different kinds of clients and using different communication mechanisms. The metamodel reifies relevant elements in context mediation helping us to conceive a solution for context integration.

(b) **The SPACES Connectors**: In order to modularize the context exchange concerns, we define SPACES connectors (also called *Ubiquitous Connectors* in this dissertation) enabling the integration of participants in context exchange. These connectors exploit standards paradigms for supporting the notion of context as a resource defined by the metamodel. They foster simple access of context resources by means of uniform interfaces and identifiers and providing flexibility in terms of representations. Furthermore, SPACES connectors support the discovery of resources considering aspects such as the Quality of Context (QoC) [Razzaque *et al.*, 2006, Sheikh *et al.*, 2008, Manzoor *et al.*, 2008, Bu *et al.*, 2006, Buchholz *et al.*, 2003].

(c) **The SPACES Architecture**: We propose a generic architecture respecting the principles stated by the ubiquitous connectors in terms of the context information integration. The defined architecture is reified by exploiting a component model that combines CBSE and SOA principles.

2. **Ubiquitous Feedback Control Loops (Ubiquitous FCLs)**: By applying autonomic computing principles [Ganek and Corbi, 2003, Kephart and Chess, 2003, Hariri *et al.*, 2006b, Kephart and Chess, 2003, Menascé and Kephart, 2007], we define *Ubiquitous Feedback Control Loops* (Ubiquitous FCLs). These FCLs support the adaptation of applications based on the context information by exploiting SPACES connectors. The volatility of adaptive applications and services is considered by the Ubiquitous FCLs in order to provide flexibility in the adaptation process. In this kind of loops, the adaptation decisions (that correspond to analysis and planing phases in autonomic computing) are delegated to the most powerful entities in the environment. However, the flexibility of these Ubiquitous FCLs enables the definition of Local FCLs on the devices hosting the adaptive applications. These local loops are executed when problems appear in the global ones.

   As part of Ubiquitous FCLs, we define an ***strategy based on constraint programming techniques*** [Apt, 2003] ***for determining the new application configuration***. The new configuration is stated considering that applications have flexibility points, which are used for bringing new functionality into them. This functionality can be implemented by one or several components. Once the potential configurations are estimated according to the context changes, we select an optimized configuration regarding dimensions such as reconfiguration costs and resource consumption. In this process, we consider the different dependencies existing between the flexibility point implementations. Therefore, the application configuration at the end of the FCL execution is valid but is also the most suitable for the

current conditions and expectations of the costumer.

3. **Evaluation:** The contributions are evaluated by means of three case studies: *i) A Caching Off-Loading Situation* illustrating the suitability of our solution in terms of context ingration, *ii)* TRACK.ME, a service-oriented platform for tracking activities of mobile users and *iii*) DIGIHOME, an smart platform enabling adaptation in home environments. Concretely, we measure the performance of our approach when there are adaptation situations and context exchange involved.

1.4 Dissertation Roadmap

The dissertation is divided in four parts. The first part encloses the State of Art. The second part presents the contribution of this dissertation, and the third one the validation of our proposal. Finally, the last part includes the conclusions and perspectives of this dissertation. Below we present an overview of the chapters that compose the different parts.

Part I: State of the Art

Chapter 2: Concepts and Background

In this chapter we discuss concepts that make part of the foundation of our proposal. In particular we discuss two kinds of approaches: *i)* general approaches for dealing with the conception of service-oriented solutions and, *ii)* specific approaches for service discovery in ubiquitous environments. General approaches include the architectural styles such as Representational State Transfer (REST) [Fielding, 2000] and SOAP [W3C, 2007]. In this kind of approaches we also introduce the Service Component Architecture (SCA) [OASIS Open CSA, 2007, Open SOA, 2007b, SCOrWare Project, 2007] and the OSGi Framework [The OSGi Alliance, 2009], which are used in the development of middleware platforms. Associated with these component models, we discuss some platforms supporting the execution of applications based on the SCA component model. In the specific approaches we introduce some Service Discovery Protocols (SDPs) enabling spontaneous communications.

Chapter 3: Ubiquitous Approaches

Chapter 3 provides an overview of middleware solutions dealing with context-aware application issues. This chapter also includes some solutions for service discovery in ubiquitous environments and data-oriented architectures enabling context mediation. The main issues associated with the presented approaches are described. The chapter also specifies concepts related to context-awareness that will be used throughout the dissertation.

Chapter 4: Autonomic Computing Approaches

In this chapter we provide an overview of the main concepts of Autonomic Computing. We also discuss the relationships between context-awareness and autonomic computing, which can be considered as complementary approaches. Several autonomic solutions dealing with context adaptation are presented as well as their main issues. At the end of the chapter we summarize the challenges faced by the dissertation.

Part II: Contribution

Chapter 5: Enabling Context Mediation In Ubiquitous Environments

Throughout this chapter we discuss the first three contributions of this dissertation: *i) SPACES metamodel*, *ii) SPACES connectors* and *iii) the SPACESarchitecture for context mediation*. Before introducing the contribution, we define the properties expected in our solution in order to satisfy the goals identified in the introduction. We finish the chapter by integrating our solution into the SCA Component Model in order to foster its reuse.

Chapter 6: Building Ubiquitous Feedback Control Loops

Chapter 6 presents the contributions associated with the approach for context based adaptation (called *Ubiquitous Feedback Control Loops*) and the strategy for selecting the new configuration. The Ubiquitous FCLs are conceived by considering the adaptation as a process where the different tasks are associated to autonomic computing responsibilities.

Part III: Validation

Chapter 7: Case Studies

This validation chapter presents three case studies used to provide a qualitative and quantitative evaluation of our contribution. The first case study is a *caching off-loading situation* requiring the context information collection in order to decide the parametrization of mobile applications. In the second one we exploit TRACK.ME, a distributed platform that respect the principles of the cloud computing [Grossman, 2009]. The last case study defines DIGIHOME, a platform that enables the adaptation of mobile applications for controlling home appliances as well as changes in room configuration.

Part IV: Conclusions and Perspectives

Chapter 8: Conclusions and Perspectives

In the last chapter we provide a summary of the contribution discussed in the dissertation. We focus in the advantages and provide an overview of the limitations. We conclude the chapter and the dissertation by describing the identified research directions associated with the contribution.

1.5 Publications

Below we cite the different publications associated with the dissertation.

International Journals

- Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, and Frank Eliassen. *The Digihome Service-Oriented Platform*. Softw. Pract. Exper., 2011 (*To appear*) [Romero *et al.*, 2011]

- Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. *A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures*. Softw. Pract. Exper., 2011 (*To appear*) [Seinturier *et al.*, 2011]

- Frédéric Loiret, Romain Rouvoy, Lionel Seinturier, Daniel Romero, Kevin Sénéchal, Ales Plsek. *An Aspect-Oriented Framework for Weaving Domain-Specific Concerns into Component-Based Systems*. Journal of Universal Computer Science (J.UCS) (*To appear*) [Loiret *et al.*, 2010a]

International Conferences

- Daniel Romero, Romain Rouvoy, Lionel Seinturier, and Frédéric Loiret. *Integration of Heterogeneous Context Resources in Ubiquitous Environments*. In Michel Chaudron, editor, Proceedings of the 36th EUROMICRO International Conference on Software Engineering and Advanced Applications (SEAA´10), page 123-126, Lille France, 2010. ACM [Romero *et al.*, 2010d]

- Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, and Frank Eliassen. *RESTful Integration of Heterogeneous Devices in Pervasive Environments*. In Proceedings of the 10th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS´10), volume 6115 of LNCS, pages 1-14. Springer, June 2010 [Romero *et al.*, 2010a]

- Daniel Romero, Romain Rouvoy, Lionel Seinturier, and Pierre Carton. *Service Discovery in Ubiquitous Feedback Control Loops*. In Proceedings of the 10th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS´10), volume 6115 of LNCS, pages 113-126. Springer, june 2010 [Romero *et al.*, 2010b]

Book Chapters

- Amirhosein Taherkordi, Daniel Romero, Romain Rouvoy and Frank Eliassen. *RESTful Service Development for Resource-constrained Environments*. In "REST: From Research to Practice", Springer (*To appear*) [Taherkordi *et al.*, 2011]

- Daniel Romero, Romain Rouvoy, Lionel Seinturier, Sophie Chabridon, Denis Conan, and Nicolas Pessemier. *Enabling Context-Aware Web Services: A Middleware Approach for Ubiquitous Environments*. In Michael Sheng, Jian Yu, and Schahram Dustdar, editors, Enabling Context-Aware Web Services: Methods, Architectures, and Technologies. Chapman and Hall/CRC, 05 2010 [Romero *et al.*, 2010c]

Workshops

- Rémi Mélisson, Daniel Romero, Romain Rouvoy, and Lionel Seinturier. *Supporting Pervasive and Social Communications with FraSCAti*. In 3rd DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services, Amsterdam, The Netherlands, 06 2010 [Mélisson *et al.*, 2010b]

- Daniel Romero, Carlos Parra, Lionel Seinturier, Laurence Duchien, Rubby Casallas. *An SCA-based middleware platform for mobile devices*. In Middleware for Web Services (MWS 2008) at EDOC2008 , Munich, Germany, 2008 [Romero *et al.*, 2008]

Tool Demonstrations

- Rémi Mélisson, Philippe Merle, Daniel Romero, Romain Rouvoy, and Lionel Seinturier. *Reconfigurable Run-Time Support for Distributed Service Component Architectures*. In Automated Software Engineering, Tool Demonstration, pages 171-172, Antwerp Belgique, 09 2010 [Mélisson *et al.*, 2010a]

Electronic Magazines

- Daniel Romero. *Context-Aware Middleware: An Overview*. In Revista Electrónica Paradigma en Construcción de Software , Bogota, Colombia, 2008 [Romero, 2008]

Part I

State of the Art

Chapter 2

Concepts and Background

Contents

State-of-the-Art Presentation

In order to discuss different technologies, paradigms and proposals associated with this dissertation, we have divided the presentation of state of the art in three different chapters. In the first one (*Concepts and Background*), we discuss different elements and concepts applied in our proposal, including the Representational State Transfer principles, the Service Component Architecture model and Service Discovery Protocols. These elements represent the foundations of our approach. In the second chapter (*Ubiquitous Approaches*) we introduce some basic concepts of context-awareness as well as middleware solutions dealing with context-based adaptation and its associated problems. Some of the presented approaches exploit the concepts introduced in the *Concepts and Background* chapter, *e.g.*, service oriented architectures and discovery elements. The last chapter (*Autonomic Computing Approaches*) provides an overview about autonomic concepts and introduces some solutions that also face the problems related to context-aware applications by applying such paradigm.

Chapter Organization

In this chapter we introduce concepts that make part of the foundation of our proposal. In particular we discuss two kinds of approaches: *i)* general approaches for dealing with the conception of service-oriented solutions and, *ii)* specific approaches for discovery in ubiquitous environments. General approaches include the architectural styles enabling the creation of APIs services (cf. Section 2.1) and component models for the development of middleware platforms (cf. Section 2.2). Associated with these component models, we discuss some platforms supporting the execution of applications based on the SCA component model (cf. Section 2.3). On the other hand, in the specific approaches we introduce the Service Discovery Protocols (cf. Section 2.4) enabling spontaneous communications. Furthermore, we specify and justify our selection between the different discussed approaches. Finally, we summarize the different elements and choices described in the chapter (cf Section 2.5)

2.1 Approaches for the Integration of Information

In this section, we discuss and compare some existing paradigms employed for dealing with application heterogeneity and therefore that we can exploit in our solution for dealing with context integration. In particular, we consider the Web Services, which have the objective of enabling interoperability between applications. We find two main approaches for the conception of this kind of services [Przybilski, 2005]: SOAP (cf. Section 2.1.1) and the REST architectural style (cf. Section 2.1.2). In next sections we present them.

2.1.1 SOAP

SOAP [W3C, 2007] provides an extensible messaging framework based on XML technologies for exchanging information in a structured and decentralized way. The framework has been conceived to be independent from programming models and implementation specific semantics.

According to the specification [W3C, 2007], SOAP fosters *simplicity* and *extensibility* by providing a minimal messaging framework. This means that SOAP does not consider features found in distributed systems such as *reliability*, *security*, *correlation* and *routing*. However, the framework can be extended with new specifications to support such features.

In SOAP, messages are exchanged between SOAP nodes representing information consumers or producers. Typically a SOAP message is composed by an *envelope*, *headers*, and *body*. Figure 2.1 depicts this structure. The SOAP envelope encloses the headers and body. The headers, which are optional, provide a mechanism for extending SOAP messages in a decentralized and modular way. For its part, the body contains the current information being exchanged and it must be structured following the XML 1.0 rules.

In SOAP, the messages can be exchanged using different *Message Exchange Patterns* (MEPs) such as one-way messages, request/response interactions, and peer-to-peer conversations. These MEPs are incorporated into the messaging framework as SOAP features, which represent extensions to the framework.

Additionally, the SOAP framework enables the message transmission by employing different protocols. The underlaying protocols are stated by means of binding specifications. These specifications define how the contract are respected and how the potential failures can be handled. The contracts are formed by the features being employed.

Web Services

The Web Services (WSs) provide an standard for enabling the interoperability between software applications running on the top of a variety of platforms and frameworks [Booth *et al.*, 2004]. Figure 2.2 (taken from [Booth *et al.*, 2004]) illustrates the different elements employed in the Web Service conception. In particular, WSs use the framework provided by SOAP as well as the

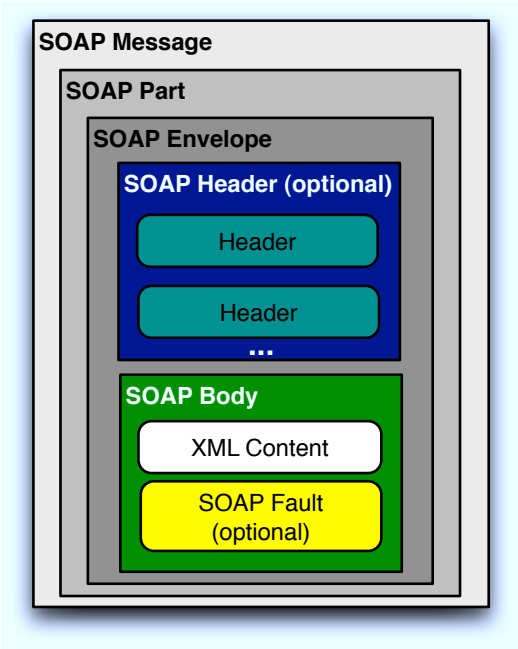


Figure 2.1: SOAP Message Structure

underlying protocols for the message exchange. The SOAP selection is motived because of the extensibility capabilities that it promotes.

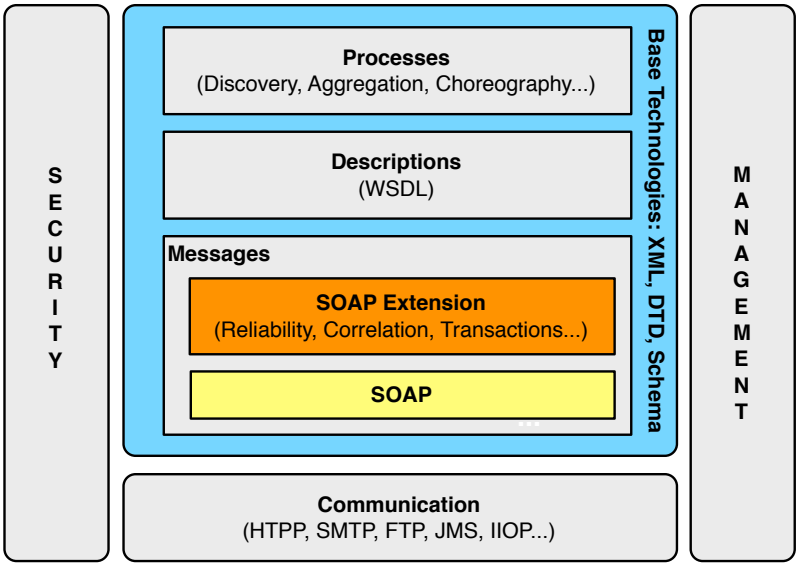


Figure 2.2: Web Service Architecture Stack

Additionally, the WSs employ the XML technologies for defining the Web Service Description Language (WSDL). This language is employed for describing the functionalities exposed by the service provider or agent. These functionalities can be advertised using a registry (called UDDI)

that the potential clients can access for retrieving the WSDLs. Complementary specifications exist for dealing with aspects such as security (WS-Security), transactions (WS-Transaction), reliable message delivering and service orchestration (BPEL).

2.1.2 REpresentational State Transfer (REST)

REpresentational State Transfer (REST) is a resource-oriented software architecture style identified by R. Fielding for building Internet-scale distributed applications [Fielding, 2000]. Typically, the REST triangle defines the principles for encoding (*content types*), addressing (*nouns*), and accessing (*verbs*) a collection of *resources* using Internet standards. Resources, which are central to REST, are *uniquely addressable* using a universal syntax (*e.g.*, a URL in HTTP) and share a *uniform and simple interface* for the transfer of application states between client and server (*e.g.*, `GET/POST/PUT/DELETE` in HTTP). REST resources may typically exhibit multiple typed representations using, for example, XML, JSON, YAML, or plain text documents. Thus, RESTful systems are loosely-coupled systems which follow these principles to exchange application states as resource representations. This kind of stateless interactions improves the resources consumption and the scalability of the system.

According to R. Fielding [Fielding, 2000], "*REST's client-server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.*" Below we summarize the key REST properties:

1. *Simplicity*: REST is based on standards and enables interaction by defining uniform and simple interfaces.
2. *Lightness*: The stateless interaction, the self-descriptive messages and the possibility of using Web Intermediaries reduce the charge supported by the interacting entities.
3. *Reusability*: The loosely-coupling between entities and the application of Web Intermediaries foster the reusability.
4. *Extensibility*: The exchanged messages can be extended by adding new headers without loosing backward compatibility.
5. *Flexibility*: In terms of resource representations and the logical URLs definition to access the resources.

RESTful Web Services

The use of REST for designing Web Services begins to be widely accepted because of the loosely-coupling and the easy deployment of services (thanks to the usage of standards) conceived in this way [Alarcón and Wilde, 2010]. These services are called RESTful Web Services [Richardson and Ruby, 2007, Tyagi, 2006, Rodriguezi, 2006]. In REST, the RESTful Web Services are considered as resources accessible by means of standards URLs and expressing the supported operations on these resources by means of simple interfaces. The main advantages with the RESTful Web Services are traduced in a low learning curve for consumers and a low support overhead for producers [Cowan, 2005]. Thus, RESTful services represent an attractive alternative to the traditional Web Services created by applying SOAP and WSDL (cf. Section 2.1.1).

The boom of REST as a Web Service design model has required the conceptions of languages and metamodels supporting the description of RESTful Web Services such as *Web Application Description Language* (WADL) [Hadley, 2006]. In particular, as it will be presented in Chapter 5, in

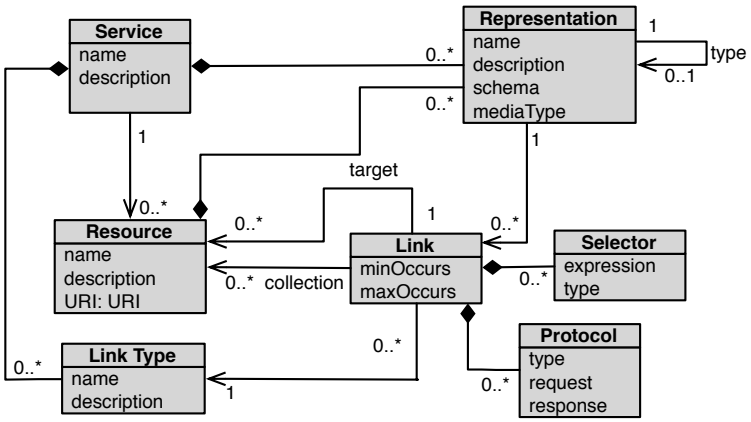


Figure 2.3: RESTful Service Metamodel

this dissertation we exploit the REST metamodel defined by [Alarcón and Wilde, 2010] for supporting the notion of context as a resource. Figure 2.3 depicts this metamodel. In the metamodel, a `Service` can offer several `Resources`, which can provide the URI patterns for the unique identifiers. The `Resources` have 0 or more `Representations` expressing the encoding syntax to exchange the resources. The `Representations` can contain `Links` to other target resources, which are retrieve by means of `Selectors`. These `Selectors` are defined depending on the concrete representation. For example, if XML is selected as representation, the *XML Path Language* (XPath) can be used for specifying the `Selectors`. The `Links` also have `LinkTypes` associated, which represent the type of the link. Finally, the `Links` are defined according to the rules stated by *Protocols*.

2.1.3 Integration via SOAP Framework vs. Integration via the REST Architectural Style

SOAP and REST provide elements enabling the conception of services accessed and provided by heterogenous entities. The strong points of SOAP are the extensibility, the usage of standards and the rigorous specification of different issues typical in distributed systems. However, SOAP tends to be complex and not very flexible because it imposes the usage of XML technologies. In SOAP, the usage of a service requires the knowledge of the different method names and protocols supported for the service access. This knowledge is necessary because SOAP is a messaging framework providing the possibility of choosing between several protocols for the message exchange. Therefore, SOAP introduces a high dependency between service consumer and provider. As already stated in the introduction of this dissertation, the ubiquitous environments are characterized by the presence of devices that have different resources and capabilities. Thus, the imposition of an specific model for exchanging the context information with a high coupling between interacting entities will limit the applicability of the solution.

On the other hand, REST simplicity, flexibility and resource centric nature enable the easy development of services. These services can be exposed by means of standard and simple protocols and a generality of interfaces. Unlike SOAP, REST makes the interaction of clients and servers possible without any further configuration, when application protocols such HTTP are employed. This interaction focuses in the data, which can be represented in several ways. Furthermore, the loose-coupling promoted by self-contained messages fosters the reuse of web intermediaries. Thus, the REST versatility in terms of resources dissemination makes it a suitable option for exchanging the context information in ubiquitous environments.

2.2 Component Models for SOA Applications

In this section we present two component models for the development of SOA applications. In particular, we focus in the OSGi framework (cf. Section 2.2.1) and the SCA Component Model (cf. Section 2.2.2).

2.2.1 OSGi Framework Specification

OSGi Framework [The OSGi Alliance, 2009] provides an open and dynamic component model, based on Java, for the service development, deployment and management. In OSGi, the applications, called *bundles*, can be dynamically deployed and updated. Bundles are JAR files containing Java classes and the required resources for providing functionalities to end users. The different bundles can share functionalities between them.

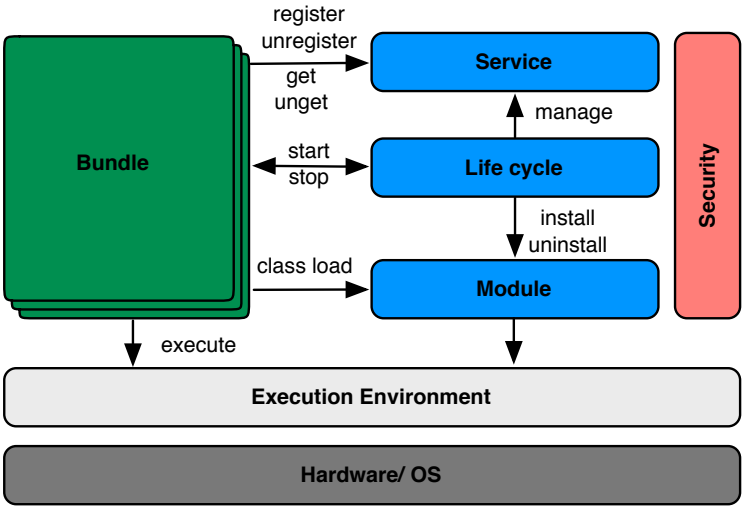


Figure 2.4: OSGi Layers

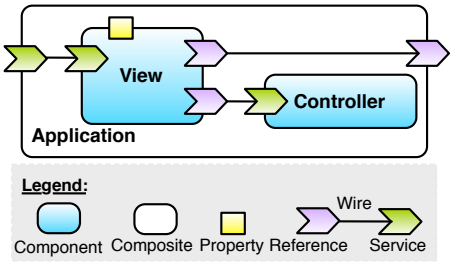
The functionalities of OSGi are organized in several layers, which are depicted in Figure 2.4 (taken from [The OSGi Alliance, 2009]). Below we discuss briefly the different layers.

1. *Security Layer*: This layer extends the Java 2 Security Architecture specification in order to limit the functionalities provided by bundles to pre-defined capabilities.
2. *Module Layer*: Establishes rules for the dependency declaration of bundles. In other words, the layer states how the bundles can import and export functionalities.
3. *Life Cycle Layer*: Provides a life cycle API to manage the bundles. In particular, this layer specifies how the bundles are started, stopped, installed, uninstalled and updated.
4. *Service Layer*: Defines a dynamic programming model that simplifies the bundle development. The proposed model decouples the service's specification (Java interface) from its concrete implementations. The concrete implementation of services can be selected at runtime.

2.2.2 Service Component Architecture (SCA) Model

SCA [Open SOA, 2007b] is a set of specifications for building distributed applications based on SOA and *Component-Based Software Engineering* (CBSE) principles [SCOrWare Project, 2007]. In

SCA, the basic construction blocks are the *software components*, which have *services* (or provided interfaces), *references* (or required interfaces) and exposed properties. The references and services are connected by means of *wires*. SCA also specifies a hierarchical component model, which means that components can be implemented either by primitive language entities or by subcomponents. In the latter case the components are called *composites*. Figure 2.5 provides a graphical notation for these concepts as well as a XML-based assembly language to configure and assemble components.



Application.composite

```
<composite name="application.composite">
  <service name="run" promote="View/run"/>
  <component name="View">
    <implementation.java class="app.gui.SwingGui"/>
    <service name="run">
      <interface.java interface="java.lang.Runnable"/>
    </service>
    <reference name="control" target="Controller/control">
      <interface.java interface="app.ControllerService"/>
    </reference>
    <property name="orientation">landscape</property>
  </component>
  <component name="Controller">
    <implementation.java class="app.ctrl.Controller"/>
    <service name="control">
      <interface.java interface="app.ControllerService"/>
    </service>
  </component>
</composite>
```

Figure 2.5: SCA Graphical Notation and Assembly Language

SCA is designed to be independent from programming languages, *Interface Definition Languages* (IDL), communication protocols and non-functional properties. In this way, an SCA-based application can be built, for example, using components in Java, PHP, and COBOL. Furthermore, several IDLs are supported, such as WSDL and Java Interfaces. In order to support interaction via different communication protocols, SCA provides the concept of *binding*. For SCA references, *bindings* describe the access mechanism used to call a service. In the case of services, the bindings describe the access mechanism that clients use to call the service.

2.2.3 Choosing the Component Model

The component model to be employed in our proposal has to be extensible, provide a clear concern separation and support for heterogeneity. The discussed component models have advantages that can be exploited in ubiquitous environments. In the case of the component model defined by OSGi, its main strong points are the dynamic deployment and update of the installed applications. In SCA, the concerns modularity and the distribution support represent relevant characteristics of the model. The two component models have desirable properties for our solution and they could be combined to get the both of them.

Because SCA promotes the integration of different technologies, including OSGi [SOA, 2007], we choose this component model. In this way we can obtain the advantages of the two component models if required. The selection of SCA is also motivated by the separation of bindings and policies from business logic, which foster the extensibility of the model, as well as by the native support for distribution. Furthermore, by using SCA, it is easier to integrate legacy applications and develop new services. However, SCA does not address the runtime management of the application, which typically includes monitoring and reconfiguration.

In the next sections we focus on platforms for the selected component model, (*i.e.*, SCA) and how some of them complement the SCA component model for dealing with runtime management issues. The discussion of platforms for OSGi such as Apache Felix [Foundation, a], Eclipse Equinox [Foundation, b] and Knopflerfish [Project,] will be not discussed in the dissertation.

2.3 SCA Platforms

In this section we present some open source platforms for executing SCA-based applications. In particular we focus the discussion in Fabric3 (cf. 2.3.1), Tuscany (cf. 2.3.2) and the FraSCAti platform (cf. 2.3.3).

2.3.1 The Fabric3 Platform

Fabric3 [Systems, 2010] is an open source platform for developing SCA applications. The platform offers ***dynamic reconfiguration capabilities***, ***high availability and reliability capabilities*** and, ***extensibility capabilities***. These capabilities are explained below.

1. In terms of *dynamic reconfiguration*, Fabric3 can be extended in order to provide introspection of applications. Moreover, it is possible to update at runtime the wires between the applications. The policies associated with non-functional services are applied at development time, deployment or during the execution.

2. Fabric3 runtimes are organized by domains. They can be embedded (*e.g.*, Ant, Maven or an IDE), consist of a single Virtual Machine (VM) or span multiple clusters. Figure 2.6 depicts the architecture of a Fabric3 domain. The `Controller` monitors and manages the available services. The `Zones` are clusters containing the different runtimes that participate in the domain. The *high availability* in Fabric3 leverages on this Zone concept. When a runtime crashes, it is synchronized with the `Zone`.

3. Concerning the *reliability*, the platform provides it at two levels: application-level and runtime-level. The former provides support for the *Java Transaction API* (JTA) [Microsystems, 2005]. The latter is based on a compensation model.

4. Fabric3 provides a kernel that can be *extended* with additional functionalities. Each runtime contains a local domain that has the basic functionalities. During the execution, it is possible to deploy extensions or contributions, which are implemented as components. Such contributions are loaded in separate classloaders by means of OSGi. Besides providing contributions isolation reducing conflicts, the usage of OSGi allows versioning.

Summary. The Fabric3 platform provides attractive functionality in terms of availability and reliability, which are relevant issues in the development of distributed system. Furthermore, the platform supports extensions for dealing with the dynamic modification of SCA wires and policies by means of OSGi.

2.3.2 The Tuscany Platform

The TUSCANY Platform [Foundation, 2010, Laws *et al.*, 2010] provides an infrastructure for development and management of SCA applications. Besides supporting the component implementations based on OSGi, TUSCANY is able to run in both non-OSGi and OSGi environment. This property makes the developing, building, launching, running and testing of Tuscany easier. The platform offers a modular and pluggable architecture enabling its customization. Figure 2.7 shows this architecture. The different elements that compose it are discussed below:

1. *Composite Applications*: Represents the business applications running on the top of the TUSCANY runtime. Of course, these applications are described using the assembly language stated by the SCA specification.

2. *SCA API and* TUSCANY *API*: Situated between the applications and the runtime, the SCA API enables applications for interacting with the platform. The API is language specific. TUSCANY provides a version for the Java language. Furthermore, the platform extends the SCA API via the TUSCANY API.

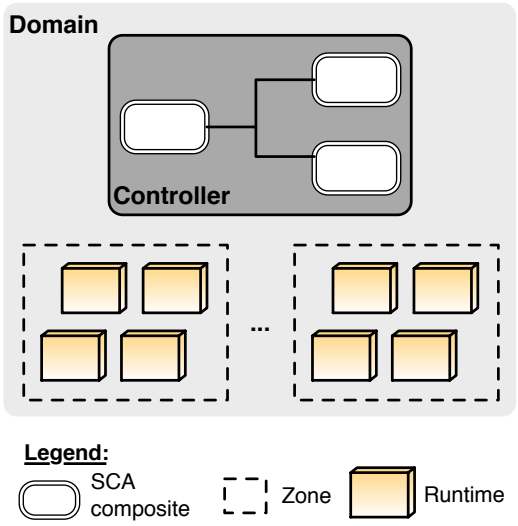


Figure 2.6: Domains in the Fabric3 Platform

3. *Core*: Supports the component instantiation, the assembly of component into composite applications and the management of the resulting applications.

4. *Extensions*: The platform provides plug points for allowing the incorporation of new functionalities. In particular, it is possible to add new bindings, implementation types, policies, interfaces and data bindings. The first four correspond to the SCA concepts discussed in Section 2.2.2. The data bindings provide support for different formats of the information exchanged between services. Conversions between formats are supported in a transparent way.

5. *Contribution Service SPI*: Allows the easy implementation of new extensions of the platform.

6. *Hosting Platforms*: These modules provide the possibility of executing the TUSCANY runtime on the top of different hosting platforms such as Apache Tomcat and Geronimo. TUSCANY can be extended to include others execution environments.

Summary. TUSCANY supports the addition of new bindings, implementation types, policies, interfaces and data bindings by using a plugin-based architecture. This means that the platform is easily customizable. Furthermore, TUSCANY can run on the top of OSGi environments. However, the platform does not provide mechanisms for the dynamic adaptation of SCA applications.

2.3.3 The FraSCAti platform

The FraSCAti platform [Seinturier *et al.*, 2011, Seinturier *et al.*, 2009, Mélisson *et al.*, 2010a] allows the development and execution of SCA-based applications. The platform itself is built as an SCA application—*i.e.*, its different subsystems are implemented as SCA components. FraSCAti provides an homogeneous view of a middleware software stack where the platform, the non-functional services, and the applications are uniformly designe7d and implemented with the same component-based and service-oriented paradigm.

To do it, the platform is composed by four layers (cf. Figure 2.8):

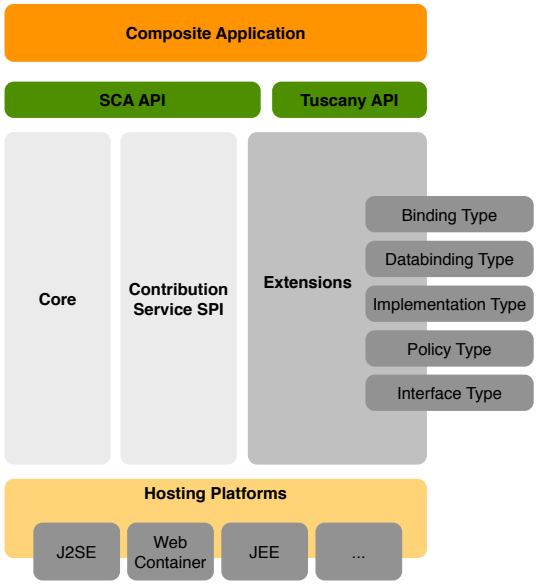


Figure 2.7: The TUSCANY SCA Java Runtime

1. The ***Kernel Level*** is based on FRACTAL [Bruneton *et al.*, 2006a], a lightweight and open component framework with basic dependency injection, introspection and reconfiguration capabilities. FRACTAL is conceived by applying concepts from *software architecture* [Shaw and Garlan, 1996], *distributed configurable systems*, and *reflective systems* [Smith, 1984]. In particular, this component model exploits the modularization and encapsulation fostered by the *software architecture*. The *reflective systems* provide the idea of defining meta-level activities as well as the reification of part of the component structure using control interfaces. From *distributed configurable systems*, FRACTAL inherits explicit component connections across multiple address spaces, and the ability to define meta-level activities for run-time reconfiguration.

   FRACTAL enables the customization of the execution policy associated with components. A concrete execution policy implemented in FRACTAL is called personality. In [Bruneton *et al.*, 2006a] are described two personalities of the component: JULIA, a general purpose personality for components with reconfiguration facilities and, DREAM, a personality for implementing message-oriented middleware solutions.

   The definition of *controllers* and *interceptors* enables the definition of personalities. The interceptors modify or extend the behavior of components when calls are done or received. For its part, each controller represents a facet of the personality (*e.g.*, the lifecycle or binding management) and exposes its services by means of an interface called a *control interface*.

   The FRACTAL components are equipped with a *control interface* that has the same role as the IUnknown interface in the COM component framework [Box, 1998]. This means that this interface allows the dynamic discovery of component capabilities and requirements. The low part of Figure 2.8 depicts this interface. As observed, the control interface includes methods for retrieving the component interfaces and type.

2. The ***Personality Level***, which customizes the component kernel by providing the execution semantics for components and implementing the SCA API and principles based on the FRACTAL component model. Table 2.1 presents the six controllers that make part of the FraSCAti personality. Each one of these controllers implements a particular facet of the execution policy of an SCA component. For this reason, the different controllers are im-

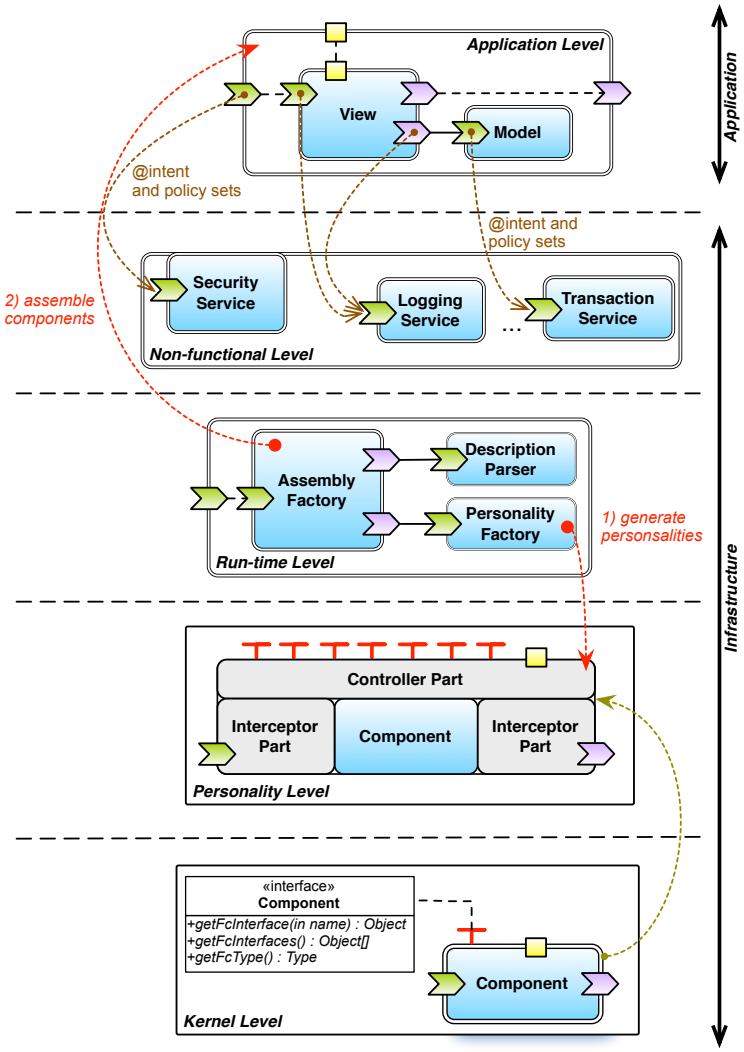


Figure 2.8: FraSCAti Platform Architecture

plemented as FRACTAL components. These controllers collaborate to provide the overall execution logic to the hosted component instance.

The Personality Level also extends SCA by allowing changes in application reconfiguration at runtime. In other words, FraSCAti enables the dynamic introspection and modification of an SCA application. This feature is important for designing and implementing agile SCA applications, such as context-aware applications and autonomic applications [Kephart and Chess, 2003]. In particular, it is possible to modify *wires*, *properties*, and *hierarchies*.

3. The ***Run-time Level*** instantiates SCA assemblies and components and defines a flexible configuration process, which is inspired by the *extender* and *whiteboard* [OSG, 2004] design patterns of OSGi. Figure 2.9 depicts the three composites that make part of the run-time level: **`Description Parser`**, **`Personality Factory`**, and **`Assembly Factory`**. The `Description Parser` loads and checks the SCA descriptors and creates the run-time model. The descriptors must be conform to a meta-model providing separation of the SCA

Interface	Methods	Definition
Wiring Controller	*bindFC(in cltItfName: String, in srvItf: Object): void*	Creates new wires
	listFc(): String[]	Retrieves the existing wires
	lookupFc(in cltItfName: String): Object	Queries the existing wires
	unbindFC(in cltItfName: String): void	Removes the specified wire
Instance Controller	*getFcInstance(): Object*	Creates component instances according to the four modes defined by the SCA specification: 1. STATELESS: All instances of a component are equivalent 2. CONVERSATION: An instance is created per conversation with a client 3. COMPOSITE: Singleton wrt. the enclosing composite 4. REQUEST: An instance is created per request
Property Controller	*getFcValue(in name: String): Object*	Retrieves the value of the specified property
	putFcValue(in name: String, in value: Object): void	Defines the specified property
Hierarchy Controller	*addFcSubComponent(in comp: Component): void*	Adds the specified subcomponent to the composite
	getFcSubComponents(): Component[]	Retrieves the list of subcomponents of the composite
	removeFcSubComponent(in comp: Component): void	Removes the specified subcomponent of the composite
Lifycycle Controller	*startFc(): void*	Allows application request to be processed
	stopFc(): void	Brings a component to quiescent state to enable safe reconfiguration operations
Intent Controller	*addFcIntentHandler(in intent: Object): void*	Adds the specified non-functional service to the component
	listFcIntentHandler(): Object[]	Retrieves the list of non-functional services of component
	removeFcIntentHandler(in intent: Object): void	Removes the specified non-functional service of the component

Table 2.1: SCA Personality Level API.

Metamodel and the FraSCAti Metamodel. The former contains all the elements specified in the SCA specification. The latter includes extensions, which are not part of the specification. The separation in two metamodels fosters the integration of new features that are not defined by the SCA specification such as new binding types.

The `Personality Factory` generates the personality of the SCA components. The nature of the code generated by the personality depends on the implementation type of the component. FraSCAti supports two different generation techniques: bytecode and source code generation.

The `Assembly Factory` creates the component assemblies that correspond to the run-time model created by the `Description Parser`. The different components in Figure 2.9 that make part of the `Assembly Factory` (*i.e.*, `Component`, `Property`, `Implementation`, `Service`, `Reference`, `Interface` and `Binding`) modularize the key concepts of the SCA model. Moreover, these components provide the modularity required for easily extending the supported bindings, interface language and component implementations.

4. The ***Non-Functional Level*** supports the SCA Policy Framework specification [Open SOA, 2007a] in order to provide non-functional services. This specification states the attachment of annotations to component assemblies for triggering the execution of non-functional services. For example, the `@Confidentiality`, `@Integrity`, and `@Authentication` annotations ensure confidentiality, integrity and authentication of service invocation, respectively.

   In order to integrate the non-functional services, FraSCAti implements them as SCA components. Then, FraSCAti offers an interception mechanism for connecting these non-functional services to application services.

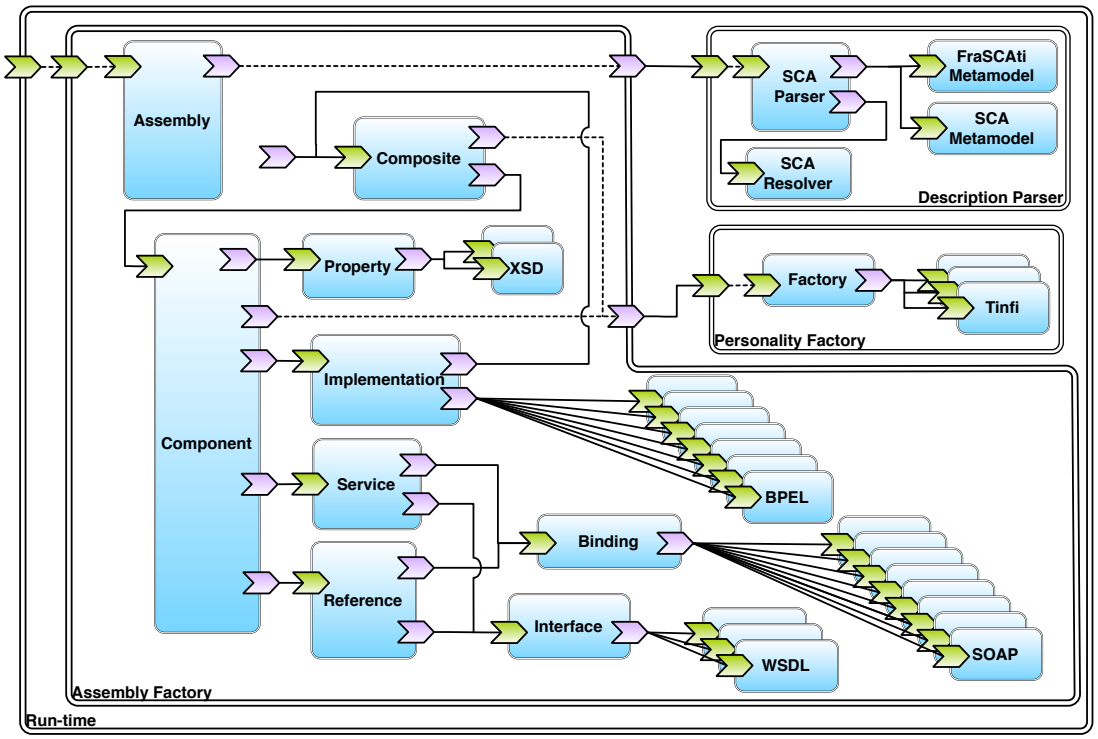


Figure 2.9: Run-time Level

Summary. FraSCAti represents a uniform SCA middleware platform where the applications, platform and non-functional services are development respecting the SCA standard. This property along with different levels provided by the platform make its customization and extensibility easier when required. Furthermore, FraSCAti brings reflexion capabilities into the SCA component model, functionality required for building SCA adaptive applications. Thus, the modularity, the usage of a uniform approach and the reflexion capabilities represent the principal advantages of the FraSCAti platform.

2.3.4 Selecting an SCA Platform

In order to work in our proposal, we require an SCA platform that satisfies the following properties:

- *Dynamic Reconfiguration Capabilities*: The purpose of context-based adaptation is to enable the dynamic reconfiguration of applications according to the current context. Therefore, the platform supporting the execution of these applications have to provide mechanisms for modifying their structure.
- *Extensibility*: The target platform has to enable the incorporation of the new functionalities necessary in the implantation of our proposal.
- *Support for Mobile Devices*: In this dissertation we target the adaptation of applications running on smartphones. This means that the runtime platform has to be customizable in order to execute such applications on devices with restricted capabilities.

Table 2.2 summarizes and compares the different SCA platforms in terms of these properties as well as considering the *supported functionalities*. As it can be observed, TUSCANY does not

Platform	Developer	Core Size	Dynamic Reconfiguration Capabilities			Extensibility Capabilities	Support for Mobile Devices	Supported Functionality		
			Application Level	Platform Level	Description			IDL	Bindings	Components
FraSCAti 1.3	OW2 Consortium	6.9 MB	✓	✓	Wires, properties and hierchies, policies	Plugin based mechanism enabling the incorporation of new Bindings, Interface Languages and Component Implementations	✓	Java, WSDL, UPnP	SOAP, REST HTTP, JSON-RPC, Java RMI, UPnP, UPnP, SLP, OSGi, JNA	Java, Java Beans, Scala, Spring, OSGi, FRACTAL, BPEL, scripts based on the Java Scripting API
Fabric3 1.6	Metaform Systems	10.2 MB	✓	✓	Extensions are required for dealing with reconfiguration. Wires and policies	Modular architecture enabling the integration of Bindings and Component Implementations	✗	Java, WSDL	Web Services (WS-* with full .NET interoperability), JMS, HTTP (sync/async, JSON, Hessian, JAXB), JAX-RS, TCP (sync/async), FTP	Java, JAX-RS/REST, Timers, Junit, Mock, Web Components
Tuscany 2.0	The Apache Software Foundation	2.1 MB	✗	✗	N/A	Plug points for Bindings, Data Bindings, Policies, Interface Languages and Component Implementation	✗	Java, WSDL	Web Services, Java RMI, HTTP, JSON-RPC, ATOM	Java, Spring, OSGi

Table 2.2: Comparison Between the Different SCA Open Source Platforms

offer reconfiguration functionality. Therefore we can not consider it as a suitable platform for the implantation of our proposal.

On the other hand, FraSCAti and Fabric3 offer runtime reconfiguration capabilities based on introspection. However, we select the FraSCAti platform because it provides an integral solution where the different elements that compose the platform are also implemented following the SCA paradigm. This feature makes the extensibility of the platform easier as well as the development of applications with a clear and natural separation of non-functional concerns. Furthermore, FraSCAti provides a lightweight version for mobile devices.

2.4 Service Discovery Protocols

Spontaneous interoperability or communication is an important issue in ubiquitous environments to deal with dynamicity and unpredictability [Zhu *et al.*, 2005, Sivavakeesar *et al.*, 2006]. This allows the interaction with resources that are dynamically discovered in the environment without requiring the deployment of new functionality. The Service Discovery Protocols (SDPs) have been conceived to enable this spontaneity. These protocols are designed considering several aspects. Some of them are presented briefly below:

1. *Service Description*: In order to describe the name and attributes of services, some SDPs (*e.g.*, Apple's Rendezvous [Inc., 2009]) employ a template-based approach. Other protocols, such as Bluetooth SDP and Jini, establish a *predefined* list of attributes and service names that are frequently used.

2. *Service Advertisement Searching*: The advertisement and searching of services is supported via an *announcement-based* approach or a *query-based* approach. In the former, the interested entities (*e.g.*, clients or service registries) listen on a channel in which the services announce their presence. In the latter, a service client sends a query associated with the searched service and it receives an immediate response. In this approach, clients do not process unrelated announcements.

3. *Service Registry Infrastructure*: The registration of services can be done via a *directory-based* model or a *nondirectory-based model*. In the directory-based model there is a central entity called directory that maintains service information and processes queries and announcements. In the nondirectory model, each service replies to the queries that match it.

4. *Service Selection*: When a service client searches for a service, it can find multiples services that match its requirements. In this case, the SDP can offer a *manual* or *automatic* selection. In the manual selection, the client has the responsibility of choosing one service between the list of matching services. In the automatic selection, the protocol provides a mechanism for select the service.

5. *Service Invocation*: Once the service is selected, the client needs to consume it. The discovery protocol can define the underlying interaction mechanism. If the protocol does not specify how to consume the service, the client and provider have the responsibility of deciding how to do it.

In the state of the art we find several discovery protocols developed by the research community, the industry and software vendors. Research conceived protocols include *NinjaSDS* [Czerwinski *et al.*, 1999, Gribble *et al.*, 2001] and *DEAPspace* [Hermann *et al.*, 2000]. In the industry we find *Salutation* [Consortium, 1999], *SLP* [Guttman *et al.*, 1999] and *Bluetooth SDP* [SIG, 2001]. *Jini* [Microsystems, 2003], *UPnP* [UPnP Forum, 2008] and *Rendezvous* [Inc., 2009] make part of the group of protocols defined by software vendors. In the following sections we give an overview of UPnP and SLP.

2.4.1 Universal Plug and Play (UPnP)

UPnP provides an architecture enabling the connectivity of intelligent appliances and wireless devices [UPnP Forum, 2008]. The protocol is intended to be used in ad-hoc or unmanaged networks in home, public places or small business.

With the UPnP architecture, the `Devices` (or service providers) and `Control Points` (or service consumers) can dynamically join the network, advertise their capabilities and learn about the presence of others devices. To do that, UPnP leverages on standard protocols and technologies. In particular the architecture exploits Internet protocols such as IP, TCP, UDP, HTTP and SOAP (for service invocation) and XML for describing services functionalities and capabilities. Figure 2.10 depicts the protocol stack employed by UPnP. Below we discuss briefly the relevant layers.

1. *UPnP-vendor*: In this layer the messages contain vendor specific information about its devices.
2. *UPnP-Forum*: This layer complements the vendor information with content provided by the UPnP Forum working committees[2].
3. *Simple Service Discovery Protocol (SSDP)*: Defined by Microsoft Corporation and Hewlett-Packard Company for the discovery of simple services such as printers or external disk drives, this protocol is the basis of UPnP. SSDP [Goland *et al.*, 1999] enables the discovery and advertisement of services without requiring previous configuration. The protocol leverages on UDP and HTTP protocols for the messages exchange and applies unicast and multicast routing schemes. The multicast is used for advertising the arrival or withdraw of services as well as for sending the service searched by clients. The unicast is employed by service providers for sending individual responses to clients.
4. *General Event Notification Architecture (GENA)*: UPnP defines the GENA protocol to notify the devices state in a asynchronous or polled way. The notifications are done using HTTP over TCP/IP.

Concrete uses of UPnP include the UPnP AV MediaServers and MediaRenderers. The MediaServers are computer systems storing and sharing multimedia content such as movies, music and photographs. DVD players, VCRs, PCs, Personal video recorders, CD players, MP3 players, Satellite set-top boxes and NAS (Network-Attached Storage) devices are examples of UPnP AV MediaServers. On the other hand, UPnP AV MediaRenderers render content and exposes an interface to control the playback. Televisions, PCs, digital media adapters, stereo systems and MP3 players represent MediaRenderers.

2.4.2 Service Location Protocol (SLP)

SLP defines a platform independent framework [Guttman *et al.*, 1999, Microsystems, 2000] for the discovery and provisioning of SLP-enabled services. Besides these functionalities, the protocol permits the organization of services and users into logical or functional groups and the recovery from basic server failures. All these functionalities are provided by default but SLP can be tuned according to the application needs.

In SLP, the services providers are called *Service Agents* (SAs) and the client are *User Agents* (UAs). Additionally, the protocol defines optional *Directory Agents* (DAs), which keep the registry of the available services in the environments. Furthermore, when present, a DA processes UAs requests. If there is no DA in the environment, the SAs agents advertise their capabilities (via multicast routing) and answer the requests from UAs.

To advertise services, the protocol defines *Service URL*s. These URLs indicate the service host and type following the format **`service:`**`<srvtype>`**`://`**`<hostname>`. The advertisement

[2] `http://upnp.org/membership/committees/`

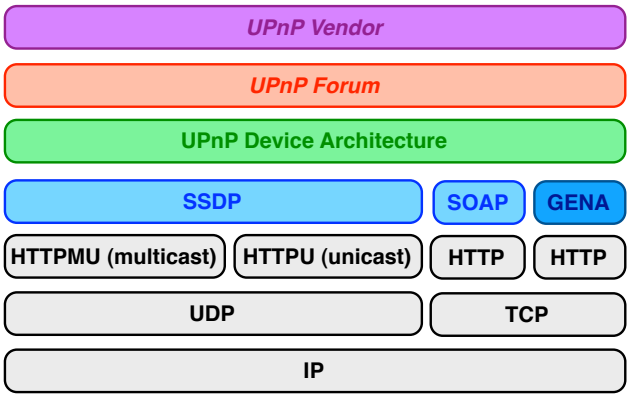


Figure 2.10: Protocol Stack in UPnP Architecture

also includes a collection of attribute/value describing the services and a lifetime. When the lifetime expires, the advertisement is no longer valid unless the service is reregistered.

Examples of SLP usages including the location of printers, file sharing in MAC OS, the finding of a variety of services on SUSE Linux[3], and the location of home actuators such as intelligent lights in the ACN protocol [Entertainment Services and Technology Association (ESTA),] for entertainment control. Additionally, the Distributed Management Task Force[4] (DMTF) defines SLP as the standard protocol employed by Web-Based Enterprise Management[5] (WBEM).

Choosing the Service Discovery Protocols

The discovery protocols employed in our solution have to be extensible and flexible enough for allowing us to exchange the context information with the required metadata. Moreover, they have to be widely used to guarantee a high integration level with legacy services and at the same time. Searching to satisfy these properties, in this dissertation we employ UPnP and SLP protocols as part of our solution for dealing with mobility issues in ubiquitous environments. The selection of UPnP is motivated because it is a well-accepted standard in the home entertainment industry to create, for example, TVs and NAS devices. Furthermore, the protocol is also language independent, allowing the implementation and interoperability of service providers and service consumers. Other attractive feature of UPnP is the support of its usage by the Digital Living Network Alliance (DLNA)[6], which ensures interoperability between devices from different manufacturers. On the other hand, we choose SLP because of its simplicity and flexibility. Furthermore, this protocol is also a standard employed by different operative systems (*e.g.*, Mac OS and SUSE Linux), the ACN protocol and the Storage Networking Industry Association[7]) (SNIA).

2.5 Summary

In this chapter we have given an overview of some existing approaches for the creation of SOA solutions and service discovery. We have also specified which of these approaches we employ in our solution. In particular, we have discussed REST architectural style and SCA advantages, which we will combine in order to provide our solution for dealing with context-based adaptation in ubiquitous environments (cf. Chapters 5 and 6). Additionally, we discuss some platforms

[3]SUSE Linux: http://www.opensuse.org/en/
[4]Distributed Management Task Force (DMTF): http://www.dmtf.org/
[5]Web-Based Enterprise Management (WBEM):http://www.dmtf.org/standards/wbem
[6]Digital Living Network Alliance: http://www.dlna.org/home
[7]Storage Networking Industry Association: http://www.snia.org/home/

that provide runtime support for SCA-based applications. Between them, we select the FraSCAti platform because of the reflection capabilities that it brings into the SCA component and its flexibility for being extended. Finally, we introduce UPnP and SLP discovery protocols, which we use in our proposal.

In the next chapter, we present some concepts and approaches from the ubiquitous computing that are essentials for a better understanding of the contribution of this dissertation. In particular, some of the addressed solutions employ the component and service discovery concepts introduced in this chapter in order to provide support for adaptation of applications.

Chapter 3

Ubiquitous Approaches

Contents

Motivation

Ubiquitous computing proposes a world where computers are everywhere and help people in their daily activities but at the same time are practically invisible. As stated by Mark Weiser,

father of the term, the ubiquitous computing ideal is "to make a computer so imbedded, so fitting, so natural, that we use it without even thinking about it" [Weiser, 1999]. To achieved it, the ubiquitous computing is grounded in the Moore's Law, which states that the number of transistors per chip, and consequently the power of microprocessors, doubles about every 18 months [Moore, 2000]. This means that increasingly tinier processors may become part of the environment (so called ubiquitous environments) and be incorporated in different kinds of objects connected together by means of wireless networks [Mattern, 2004, Mattern, 2005].

Although the ubiquitous computing tries to make computers more helpful and easier to use, the existence of spread and interconnected devices in the environment is not enough. The devices, and more specifically the applications running on them, should sense the environment and change their behavior according to it. For this reason, within the ubiquitous computing, we find the context-aware computing, which deals with these concerns [Weiss and Craiger, 2002, Schilit *et al.*, 1994]. This paradigm aims to build context-aware applications, which are applications benefiting from the pervasive computational resources available in the environment. However, in order to do it, several issues associated with heterogeneity, distribution and dynamicity have to be tackled. These problems, combined with the increasing popularity of ubiquitous computing in the last years, have arisen the interest in context-aware applications. Thus, in the literature we find several proposals [Román *et al.*, 2002b, Sousa and Garlan, 2002, Gu *et al.*, 2004, Gu *et al.*, 2005] dealing with context-awareness and the associated problems.

Chapter Organization

The goal of this chapter is to provide an overview on context-awareness. Therefore, the rest of this chapter is organized as follows. In Section 3.1 we introduce concepts associated with context-awareness that are relevant for this dissertation. Then, we describe some existing solutions for dealing with the building of context-aware applications (cf. Sections 3.2, 3.2 and 3.4) as well as their limitations (cf. Section 3.5). We finish with a summary of the elements presented in the chapter (cf. Section 3.6).

3.1 Definitions and Concepts

In this section we discuss some concepts associated with context-awareness. In particular, we introduce the terminology that we employ in this dissertation.

Context Information. In the literature, we can find several definitions of context information [Hirschfeld *et al.*, 2008, Coutaz *et al.*, 2005]. In this dissertation, when the term context is employed we mean *"any information that can be used to characterise 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"* [Dey, 2001]. Context information includes (but is not limited to) current location, relative location (*e.g.*, proximity to stores in a mall), physical environment (*e.g.*, temperature, time, light level, sound level, available bandwidth), device characteristics (*e.g.*, battery level from a laptop or smartphone), user preferences and user's activity [Mascolo *et al.*, 2002].

Quality of Context (QoC). QoC attributes [Razzaque *et al.*, 2006], indicators [Sheikh *et al.*, 2008] or parameters [Manzoor *et al.*, 2008, Bu *et al.*, 2006] specify the quality of the information that is used as context information [Buchholz *et al.*, 2003]. Precision, probability of correction, reputation and trust-worthiness are examples of QoC attributes. Table 3.1 shows the definition of some of these attributes.

QoC Attribute	Explanation
Update frequency/Refresh Rate	How often the information is updated
Lifetime	When the information becomes obsolete
Precision/Resolution	The granularity of the provided information
Probability of Correction	Margin of error, accuracy
Reputation and Trust-worthiness	The confience level in the provider (subjective and based on expirience)

Table 3.1: Some QoC Attributes

Information Sources. Any entity providing raw data that can be used potentially as context information. Sensors, wireless sensor networks (WSNs) and mobile devices (*e.g.*, providing information about the battery, memory and user preferences) are examples of information sources.

Context Sources or Context Providers. A context source or context provider is an entity that processes data for producing high level context information and makes it available to different context consumers. Web Services, publish-subscribe systems, instant-messaging systems, repositories, and smartphones are examples of context sources [Cohen *et al.*, 2004]. Depending on the context consumer, the context providers can also be considered as information sources.

Context Consumers. Entities employing the context information provided by context sources. Examples include (but not are limited to) context-aware applications, context-aware middleware and context-sources.

Context-aware Application. In this dissertation, we use the term *context-aware application* to refer to applications that benefit from context information for deducing the most suitable way to serve the final user. This deduction is made with a minimal or no intervention of the user [Cohen *et al.*, 2004].

Context-aware Middleware. Middleware platforms can exploit the context information in order to improve their behavior. In a similar way, these platforms can provide services for retrieving the context information and adapt applications according to it. These middleware platforms providing and/or exploiting the context information are called context-aware middleware in this dissertation.

Context Mediation. As already stated in Chapter 1, mediation is the process of collecting and processing usage data from different data resources (databases, Web Services, networked devices, etc.), to compute pertinent indicators and to deliver them to application programs (Web Services, enterprise applications, monitoring tools, etc.) [Coutaz *et al.*, 2005]. This context mediation is a keystone in the adaptation of context-aware applications and therefore it has to be treated carefully.

3.2 Middleware Solutions for Context-Awareness

In this section, we present some existing middleware platforms that deal with the issues associated with context-awareness. We focus the discussion on the main functionalities of the platforms in terms of context management and adaptation. In particular, we characterize these platforms by considering the following dimensions:

1. ***Technology and Paradigms Applied***: In this dimension we include the most relevant elements employed in the conception of the approach.

2. ***Communication Mechanisms***: Here we consider the *interaction* and *discovery* mechanisms. When supported, we specify the protocols and/or models for each dimension.

3. ***Context Information***: This dimension indicates if the approach is flexible in terms of context representations (*i.e.*, the support for different formats of information) and if it provides support for QoC.

4. ***Loose Coupling***: We characterize the loose coupling promoted by the approaches in terms of *mobility support* and *dependency definition*. Mobility support refers to the platform capacity for allowing the dynamic arrival and departure of entities (*e.g.*, context providers, clients, servers, sensors, etc.) that can be part of the platform ecosystem. Therefore, discovery capabilities are required for mobility support. For its part, the dependency definition expresses how the entities are related to each other.

5. ***Independent Communication Mechanism***: In this dimension we indicate if the approach provides flexibility for selecting and/or adding new *interaction* and *discovery* mechanisms.

6. ***Adaptation***: With this dimension we indicate if the platforms support static or dynamic adaptation and if the adaptation is at the platform and/or application levels.

3.2.1 Gaia

`Gaia` [Román *et al.*, 2002b] is a distributed middleware that provides similar functionality to an operating system. This middleware platform allows the coordination of software entities and heterogeneous networked devices in physical spaces (so called active spaces). `Gaia` provides services for event management and distribution (via event channels), context information query (for the context-based adaptation of applications), detection of digital and physical entities, storage of the information associate with entities, and file management. `Gaia` also provides a framework to build or adapt existing context-aware applications.

Characterization in Terms of the Dimensions. `Gaia` is conceived by using component-based design. The interactions between entities can be synchronous or asynchronous and are done via events and RPC. The already mentioned event channel allows service discovery and represents the dependency point. However, `Gaia` does not provide flexibility for selecting different communication protocols for discovery and interaction. Dynamic and static adaptation capabilities are provided at the application and middleware levels. Regarding the context information dimension, even if `Gaia` provides context processing, it does not consider QoC and the support for multiple context representations.

3.2.2 Gaia Microserver

`Gaia Microserver` [Chan *et al.*, 2005] is an extension of the `Gaia` middleware written in J2ME [Topley, 2002], which provides access to the native capabilities of mobile devices. To do this, the extension exports the functionality of these devices as `Gaia` components. In other words, the J2ME middleware acts as a proxy from the software running on mobile devices to `Gaia`. The Microserver also enables the interoperation of C++ native code with Java code. `Gaia` uses the Microserver platform to deliver dynamic software components and multimedia contents to users through their mobile devices.

Characterization in Terms of the Dimensions. Gaia Microserver brings into the Gaia platform the possibility of exploiting mobile devices. The interaction between these devices and other entities uses bluetooth [SIG, 2001] and GPRS [TelecomSpace,]. The J2ME proxy enables the discovery of the mobile devices and the communication is done via events as in Gaia. The provided adaptation is dynamic but only at the application level.

3.2.3 Aura

Aura [Sousa and Garlan, 2002] is an architectural framework for ubiquitous computing applications. This middleware platform provides services for the management of tasks, applications and context. In Aura, the tasks are abstract representations of a collection of services. When a user moves from one environment to another, a *Task Manager* migrates the task representations and instantiates the service providers in the new location. The services are provided by existing applications. Aura gives support for registry and access of services via an *Environment Manager* based on Jini technology [Waldo, 2000]. On the other hand, to manage context, Aura defines *Context observers*, which collect and notify changes on the context information to the Task and Environment Managers. In Aura, this context information enables the derivation of user intents.

Characterization in Terms of the Dimensions. Defining a component-based architecture, Aura develops software connectors that benefit from Jini technology for interaction and service registry. However, the platform does not support mobility. In a similar way, the mechanism associated with the context retrieval and processing does not consider the QoC dimensions and multiple context representations. Finally, regarding the adaptation dimension, Aura only provides dynamic adaptation at the middleware level.

3.2.4 CORTEX

CORTEX [Sorensen *et al.*, 2004, Blair *et al.*, 2004] is a context-aware middleware for pervasive and ad hoc environments. The platform is based on the concept of *Sentient Objects* (SOs) and *Component Frameworks* (CFs). SOs are autonomous entities that can get data from the environment and share information between them. They consume and produce events. SOs are able to make decisions and perform actions based on the information sensed. On the other hand, the CFs offer services to the SOs, such as *publish-subscribe*, *group communication*, *context retrieval*, *service discovery* (via protocols such as SLP and UPnP) and *QoS management*. CORTEX can be reconfigured at runtime using a reflective API.

Characterization in Terms of the Dimensions. CORTEX is built on the top the OpenCom component model. The usage of such component model makes the support of dynamic adaptation at the application and middleware levels easier. On the other hand, by encapsulating the context distribution concerns on the SOs, CORTEX has the flexibility for supporting different discovery protocols. However, the SOs do not consider QoC as a relevant aspect of the context information. Furthermore, the interactions are based only on the SOAP messaging framework.

3.2.5 CARISMA

CARISMA [Capra *et al.*, 2003] (Context-aware Reflective mIddleware System for Mobile Applications) applies the reflection paradigm to enhance the development of adaptive and context-aware mobile applications. The idea behind CARISMA is to customize the platform considering the applications needs. In order to do that, the middleware behavior with respect to an application is reified as meta-data in a profile. This profile contains the description of associations between the service that the middleware customizes, the policies that can be used in the service invocation and the context configuration that allows the use of the policies. In each service invocation, the client application passes its profile to the platform and determines the policies that can be applied

according to the current context. CARISMA provides a reflective API for modifying at runtime the associations described by the profile. The conflicts that may arise between profiles are resolved using a micro-economic approach. In this approach, the system is modeled as an economy where the consumers (applications) reach an agreement about a limited set of goods (the policies) using the middleware platform like auctioneer.

Characterization in Terms of the Dimensions. CARISMA provides adaptation at the middleware level by exploiting reflection mechanisms. The dependency between the middleware platform and the applications customizing it is stated via the profiles. The context information is just used for determining the profiles that can be applied. CARISMA does not include QoC dimensions in this analysis. In terms of communications, the platform does not consider the discovery capabilities as a relevant issue in context-based adaptation.

3.2.6 MobiPADS

MobiPADS [Chan and Chuang, 2003] (Mobile Platform for Actively Deployable Service) is a system for mobile environments. In this middleware platform, the services, called mobilets, can migrate between MobiPADS environments. Each mobilet consists of a slave and a master. The slave resides in the server side, and the master in the client side. The mobilets are configured as chained objects to provide augmented services and protocols to the mobile applications. MobiPADS achieves context-awareness with the utilization of an event model. The platform monitors the status of the interested context and notifies the changes to the subscribed entities. By means of the event model, the primitive events are composed in an event graph. In this way, when an event service is built and subscribed, it has to monitor and analyze the basic events and to match them according to the event graph structure.

Characterization in Terms of the Dimensions. MobiPADS employs a service channel for making the mobilets available. However, despite the existence of this service channel, spontaneous communications are not possible. Considering adaptation, it is supported in the platform level, by means of system profiles, and in the mobilets, by allowing them to change according to the events that they receive (*i.e.*, it is possible to adapt applications and the middleware platform dynamically). Finally, the event graph allowing the context management does not include the analysis of QoC dimensions.

3.2.7 MiddleWhere

MiddleWhere [Ranganathan *et al.*, 2004] is a distributed middleware architecture for location built on the top of Gaia (cf. Section 3.2.1). This means that the main goal of the platform is to provide advantages in the development of location-aware applications. To do that, MiddleWhere defines a hierarchical location model that deals with three kinds of location: points, lines, and polygons. The location, which is stored in a spatial database, can be expressed in coordinates and a symbolic way. This location information is characterized in terms of Quality of Context attributes such as *freshness*, *confidence* and *resolution*. The applications can use the location data in two ways: *i)* Querying the location of the interested objects and, *ii)* Subscribing to be notified when a location condition becomes true.

Characterization in Terms of the Dimensions. Based on CORBA [Group, 2006a] and Gaia, the most relevant property of this platform is the usage of quality attributes associated with context information. Nevertheless, Middleware does have a direct support for adaptation at the platform and application levels. On the other hand, the interaction between MiddleWhere and applications is done via adapters. In the interaction, the discovery capabilities are not offered and therefore there is not mobility support.

3.2.8 SOCAM

Service-oriented Context-Aware Middleware (SOCAM) [Gu *et al.*, 2004, Gu *et al.*, 2005] is a platform to build context-aware mobile services. SOCAM uses ontologies to model the context. The platform can support semantic representation, context reasoning and context-knowledge sharing. SOCAM architecture consists of:

(a) *Context Providers* giving context information, which is represented as context events in the form of OWL descriptions;

(b) *Context Interpreters* that provide high-level context information and therefore they are also considered context providers. The Context Interpreters include *Context Reasoners*, containing rules that trigger actions associated with context changes, and *Context Databases*, containing instances of the current ontology;

(c) A *Location Service* for registry and discovery of context providers and others services;

(d) *Context-aware Mobile Services* that adapt their behavior according to context information. They obtain this information by querying or listening specific events from context providers.

Characterization in Terms of the Dimensions. As already stated, SOCAM employs ontologies in order to model context information. Nevertheless, this modeling does not consider as relevant the QoC information. The location service enabling discovery is based on a SLM service discovery and the interaction is event-based. There is not support for several communications mechanisms. Regarding the adaptation, it is only provided at the application level.

3.2.9 CAPNET

CAPNET [Davidyuk *et al.*, 2004] is a context-aware middleware for mobile multimedia applications. CAPNET offers a component based architecture, which core components are:

1. *Component Manager*, which controls the components and their stubs;
2. *Connectivity Manager* that manages and monitors the connection of the mobile devices;
3. *Messaging*, which creates channels and supports asynchronous communication, remote procedure calls and channel-related operations;
4. *Service Discovery* for locating services and available components.

CAPNET also has others components to deal with context-awareness: *Context-Based Storage*, *Context*, *User Interface and Media*. The Context-Based Storage component stores and retrieves context data by request. The Context component provides context information acting as a wrapper for context sensors. The User Interface component supports the design and implementation of UI applications. In order to do that, the UI allows three different techniques: abstract UIs (XML), plug-in UIs (downloadable Java code) and Web-based UIs (HTML). The media components provide functionality to capture images, audio and video, facilitating the portability and scalability of native media capabilities across the various devices.

Characterization in Terms of the Dimensions. CAPNET enables its customization at deployement time. The communications between the middleware platform and applications is based on events and several interaction protocols are supported. The Service Discovery component that allows the discovery of services uses the Jini Technology. In CAPNET, the context management does not include the support for multiple representations and QoC attributes.

3.2.10 Reconfigurable Context-Sensitive Middleware (RCSM)

Reconfigurable Context-Sensitive Middleware[Yau *et al.*, 2004, Yau *et al.*, 2002] (RCSM) provides an object-based framework for developing and supporting context-sensitive applications. In RCSM, the context-aware applications are modeled as context-sensitive objects with two parts: *i)* A context-sensitive interface that encapsulates the description of the applications' context-awareness and *ii)* A context-independent implementation, which remains context free. To allow ad-hoc communications, RCSM employs an Object Request Broker (R-ORB) that has functionalities for context discovery, collection and propagation. A R-ORB is able to establish context-triggered communication channels (CTCs) between remote R-ORBs based on application-specific context. To maintain CTCs with remote devices, the R-ORB uses R-GIOPs (RCSM General Inter-ORB Protocols).

Characterization in Terms of the Dimensions. RCSM exploits the object paradigm for enabling the dynamic and static adaptation of context-aware applications. Even if the platform does not provide discovery capabilities, its R-ORB brings into play flexibility in terms of interaction and discovery protocols. However, the platform does not consider the quality of the context information as a relevant issue.

3.2.11 CARMEN

CARMEN [Bellavista *et al.*, 2003] (Context Aware Resource Management ENvironment) is a middleware to manage resources in wireless settings assuming temporary disconnections. The middleware supports the design, development and deployment of context dependent services for the wireless Internet. CARMEN uses metadata for representing the context characteristics and the choices in the service behavior. Two types of metadata are used: profiles and policies. The profiles describe users, devices, service components and sites. The policies specify bindings, migration and access control. Each user is associated with a single proxy or Mobile Agent (MA) through which the resources can be accessed. When the user moves from an environment to another one, the proxy is migrated (using wired connections). This MA is responsible for making the resources available in the new location. The resource migration can be done using one of the next strategies (specified in the profile of the device): moving the resources with the agent, copying the resources, using remote references or rebinding to new services with similar functionality. The middleware solution also provides an articulated naming system with identification, discovery, and directory facilities.

Characterization in Terms of the Dimensions. CARMEN uses reflection techniques for enabling its adaptation at runtime. The interaction based on Mobile Agents does not provide flexibility for selecting different protocols. In a similar way, CARMEN discovery facility does not consider the usage of different mechanisms. The metadata approach for context description excludes the context quality treatments.

3.2.12 Cooltown

Cooltown [Debaty *et al.*, 2005] is a distributed software framework to integrate the physical world with the Web. Each physical entity (*i.e.*, people, things or places) is represented as a *Web Presence* with a URL associated. The URLs are passed between devices for interactions. The Web Presences have the following core modules for modeling physical entities on the Web: *i)* the *description module* containing information about the characteristics and capabilities of the identity, *ii)* the *directory module* that manages the relationship of the Web Presence, *iii)* the *discovery module* allowing the automatic update of the relationships, *iv)* the *autobiographer module*, which is a log of the Web Presence, *v)* the *observer module* for observing the discovery module and triggering actions when specific criteria are met and *vi)* the control module that enables interactions with

the physical world. The relationships between the Web Presences include *Contains*, *isContainedIn*, *isNextTo* and *isCarriedBy*, and new types can be added. The relationships are directional, they have properties and can be subtypes of other relationships. `Cooltown` offers also tools to build the Web Presences.

Characterization in Terms of the Dimensions. `Cooltown` considers each entity as a web resources, which have URLs enabling their access by means of HTTP. The discovery module of the platform is extensible, meaning that different discovery mechanisms can be used. Furthermore, `Cooltown` is able to support the joining and leaving of the Web Presences. The directory module is the mechanism for stating the dependencies of the Web Presences. Finally, the platform can be easily customized at deployment time.

3.2.13 A Large Scale Peer-to-Peer Context Dissemination Middleware

In [Yasar *et al.*, 2010] authors propose a middleware solution enabling the optimization of context information dissemination by defining virtual groups that share some common contextual information (*i.e.*, location, direction, interest). The goal of the middleware platform is to reduce irrelevant communications between nodes in a large scale peer-to-peer network. In the approach, the context information is modeled by using an ontology which contains specification of context artifacts and their relations. The context-aware groups are composed by a criteria such as location, a set of contextual information interests (*e.g.*, traffic or parking information), and a set of sub-interests (*e.g.*, traffic jams or accidents). The group exchanging only relevant information are formed by using a backpropagation algorithm [Preuveneers and Berbers, 2007] for inter and intra-group communication. Finally, the middleware platform is evaluated with a large scale vehicular network [Yasar *et al.*, 2008].

Characterization in Terms of the Dimensions. The approach focuses in the dissemination of context information and their optimization by reducing the exchange message between context-based groups, an important aspect in context mediation. Furthermore, with the applied back-propagation algorithm consider the quality of the context information. The discovery mechanism exploits broadcasting for discovering context-aware groups and notifying the groups join and leaving. Nevertheless, there is not flexibility in terms of communications.

3.2.14 A Peer-to-Peer based infrastructure for Context Distribution in Mobile and Ubiquitous Environments (MUSIC Peer-to-Peer)

In [Hu *et al.*, 2007], the authors propose a peer-to-peer infrastructure dealing with context mediation. Context-aware peers are grouped into three categories according to their resources and functionalities: *sensors peers*, *disseminators peers*, and *consumer peers*. This classification configures the deployment of the infrastructure components on the context-aware peers. The approach uses two key services to retrieve the context information: the Context Service and the Distribution Service. The Context Service stores context information in a local repository and processes the queries. The Distribution Service is used by the context service to retrieve context information from peers (when the information is no in the local repository). The approach therefore exploits the collectiveness property of the peer-to-peer paradigm to provide context fault tolerance.

Characterization in Terms of the Dimensions. The most relevant property of this approach is the usage of the peer-to-peer paradigm for interaction and discovery. The context integration is the focus of the approach. Nevertheless, in the peer-to-peer based infrastructure adaptation and context management issues are not considered.

3.2.15 Summary of Middleware Solutions

The set of middleware solutions that we just presented can be grouped in four categories:

1. *General Solutions for Context-Awareness*: This category refers to the middleware platforms searching to face the different issues associated with context-based adaptation, *i.e.*, heterogeneity, mobility and the adaptation itself. In this category we include Gaia and RCSM. The advantage with this kind of solutions is the functionality richness for the development of context-aware applications. The problem with them is that they tend to be inflexible in terms of relevant aspects such context management (*i.e.*, context integration and processing) and communications. Furthermore, although some of their functionality can be improved, the extensibility becomes a difficult task because of the same complexity associated with platform.

2. *Solutions for Context Processing and Integration*: In this category we find CORTEX, CARISMA, Music Peer-to-Peer and the Large Scale Peer-to-Peer Context Dissemination Middleware. This kind of solutions focuses in the exchange and processing of context by means of efficient mechanisms. The strong point of this category is the simplicity and easy usage of the solutions. The drawback is associated with the not consideration of adaptation at the application level.

3. *Solutions for User Mobility*: In this category we include the middleware platforms supporting service migration or replacement between environments and considering the usage of mobile devices. Therefore, in this category we classify Aura, MobiPADs, CARMEN, Gaia Microserver and SOCAM. The interesting aspect of this approaches is the possibility given to user of benefiting from the same services even if the user is nomad. However, service migrations/replacement requires highly controlled environments, where it is required to provide homogenous services and/or entities supporting the migration of the functionality.

4. *Specific Purpose Solutions*: In this category we include the solutions conceived for developing an specific kind of context-aware adaptation. For example, CAPNET enables the development of multimedia applications, MiddleWhere of location-aware applications, and Cooltown the conception of Web presences integrating the physical world with the web. These specific solutions allow the clear identification of the different concerns associated with context mediation in ubiquitous environments such as the need for QoC information.

The previous classification enables us to identify the main strengths and weaknesses of the different approaches and to confirm the properties that we will offer in our proposal in terms of context mediation: *independent representation of context information*, *independent communication mechanisms*, *loose-coupling between context producers and consumers*, and *flexibility and extensibility*. We provide more detail about these properties in Chapter 5.

3.3 Service Discovery Solutions for Ubiquitous Environments

In Section 2.4, we have introduced the basic concepts about service discovery. In this section we discuss some middleware approaches dealing with the problem of mobility in ubiquitous approaches. As it will be presented, these approaches focus in the interoperability between SDPs and not on the flexibility in terms of extensibility and adaptation.

3.3.1 INDISS: Interoperable Discovery System for Networked Services

INDISS [Bromberg and Issarny, 2005] is a system based on event-based parsing techniques to provide SDP detection and full service discovery interoperability. The detection of protocols is

done by using a `Monitor Component`. This component joins different multicast groups (associated with different discovery protocols) and waits for multicast messages or reply messages depending if the SDP is passive or active, respectively. The data arrival on an specific port and multicast address indicates the usage of the SDP associated with them.

Once the used protocols are identified, the next step is enabling the interoperability of SDPs. With this purpose, a `Parser` component extracts semantic concepts as events from the SDPs messages. Then, a `Composer` component transforms the events into messages of the underlying SDP used by the client application.

Characterization in Terms of the Dimensions. INDISS focuses in the interoperability between discovery protocols. Therefore, this platform provides flexibility in terms of discovery mechanism but it does not consider the possibility of using diverse interaction protocols. The component-based architecture of INDISS can be easily configure at deployment time but mechanism for dynamic adaptation are not considered.

3.3.2 ReMMoC: A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments

ReMMoC [Grace *et al.*, 2005] is a Web-Service based reflective middleware for discovery and access of services in mobile clients. This middleware platform has two main functionalities: *i)* the detection of the available services in the environment independently of the protocol used in their advertisement and *ii)* the interoperability with services implemented upon different interaction types. To provide these functionalities, ReMMoC exploits the *Component Framework* (CF) architecture promoted by the OpenCOM model [Coulson *et al.*, 2004]. In this way, the ReMMoC architecture is composed by the the `Service Discovery CF` and `Binding CF`. The former is associated with the functionality *i)* and the latter with the *ii)*. The two CFs allow the incorporation of new protocols by defining personalities. The Service Discovery CF is able to reconfigure itself to work with the current discovery protocols being used in the environment. Furthermore, this CF executes service lookup by employing different SDPs in parallel. For its part, the Binding CF can adapt itself to become a client of a service discovered at runtime.

Characterization in Terms of the Dimensions. ReMMoC provides flexibility in terms of interaction and discovery mechanisms. In terms of adaptation, dynamic reconfigurations are possible at the middleware and application levels thanks to the use of the OpenCOM model. However, the platform is not context-aware meaning that there is not support for context-based adaptation.

3.3.3 A Multi-protocol Framework for Ad-hoc Service Discovery

In [Flores-Cortés *et al.*, 2006], authors propose a component-based framework for the development of an adaptive multi-personality service discovery middleware, which operate in fixed and ad-hoc networks. The solution, based on the OpenCOM model, has an architecture composed by 6 components:

1. The `Advertiser component` has the role of a Service Agent, *i.e.*, this advertises the services descriptions to neighbor nodes. The supported discovery protocols are SSD [Sailhan and Issarny, 2005], GSD [Chakraborty *et al.*, 2002], ALLIA [Ratsimor *et al.*, 2002] and SLP [Guttman *et al.*, 1999].

2. The `Request component` is a User Agent enabling the transmission and processing of request messages. Additionally, this component matches the requested service descriptions with descriptions stored in the local cache.

3. The `Reply component` generates and sends service replies when the request component detects that a service description matches.

4. The `Cache component` stores information that protocols require for working. This information includes key messages, service descriptions and routing tables.

5. The `Policies component` provides three functions: *i)* The retrieval of policies from XML files, *ii)* the application of these policies to components connect to the Policies component and, *iii)* the change of policies values according to user preferences and application needs. The policies include caching preferences, advertisement preferences, directory preferences and forwarding preferences.

6. The `Network component` provides different routing schemes such as unicast, multicast and bordercast.

This architecture promotes component re-use and simplifies configuration and dynamic reconfiguration of multiple concurrent protocols.

Characterization in Terms of the Dimensions. This platform supports the usage of multiple discovery protocols. However, it does not consider the flexibility in terms of interaction mechanisms. The platform can be customized at deployment time but despite the use of OpenCOM, it does not provide dynamic reconfiguration capabilities.

3.3.4 Service Discovery Solution Summary

The discussed solutions for service discovery provide two different mechanism for dealing with the protocol heterogeneity: *i)* interoperability of protocols and *ii)* middleware adaptation according to the current protocol configuration and protocol functionality reuse. Both of them provide the flexibility in terms of discovery protocols for context based adaptation. However, the monitoring of available protocols and reconfigurations according to them can become an expensive task considering that the focus of the adaptation is the user application. Moreover, the development of parsers and composers for the interoperability is a difficult task considering that some protocols define also the underlying interaction mechanism and some information is lost in the translation process.

3.4 Data-Oriented Architectures in Context-Mediation

The data-oriented paradigm promotes loose coupling between interacting entities by exposing data as first class entities. This means that the communication between consumers and producers focuses on the data exchange via messages. These messages are sent using an interoperable protocol. The interfaces are defined by means of a data model, participant roles and metadata describing the data structure and including QoS information. The interfaces hide the interaction entities code. Furthermore, the data-oriented paradigm promotes the separation of data handling and data processing for making the modification of applications easier.

A popular data-oriented paradigm is the REST architectural style, which we have already discussed in Section 2.1.2. Because of its simplicity and the already mentioned advantages of data-oriented approaches, REST represents an attractive alternative for the exchange of context information in ubiquitous environments. However, in the state of art we do not find a lot of works dealing with this issue by applying such architectural style. In [Christensen, 2009], for example, the authors analyze the potential of combining Cloud Computing and RESTful Web Services for the creation of a new generation of mobile applications that exploits the context information. Other REST related works exploit the paradigm for integrating mobile devices in distributed systems. In particular, [Riva and Laitkorpi, 2009] describe a methodology for building mobile services based on REST. Authors in [Ulmer *et al.*, 2009] propose the combination of REST and WSDL for providing a distribute object model working with mobile devices. In [Antila and Mantyjarvi, 2009] describe a resource-centric architecture for sharing information between mobile devices. Other usages of REST in literature include the exposition of databases

by means of RESTful based connectors [Marinos *et al.*, 2010] and the modeling of widgets from application interfaces as REST resources in order to enable the adaptation of these interfaces according to the devices look and feel [Stirbu, 2010].

New Generation of Mobile Applications

As already stated, [Christensen, 2009] considers the potential of smartphones for building context-aware applications. In particular, the author considers the possibility of a customer *always connected* thanks to the support of connection modes such as 3G and 802.11x. This *always connected* capability open up for benefiting from architectural advantages of cloud computing. Furthermore, the GPS combined with the different connection modes foster the development of new kinds of applications such as Location Based Services (LBS) [LaMarca *et al.*, 2005] and spatial augmented reality (SAR) [Marsal, 2009]. Regarding the limited capabilities of the smartphones, the cloud computing provides off-device storage, processing and queuing capabilities, and mechanisms to secure the integration of the device within the cloud environment. Finally, the RESTful Web Services act as a bridge between the smartphones and the cloud environment. These services are suitable for this integration because of their simplicity and easily consumption. The RESTful Web Services are memory friendly and they can be processed by means of event-based parsers. Moreover, the flexibility for encoding the body with different formats (binary, XML, plain text, HTML) reduces the time, processor and memory resources required for processing the REST requests and responses.

3.5 Limitations of the existing approaches

In the previous sections we gave an overview of the existing solutions for dealing with context-awareness in ubiquitous environments. Now, in this section we provide an analysis of the main limitations of these approaches. To do that, we summarize in Table 3.2 the discussed solution. In this table, we consider the dimensions introduced in Section 3.2.

Observing Table 3.2, we deduce that one of the main limitations of the existing approaches is the **lack of flexibility** in terms of the *context information* and *independent communication mechanisms* dimensions. In general, the proposed platforms support several kinds of protocols and mechanisms for interaction and discovery. However, most of them only provide a determined communication method and do not consider the possibility of adding new ways of doing it. This restriction represents a problem in ubiquitous environments, where we find a plethora of communication mechanisms used by the present entities. We also see that there is a lack of flexibility regarding context representations. The satisfaction of this property is important because the resource variability of the different entities in the environment. Even platforms, such as ReMMoC and GAIA Microserver, conceived for working in mobile devices do no offer multiple context representations.

Other limitation associated with the existing proposals is **the absence of consideration for the quality of the context information** (cf. Section 3.1). If the exchanged information does not respect a minimum of requirements, the triggered adaptations based on such information will lead to unexpected application behaviors. In terms of *dependency definition* and *implementation technologies*, we observe that several proposals tend **to impose RPC approaches and specific development models**. The problem with this kind of restrictions is that they make the integration of new entities difficult. In particular, the integration of legacy systems can become an expensive task. Furthermore, the usage of a particular implementation technology restricts the type of devices that can be exploited for executing the context-aware applications.

Considering the *adaptation capabilities*, the static and dynamic adaptations are equally important in ubiquitous environments. The possibility of reconfigurations at deployment time of applications and the underlying platforms are required because of the diversity of devices available in the environment. In a similar way, the adaptation at runtime enables the usage of the context information for improving applications and platforms behavior. Therefore, it is desirable to have

dynamic and static adaptation at the application and middleware levels. However, in Table 3.2 we see that only GAIA and MobiPADS offer support for adaptation at both levels. The other approaches make available the information and then applications have the responsibility of using the context information for reconfiguring themselves. This means that application developers not only have to provide flexibility points in their applications but they also have to conceive an additional layer exploiting the context information. Additionally, if the programming model or the platform itself do not offer adaptation capabilities, the development of such context-aware applications becomes a real challenge. Thus, the **lack of support for adaptation** represents another limitation of existing approaches.

3.6 Summary

In this chapter we have discussed three kinds of ubiquitous approaches: context-aware middleware, solutions for service discovery and REST-based approaches. We have summarized the principal functionalities and characteristics of the different approaches and then we have identified the existing limitations motivating our work. In particular we find that **the lack of flexibility related to communication and context information**, **the limited support for adaptation**, **the imposition of RPC for dependency definition**, **the imposition of a single development technology** and **the minor importance given to the QoC** represent the principal problems associated with the existing approaches. From these limitations we derive the following challenges that we need to face in order to provide a middleware solution for ubiquitous environments:

1. *How to deal with communication heterogeneity*: The plethora of devices, services and applications available in ubiquitous environments promote the proliferation of different kinds of interaction and discovery paradigms. A solution integrating context information and enabling application adaptation have to provide the flexibility for selecting the most suitable mechanisms in different situations. This has to be done considering the hardware and software variability of client and service providers. Therefore, flexibility is key in a middleware solution dealing with this kind of heterogeneity.

2. *How to deal with context heterogeneity*: As already stated, in ubiquitous environments the heterogeneity is the rule and not the exception. This heterogeneity also covers the context information available in the environment. In particular properties that characterize the information such as the freshness and precision can be relevant for the application adaptation. Therefore, a mechanism dealing with this issue have to be provided.

3. *How to provide support for adaptation*: The integration of context information is the first step for enabling context-based adaptation. A solution dealing with such kind of adaptation has to enable not only the reconfiguration of application but also of the platform itself.

In the next chapters we discuss others approaches that also deal with context-awareness issues. However these approaches have the particularity of applying principles from the autonomic computing for providing the adaptation of applications.

	Dimension / Middleware Approach	Technology and Paradigms applied	Communication Mechanisms		Context Information		Loose Coupling		Independent Communication Mechanism		Adaptation		
			Interaction	Discovery	Support for Multiple Representations	QoC	Mobility Support	Dependency Definition	Interaction	Discovery	Static	Dynamic	Level
Service Discovery Solutions	INDISS	Components	Depending on the underlaying discovery protocol	Interoperability between protocols, SLP, UPnP	✗	✗	N/A	Events	✗	✓	✓	✗	Middleware
	ReMMoC	Components, Component Frameworks (CFs), OpenCOM	CORBA, SOAP, event publisher and subscriber	SLP, UPnP	✗	✗	N/A	WSDL	✓	✓	✓	✓	Middleware
	A multi-protocol framework for ad-hoc service discovery	Components, OpenCOM	Depending on the underlaying discovery protocol	SSD, GSD, ALLIA, SLP	✗	✗	N/A	Component Interfaces	✗	✓	✓	✗	Middleware
Middleware Solutions for Context-Awareness	GAIA	CORBA, Components	Asynchronous and synchronous commun. via Events and RPC	Events (channels, producers and consumers)	✗	✗	✓	Channels	✗	✗	✓	✓	Application, Middleware
	GAIA Microserver	J2ME, Components	Bluetooth, GPRS	J2ME Proxy, Events (channels, producers and consumers)	✗	✗	N/A	J2ME Proxy/ Channels	✗	✗	✗	✓	Application
	MUSIC Peer-to-Peer	JXTA, JXME	Peer Communication Service	Discovery Service	✗	✗	✓	Peer Group	✗	✗	✗	✗	N/A
	Cortex	Components, Reflection, OpenCOM	Events, SOAP	SLP, UPnP (sentient objects)	✗	✗	✗	Sentient Objects	✗	✓	✓	✓	Middleware
	CARMEN	Components, Reflection	Proxies (MAs)	Discovery Facility	✗	✗	✗	Proxies (MAs)	✗	✗	✗	✓	Middleware
	Aura	Components, Jini	Connectors	Jini	✗	✗	✗	Environment Manager	✗	✗	✗	✓	Middleware
	CARISMA	Reflection	Stubs, Connectors	✗	✗	✗	✗	Profiles	✗	N/A	✗	✓	Middleware
	Cooltown	Components	HTTP, URLs	Discovery Module	✗	✗	✓	Directory Module	✗	✓	✓	✗	Middleware
	MiddleWhere	Components, CORBA, GAIA middleware	Adapters	✗	✗	✓	✗	Adapters	✓	N/A	✗	✗	N/A
	MobiPADS	Reflection, Components	Channel Service (TCP connection)	✗	✗	✗	✗	Channel Service, Mobilets	✗	N/A	✓	✓	Application, Middleware
	SOCAM	Ontologies, SOA	Events	SLM service discovery (for context providers)	✗	✗	✗	Location Service	✗	✗	✗	✓	Application
	RCSM	Objects, Components	R-ORB	R-ORB	✗	✗	✗	Channels (CTCs)	✓	✓	✓	✓	Application
	Large Scale Peer-to-Peer Context Dissemination Middleware	Ontologies	Relevance backpropagation algorithm	Peer-to-Peer based	✗	✓	✗	Context-based groups	✗	✗	✗	✗	N/A
	CAPNET	Components, Jini	Events, Channels, RPC, TCP/UDP, HTTP Multicast	SD Component that locates services components	✗	✗	✗	Messaging Component	✓	✗	✓	✗	Middleware

Table 3.2: Different Ubiquitous Middleware

Chapter 4

Autonomic Computing Approaches

Contents

Motivation

Nowadays the Information Technology (IT) Industry has to face the rapidly growing complexity of operating, managing, and integrating computing systems [Ganek and Corbi, 2003]. In general, these systems are heterogenous infrastructures composed of different kinds of applications, components and tuning parameters. Consequently, the maintenance of such systems is a difficult process prone to errors, which requires skilled IT professionals. Therefore, the complexity of systems has a double impact on the companies: *i)* the affectation of the business operation in case of system failures and *ii)* the maintenance has a high operation cost. To deal with these issues, the IT Industry introduces a new paradigm for developing software systems, the *Autonomic Computing* [Ganek and Corbi, 2003].

The objective of the Autonomic Computing is the development of self-managing systems, which are able to configure, heal, protect and optimize by themselves. The idea is inspired by the properties of natural systems such as the autonomic nervous system [Kephart and Chess, 2003]. In particular, this system governs low-level functions, such as the body temperature and heart rate, freeing the brain from these responsibilities. In a similar way, autonomic systems search to reduce the work of system managers by automatizing, whenever possible, the maintenance tasks. The idea behind this is to build systems that require a minimal human intervention to work properly.

Chapter Organization

The goal of this chapter is to provide an overview of the main concepts of Autonomic Computing (cf. Section 4.1) employed throughout this dissertation, state the relation between this kind of computing and context-awareness (cf. Section 4.2) and discuss some existing Autonomic Solutions (cf. Sections 4.3 and 4.4). We also present a summary of the foundations of this dissertation (cf. Section 4.5) and the challenges derived from the discussed approaches (cf. Section 4.6). In Section 4.7 we finish with a summary of the different elements addressed during the chapter.

4.1 Feedback Control Loops (FCLs)

Autonomic systems are characterized by their ability to devise and apply counter-measures when necessary, including the ability to detect adaptation situations. These self-management systems can perform such activities based on situations they observe or sense in the IT environment rather than requiring human intervention to initiate the task. In general, the following properties characterize the self-management systems [Hariri *et al.*, 2006b, Kephart and Chess, 2003, Menascé and Kephart, 2007]:

Self-configuration stipulates that the system shall be capable of adapting its behavior to the execution context. For instance, it should be possible to add or remove some functionalities (*e.g.*, a business component providing a service) without a complete interruption of all the services. Moreover, system parts that are not directly impacted by the changes should progressively adapt themselves to these changes;

Self-optimization states that the system should control and monitor the system resources it consumes. The system should then detect a degradation of service and cater with the necessary reconfigurations to improve its own performance and efficiency;

Self-healing establishes that the detection, analysis, prevention, and resolution of damages should be managed by the system itself. The system should overcome hardware and software failures;

Self-protecting aims at anticipating, detecting, identifying and protecting against threats. System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent system-wide failures.

The four *self* properties enclose the main concerns associated with system maintenance. By automatizing their achievement, the burden of system managers is considerably reduced. However, the building of fully autonomic systems remains a challenge. As it will presented in Section 4.3, most of the existing solutions focus only in providing partially some of these properties. In this dissertation, we contribute in the conception of this kind of applications by combining Autonomic Computing concepts with resource-oriented approaches, SOA and CBSE principles.

MAPE-K Model

The properties present in self-management systems are reached by clearly identifying the different phases that compose the adaptation of applications, which is materialized as Feedback Control Loops. These phases are stated in the *MAPE-K* model (cf. Figure 4.1). *MAPE-K* states for Monitoring, Analysis, Planning, Execution and Knowledge. The first four names are the control loops phases. The last one represents the standard data shared among these phases, such as symptoms and policies, which must be complete (*i.e.*, including the whole aspects influencing adaptation decisions), modifiable (*i.e*, following the application changes), and at a high-level of abstraction (*i.e*, comprising only relevant information). The Monitoring phase encloses the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) retrieved from a managed resource. In the Analysis phase the information retrieved in the previous one is correlated in order to model complex *adaptation situations*. The Planning phase encloses the mechanisms that determine the actions for achieving the goals and objectives identified in the analysis. The Planning uses *adaptation policies* information to guide its work. The Execution phase groups the mechanisms that control the execution of an *adaptation plan* with considerations for dynamic updates. The *MAPE-K* model also defines the *Managed Resources* as entities that exist in the runtime environment of an IT system and that can be managed. These resources include the Execution Environment, Supporting Platform, and Application entities. All these managed resources provide Sensors to introspect the entity states, and Effectors to reconfigure them dynamically.

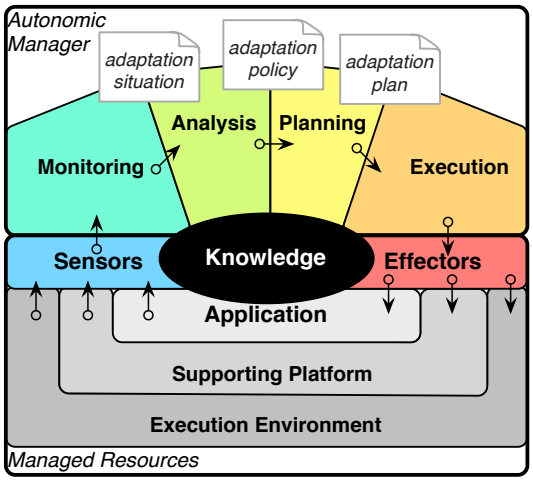


Figure 4.1: Overview of the *MAPE-K* Autonomic Control Loop.

4.2 Relation Between the Autonomic Computing and the Context-Aware Computing

As already said, the autonomic computing states the development of applications that have *self-** properties. These properties reduce the responsibilities of the system administrator, who just has to specify high level policies indicating how the system can be adjusted at runtime. To achieve it, this kind of computing clearly identifies the different tasks that compose the adaptation process as well the data required in the different tasks. As in any process, the flow of information is vital for reaching the adaptation goal.

For its part, the ubiquitous computing, and more specifically, the context-aware computing proposes the development of applications that have the ability of reacting by themselves to the environment changes. In this computing, context information is the key for the adaptation. The

rules that determine the changes according to this information can be expressed as high level policies. These rules can be also considered as context information. In autonomic computing, the adaptation actions are also inferred according to the current situation. Therefore, we can consider that the two kinds of computing are complementary approaches [Klein *et al.*, 2008].

In particular, context-aware applications can be conceived by following the feedback control loop principles. The data exchanged in the different steps in the loop corresponds to the context information collected from the environment. In each phase of the FCL, this information is processed for producing higher level information that will be used in the next phase. The result of the FCL is the execution of context-based adaptation of applications running in the environment with a minimal human intervention. In the next section we present some approaches combining context-awareness and principles from autonomic computing.

4.3 Autonomic Solutions

In this section we present some of the existing solutions dealing with the adaptation of applications by applying the autonomic computing principles. In order to identify the advantages and limitations associated with these solutions, we use the same dimensions defined in Section 3.2:

- *Technology and Paradigms Applied*,
- *Communication Mechanisms*,
- *Context Information*,
- *Loose Coupling*,
- *Independent Communication Mechanism*,
- *Adaptation*.

4.3.1 JADE: A Middleware for Self-Management of Distributed Software Environments

JADE [Boyer *et al.*, 2005, Bouchenak *et al.*, 2006, Depalma *et al.*, 2008] is a middleware solution enabling the autonomous reconfiguration of distributed applications. In particular, the platform combines the FRACTAL Component Model and Control Loops principles in order to enable the software management by programs. JADE wraps legacy systems into FRACTAL components to provide a *management layer*, which provides introspection and reconfiguration capabilities. Such functionality is provided by a uniform interface for the managed resources. This means that the control interface is the same for all the encapsulated software but the wrapper implementation is specific to each legacy system. The specified functionality defined by the interface includes the management of attributes, bindings and the system lifecycle.

The implementation of the autonomic behavior is materialized by means of feedback control loops. `Autonomic Managers` are introduced for the realization of the loops. JADE provides generic `sensor` (for event detection), `actuator` (for execution of reconfigurations) and `analysis/decision` (that represents the reconfiguration algorithm) components for the building of Autonomic Managers. Specifically, the platform provides managers for *self-optimization* and *self-healing* of J2EE based applications.

Characterization in Terms of the Dimensions. JADE control loops enables adaptation of J2EE applications at runtime. The interaction between the different elements of the loop is done via HTTP. The middleware solution does not deal with context processing and discovery issues.

4.3.2 Agent-based Middleware for Context-Aware Services

In [Soldatos *et al.*, 2007] authors present a middleware solution that makes context-aware service creation easier by enabling developers to focus on the service logic and not on the middleware concerns. The platform provides reusable functionality for controlling sensors and actuators in smart rooms, management of user access to services and modeling of contextual states. To do this, [Soldatos *et al.*, 2007] proposes an agent-based architecture with the following elements:

1. *Core Agents*: They allow communications between the distributed entities of the system as well as the extensibility of the platform. Further, they are independent from the concrete environment where the platform is deployed. Core agents include the *Device Desktop Agents*, *Device Agents*, *Personal Agents* and *Agent Manager*. The Device Desktop Agents implement the user interface for service access. The Device Agents enable the communication between the different devices and the framework. The Personal Agents permit interaction between users and the Agent Manager, which provides support for incorporating new services in the platform.
2. *Basic Service Agents*: Represent basic services that are tightly coupled with the smart rooms. Basic Agents include agents for situation recognition (*Situation Watching Agents*) and control of sensor and actuators (*Smart Room Agents*).
3. *Ubiquitous Service Agents*: Implement the non-intrusive service logic of various context-aware services.

According to authors, this middleware solution provides typical autonomic features of agent-based platforms such as persistence, cloning and migration of components to other hosts. Additionally, based on JADE platform (cf. Section 4.3.1), the different agents in the platform are extended with capabilities for querying the status of other agents. This characteristic is exploited for downgrading and upgrading the agents functionality according to the availability of other agents. The described platform also provides *self-healing* functionality, which is achieved by means of agent migration.

Characterization in Terms of the Dimensions. The agent-based architecture of the middleware solution enable its adaptation at deployment and runtime. To deal with discovery issues, JADE introduces a Knowledge Base Server that has the role of service registry. The NIST Smartflow middleware [Fillinger *et al.*, 2006] is exploited for interactions. Even if the middleware solution pretends to make the development of context-aware services easier, it does not consider issues associated with the context such the QoC properties and its representation.

4.3.3 Framework for Autonomic Context-Aware Service Composition

[Bottaro *et al.*, 2007] propose a framework for building home services. The proposal benefits from service-oriented computing and autonomic managers concepts for enabling context-aware service compositions. The principal elements of the framework include a `Service-Oriented runtime`, a `Context Service` processing information from context sources and `Autonomic Handlers` attached to the business services. The employed `Service-Oriented runtime` is IPOJO [Cervantes and Hall, 2004], an OSGI-based service-oriented component framework. On the other hand, the `Autonomic Handlers` monitor the execution context in order to modify the bindings or life cycle of the application components. The handlers are configured at runtime by using high-level policies.

In order to identify the required reconfiguration actions and guarantee service continuity, an autonomic handler confronts the requirements specified by a binding policy to the list of available services. If one of the requirements of the policy is not respected, the handler invalids the component (*i.e.*, the handler un-publishes the service and announces its departure) and unbinds the used providers. When there is more than one service satisfying the requirements, the handler ranks and selects the best one according to a ranking policy.

Characterization in Terms of the Dimensions. The described framework combines CBSE and SOA by means of IPOJO. The runtime adaptation capabilities of the platform are limited to the application level. The Context Service of the approach excludes the QoC properties. Regarding the communication aspects, the autonomic handlers enable the usage of different discovery mechanisms. However, information about the interaction mechanisms is not specified.

4.3.4 Adaptation Platform for Autonomic Context-Aware Services

[Cremene *et al.*, 2008] proposes a solution for dynamically discovering the context structure and the adaptation strategies. To do this, authors defines general Application-Context (A-C) descriptions that are specialized for producing Specific A-C descriptions that can be reconfigured by means of strategies discovered at runtime. The different components of the applications must be described in terms of their structure, syntax, semantic and behavior. The solution defines an architecture based on Feedback Control Loops composed by the following elements:

1. *Adapted part*: Represents the application to be adapted. This application is executed on a reflective platform that provides introspection and reconfiguration functionalities.

2. *Observer*: This module extracts information about the context and the services by means of dedicated components called *observers*. To do this, the module has the *Context Discovery Manager* and the *A-C Description Manager* submodules. The former decides the aspects to be considered and searches for observer components. The latter creates and updates the A-C descriptions at runtime.

3. *Controller*: Is divided in two modules. An *Adequacy Verifier*, which checks the suitability between the application and the context, and a *Strategy Search Engine* for searching the correct strategy when the application does not fit its context.

4. *Component repository*: Stores reusable components including observers. The *Controller* component uses these components to adapt applications when required.

Characterization in Terms of the Dimensions. The presented middleware solution, based on the Corba Component Model [Group, 2006b] (CCM), supports dynamic adaptation of applications. Considering the context aspect, the specified context processing does not include the QoC properties. The found description about the approach does not provide information about the flexibility in terms of communications

4.3.5 The ANS (Autonomic Network Services) Framework

ANS [Huebscher *et al.*, 2007, Huebscher and McCann, 2005] is a framework for the monitoring of medical patients at home. The framework applies autonomic principles to adapt itself to different activities carried out by the user, which depends on temporal and locational aspects. ANS is executed on sensor nodes spread in home. Each node can have many functions in order to increase reliability. These functions are materialized as components implemented with the most suitable language according to the task. On the other hand, the logic determining the functionality of reconfiguration is implemented by means of *Tesserae*, a lightweight language that makes part of the framework.

In ANS, the context information and its quality enable autonomic intelligence throughout the architecture. In particular, the autonomic rules defining the required configurations (*i.e.*, the binding/unbinding of context services) exploit QoC information for selecting the most suitable context services for applications. The framework identifies *precision*, *probability of correction*, *resolution*, *up-to-dateness* and *refresh rate* (cf. Section 3.1) as common QoC attributes for different kinds of context information. These attributes have to be provided by the context services and updated regularly because they can eventually change.

Considering the autonomic properties (cf. Section 4.1), the platform supports *self-healing* and *self-optimization* in terms of context providers. The *self-healing* is guaranteed because the framework can identify the failures of context services and replace them. The *self-optimization* is given by exploiting the QoC of the available context providers. The most suitable context provider is selected using utility functions. Each application type has its own utility function.

Characterization in Terms of the Dimensions. The ANS component based architecture enables its dynamic and static adaptation. The keystone of the autonomic intelligence in the approach is the QoC properties. ANS also supports discovery in order to monitoring the context providers. For interactions, low level protocols are used. In both aspects, interactions and discovery, ANS permits the addition of new protocols.

4.3.6 MIddleware DemonstrAtor Server (MIDAS) Framework

MIDAS [Mohyeldin *et al.*, 2005] is a service-based framework for building context-aware applications that adapt their behavior according to dynamic radio resource restrictions like available bandwidth and link interruptions. In MIDAS, the architecture is self-descripting, *i.e.*, the system model is part of the system. The framework identifies four basic service types: *i) Sensors*, for data retrieval; *ii) Interpreters* for data processing; *iii) Actuators* for deploying instructions; and *iv) Context elements* representing distributed buffers decoupling the sensors and interpreters. This context elements buffer information about the system states, the system model itself and the system environment. The framework also defines a *model activator* service that makes the adaptation decisions within the system model. As the activator is also part of the system model, it can be affected by such adaptation decisions.

Characterization in Terms of the Dimensions. The MIDAS framework provides support for mobility and flexibility in terms of interactions. However, the framework description does not provide information about the discovery capabilities as well as the interaction protocols. In terms of adaptation, the framework enables static and dynamic adaptations at the application and middleware levels.

4.3.7 AutoHome: an Autonomic Management Framework for Pervasive Home Applications

`AutoHome` [Bourcier *et al.*, 2010] is a service-oriented framework for building autonomic home applications. The goal of the platform is to separate the implementation of autonomic functions from the application itself. To to this, `AutoHome` framework proposes an architecture composed by the following elements:

1. *A Middleware for Autonomic Service-Oriented Applications*: This element contains a context facility and an *Autonomic Service-Oriented Component Runtime*. This component runtime proposes generic *touchpoints* for monitoring and control of pervasive applications. These touchpoints represent the interface between autonomic managers and applications. In other words, they allow the dynamic reconfiguration meaning the replacement of services at runtime. `AutoHome` identifies touchpoints for monitoring and reconfiguration activities. The touchpoints for monitoring include property value, method invocation, and container's state. The touchpoints associated with reconfiguration activities are the property value modification, method invocation and container's state reconfiguration.

2. *Service-Oriented Applications*: Are the applications built on the top of the framework, which will be managed in an autonomic way.

3. *Autonomic Managers*: Organized in a hierarchy, the framework offers `Gateway Managers`, `Application Managers` and `Service Managers` as autonomic managers. The

`Gateway Managers` manage the gateway performance and solve conflicts between applications. The `Application Managers` controls the life cycle of applications, *i.e.*, they can start, stop, add or withdraw applications. Finally, the `Service Managers` deals with the instance configurations and their behavior adaptation.

Characterization in Terms of the Dimensions. The `AutoHome` framework is an extension of iPOJO [Escoffier *et al.*, 2007], a service-oriented component runtime built on the top of OSGi. The framework offers the dynamic reconfiguration of the service-oriented applications by allowing service replacement. At the middleware level, static adaptations are supported via the configurations of the autonomic manager hierarchies. The interaction and discovery mechanisms of services are the responsibility of the iPOJO component containers and they can be modified at runtime. Regarding the context management, even if the framework provides a context facility, it does not consider the QoC properties.

4.3.8 MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments

`MUSIC` [Rouvoy *et al.*, 2009] is a component-based planing framework for context-aware adaptations, that extends the functionalities offered by the MADAM platform [Floch *et al.*, 2006, Hallsteinsen *et al.*, 2004]. MUSIC models applications as component frameworks, which define the functionalities that can be modified at runtime. The selection of a new configuration in response to context changes is done by using an utility function that determines the QoS (Quality of Service) for the alternative configurations. To do this, applications have a QoS-aware model that focuses on the QoS properties of the different components that make part of them. The values of these properties can be static (*i.e.*, they are not modified according to the current status of the service) or dynamic (*i.e.*, they can change over the time). The existence of dynamic QoS properties makes the negotiation of the Service Level Agreement [Dan *et al.*, 2003] (SLA) necessary. The MUSIC platform provides a negotiation protocol that is inspired by the WS-Agreement specification [Andrieux *et al.*, 2005].

The architecture of the platform includes the following components: *i)* the `Context Manager` that monitors context changes, *ii)* the `Adaptation Controller` which coordinates the adaptation process, *iii)* the `Adaptation Reasoner` supporting the execution of planning heuristics, *iv)* `Plan repository` that keeps plans associated with the different interfaces, *v)* the `QoS Manager` that provides the QoS values required for evaluating the utility of each configuration, *vi)* the `Configuration Executor` handling the reconfiguration process, *vii)* the `Service Discovery` that advertises and discovers services by using different discovery protocols, and *viii)* the `Remoting Service` creating the bindings between service clients and service providers.

Characterization in Terms of the Dimensions. The MUSIC platform exploits OSGi, which makes the provisioning of adaptation at the middleware and application level easier. Flexibility in terms of discovery protocols is done by means of the `Service Discovery` component. The support for mobility is guaranteed thanks to the the SLA monitoring executed by the platform and the associated service replacement. The interactions employ HTTP as underlaying protocol. In terms of context processing, the platform excludes the QoC properties management.

4.3.9 Rainbow

`Rainbow` [Garlan *et al.*, 2004] is a generic framework for supporting self-adaptation of software systems. The framework defines a control loop to monitor properties of systems at runtime, detect constrains violation and performs global or module level adaptations when required. The loop is divided in two parts: *i)* the *Adaptation Infrastructure* that reifies the reusable functionality

of the framework, and *ii)* the *System-Specific Adaptation Knowledge* that is specific to each system. The Adaptation Infrastructure is composed by the following layers:

- The *System-layer Infrastructure* that represents the system access point. This layer provides a monitoring mechanism, implemented as probes, that measures properties of different systems. The measurement information is published and consumed by probes. Additionally, the layer provides a *resource discovery* mechanism and an *effector* mechanism for executing the system modifications.

- The *Architecture-Layer Infrastructure* aggregates the information from the probes and updates the system architectural model via a *model manager*. A *constrain evaluator* checks for restrictions violation and triggers adaptation by using an *adaptation engine* that decides the required actions to execute.

- The *Translation Infrastructure* allows the mapping of information between the system and the model in both senses. This layer has a *translator component* sharing mappings that are stored in a *translation repository*.

In order to use the functionalities from the Adaptation Infrastructure, systems have to specify information such as the operation model (including component types and properties), behavioral constrains, and adaptation strategies. This information is the System-Specific Adaptation Knowledge that composes the `Rainbow` control loops.

Characterization in Terms of the Dimensions. `Rainbow` is built on the top of the Acme architectural style toolset [Garlan *et al.*, 2000]. The interactions employ XML messages and are done via Java RMI. The provided discovery facility is based on resource types and is limited to resources controlled by the platform. Regarding the adaptation, the `Rainbow` control loop enables dynamic adaptation at the application level. The middleware can be customized by adding the required information for adaptation of the concrete systems. In the adaptation process, context information is not considered.

4.3.10 Distributed Autonomous Component-Based ARchitectures (DACAR) Solution

`DACAR` [Dubus and Merle, 2006] is a solution, based on OpenCCM [Briclet *et al.*, 2004], for the development of control loops adapting CORBA component-based applications. The solution defines a metamodel that combines OMG Deployment and Configuration (D&C) specification [Group, 2006c] and Event-Condition-Action (ECA) rules. The metamodel has a *platform part*, a *knowledge part* and *third part describing high level autonomic policies*. The platform part is encapsulated in a software component providing operations for monitoring and instantiation of applications. The implementation of such component depends on the platform selected to execute the applications. The knowledge part is modeled by using OMG D&C descriptors, which are reified as Java objects that represents the executable model of running applications. The rules are implemented as lightweight components by exploiting the Fractal Component Model.

For modeling the ECA rules, `DACAR` considers two kinds of events: *endogenous* events and *exogenous* events. The endogenous events are associated with the knowledge part of the control loop. The knowledge part is represented as model that produces events when concepts or values are modified. The exogenous events comes from the execution platform.

`DACAR` defines three kinds of rules:

- *Monitoring Rules*, which are triggered by exogenous events. These rules act on the knowledge part of the control loop when changes are detected in the running platform. Monitoring Rules are generic, meaning that they can be used in different kinds of applications.

- *Deployment Rules* that are executed when endogenous events are generated. They update the execution platform when the knowledge part is modified. As Monitoring Rules, Deployment Rules are also generic.
- *Architectural Rules* triggered by endogenous events. They perform actions on the knowledge part modifying it according to the properties that this knowledge part has to fulfill.

Characterization in Terms of the Dimensions. DACAR provides dynamic adaptation at the application level by defining ECA rules. The adaption at the middleware level is not considered. All the communication issues are delegated to the execution platform—*i.e.*, OpenCMM. Therefore, the flexibility in terms of interactions and discovery depends on the concrete implementation of the approach.

4.4 Limitation of the Existing Approaches

In this section we discuss the main drawbacks associated with the approaches that we have presented in the previous sections. Table 4.1 summarizes the characteristics of such approaches.

Regarding the adaptation capabilities, we observe that thanks to the autonomic principles the different approaches provide dynamic configuration at the platform or application level. However, as discussed before, the reconfigurations at runtime in both levels are key in ubiquitous environments in order to deal not only with the heterogeneity in terms of communications but also for improving application behavior benefiting from the context information. Therefore, there is a **lack of flexibility** in terms of where the adaptation is supported at runtime.

Other important issue with the autonomic approaches is **the low priority given to the information integration**. Although the provision of autonomic adaptation is the focus of the described approaches, the information flow is a keystone for this adaptation. Therefore, considering the different communication paradigms as well as hardware executing the applications, it is necessary to provide a solution dealing with interaction and discovery concerns. In particular, the solution supporting the context-based adaptation should be aware of limitations of devices running the adaptive applications. These limitations do not only affect the non-functional aspects of the adaptation but also the decision making of the Feedback Control Loops.

Associated with the issue mentioned in the previous paragraph, we find the **lack of relevance given to context information**. Dimensions such as the QoC and multiple representation are not considered. Only the ANS framework exploits Quality of Context in order to trigger reconfiguration processes. This omission is not suitable because the adaptation is based on the context information. Therefore, if the correct information can be not retrieved, the resulting adaptation can not reflect the expected behavior according to the current environment state.

Finally, although some of the approaches provide self-configuration (cf. [Huebscher *et al.*, 2007, Bottaro *et al.*, 2007]) in a certain degree, they assume a controlled environment where the different Feedback Control Loop's participants are always present or at least one replacement. This property is not suitable when the adaptation logic is not necessarily centralized in one entity. If the different responsibilities of the adaptation process are spread between entities (*e.g.*, in the concrete case of mobile devices running the adaptive applications), interaction problems should be not a problem for executing the adaptation. This autonomy should also make part of the self-configuration property of the solution. Thus, a **high dependency level** between the entities that compose the Feedback Control Loop represents a key issue in the presented autonomic approaches.

4.5 State-of-the-Art Synthesis

In the state of the art of this dissertation, we have presented the principal concepts, technologies, and elements that we apply in our approach. In particular, we focused our discussion on

Dimension / Middleware Approach	Technology and Paradigms applied	Communication Mechanism		Context Information		Loose Coupling		Independent Communication Mechanism		Adaptation		
		Interaction	Discovery	Support for Multiple Representations	QoC	Mobility Support	Dependency Definition	Interaction	Discovery	Static	Dynamic	Level
Agent-based Middleware for Context-Aware Services	Agents, Components	NIST Smartflow middleware (NSFS)	Knowledge Base Server for the registry/unregistry of services	✗	✗	✗	Component Interfaces	✗	✗	✓	✓	Middleware
ANS	Components, sensors, tesserae language, utility functions	Low level protocols	Applied in the monitoring of context providers	✗	✓	✗	QoC	✓	✓	✓	✓	Middleware
Adaptation Platform for Autonomic Context-Aware Services	Graphs, CCM, SOA	N/S	N/S	✗	✗	N/S	Component Interfaces	N/S	N/S	✗	✓	Application, Middleware
Framework for Autonomic Context-Aware Service Composition	Service Component Model, SOA, iPOJO, Policies	N/S	Service advertisement when it is valid	✗	✗	✗	Component Interfaces	N/S	✓	✗	✓	Application
JADE	J2EE, Fractal	HTTP	N/A	N/A	N/A	N/A	N/A	✗	N/A	✓	✓	Application
MIDAS	Components	N/S	N/S	✗	✗	✓	Proxies (Context elements)	✓	N/S	✓	✓	Application, Middleware
AutoHome	iPOJO (Components and SOA)	Component Container	Component Container	✗	✗	✗	Client and Server Interfaces	✓	✓	✓	✓	Application, Middleware
MUSIC	OSGi, SOA, WS-Agreement, utility functions	HTTP	Service Discovery Component	✗	✗	✓	Component Interfaces, Service Descriptions	✗	✓	✓	✓	Application, Middleware
Rainbow	Components, Acme, RMI, XML	XML messages over RMI	Resource Discovery Mechanism	N/A	N/A	✗	System-Layer Infrastructure	✗	✗	✓	✓	Application (dynamic), Middleware (static)
DACAR	OMG D&C, ECA rules, OpenCCM, Fractal	Provided by OpenCCM	CORBA Trading Service	N/A	N/A	✗	Component Interfaces	✗	✗	✗	✓	Application

Table 4.1: Different Autonomic Approaches

paradigms for the construction of SOA based applications (cf. chapter 2), ubiquitous computing (cf. chapter 3) and autonomic computing.

In the SOA part we have introduced some mechanisms for service API conception and the structuring of service-oriented applications. We have chosen the REST architectural style and the SCA component model as pillars to define an elegant and suitable approach for integration in ubiquitous environments. In the context of ubiquitous computing we have defined key elements in our approach such as the QoC information and discussed some existing approaches for context integration as well as for service discovery. Regarding the autonomic computing, we have defined Feedback Control Loop principles that we will exploit for enabling the adaptation of applications. The reconfiguration capabilities of our approach are based on the reflection capabilities of the FraSCAti platform, which supports execution of SCA applications.

4.6 Dissertation Challenges

Considering the approaches and concepts discussed in this chapter, in this section we include and complement the challenges defined in chapter 3. These challenges are presented below:

1. *How to deal with communication heterogeneity*: The plethora of devices, services and applications available in ubiquitous environments promote the proliferation of different kinds of interaction and discovery paradigms. A solution integrating context information and enabling application adaptation has to provide the flexibility for selecting the most suitable mechanisms in different situations. This has to be done considering the hardware and software variability of clients and service providers. Therefore, flexibility is vital in a middleware solution dealing with this kind of heterogeneity.

2. *How to deal with context heterogeneity*: As already stated, in ubiquitous environments the context information is a key for the adaptation of applications. However, this information is affected by the heterogeneity that becomes the rule and not the exception in this kind of environments. In particular, the context provider can make the information available in different ways and the context can be characterized by different properties such as the freshness and precision (maybe relevant for the adaptation of applications). Therefore, the approaches dealing with context-based adaptation should consider the relevance of context and focus on it in order to provide a suitable result.

3. *How to provide support for adaptation*: The integration of context information is the first step for enabling context-based adaptation. A solution dealing with such kind of adaptation has to enable not only the reconfiguration of application but also of the platform itself. In a similar way, the solution should consider that the changing nature of ubiquitous environments makes the distribution of the adaptation concerns necessary. This means that the different adaptation responsibilities identified by the autonomic computing, *i.e.*, context gathering and processing, the determination of required reconfigurations and the execution of the adaptation itself, can be spread between different entities. With this distribution, we need to face the different kinds of heterogeneity and the mobility already mentioned. Therefore, a global approach allowing adaptation of context-aware applications and considering these issues is required.

4.7 Summary

One of the objectives of this chapter was to present some approaches dealing with context-based adaptation by applying principles from the autonomic computing. We have also discussed some issues associated with these approaches such as the lack of flexibility and, the low priority given to the integration and context information. Other objective of this chapter was to show that

context-aware computing can benefit from autonomic computing for the conception of context-aware applications. On the other hand, we have summarized the three chapters of the state of the art in order to illustrate how they will be used in our proposal. Finally, we have presented again the challenges associated with adaptation and heterogeneity in terms of communications and information that we need to tackle for building context-aware applications. In the following chapters we introduce our proposal in order to face these different challenges.

Part II

Contribution

Chapter 5

Enabling Context Mediation In Ubiquitous Environments

Contents

In order to present the contribution of this dissertation, we divide it in two parts: *i)* **the approach enabling the integration of context information** and, *ii)* **the approach for context-based adaptation in ubiquitous environments**. This chapter focuses on the first part of contribution. The second one will be addressed in Chapter 6.

Motivation

In ubiquitous environments, the development of context-aware applications, which are adapted according to the current environment conditions, requires to consider several issues in terms of heterogeneity, mobility and distribution. The developer of this kind of applications should focus only in providing the modularity and flexibility points required for adapting the applications as well as in the business logic. However, this is not always possible. In particular, considering the challenges described in Section 4.6, the context-aware application developers need to face the following issues:

1. **Heterogeneity of Services and Context Providers:** The available services in ubiquitous environments can be conceived using different architectural styles and developed with different technologies. In a similar way, they are running on several kinds of computational devices. This freedom becomes a problem when we need to integrate different kinds of service providers in order to enable context-based adaptation. In particular, this heterogeneity makes the interoperability difficult, and requires that the applied implementation

technologies be transparent as possible. However, it is not always the case: depending on the selected architectural style, the consumers and providers can be more or less coupled and more or less aware of how each one is implemented. Furthermore, although a common model for interaction is always required, it is not always the most suitable to make the development and evolution of consumers and providers easier.

2. **Heterogeneity of Protocols:** The heterogeneity in terms of services is also due to the lack of standard protocols for locating services and interacting with them. In general, each service provider selects the most convenient communication mechanisms for its context. Therefore, when a client can only communicate via certain protocols, its service access is limited. This means that some context information can be missed in the adaptation of context-aware applications, and can result in an unexpected behavior.

3. **Mobility of Service Consumers, Providers and Context-Aware Applications:** The proliferation of smartphones makes ubiquitous environments highly changing. Thanks to the presence of these devices, service consumers, providers and context-aware applications can join and leave the environment at anytime. Hence, smartphones increase the available resources that can be (or must be) used in the adaptation of applications. However, if this mobility is not managed at all, the dynamic nature of ubiquitous environments is wasted and the developed applications are not completely context-aware.

These issues are mainly associated with the integration of context information in the adaptation process. For this reason, before enabling adaptation we need to deal with such issues.

Contributions of the Chapter

In this chapter, we focus on the description of our contribution, which enables ***transparent integration of heterogenous context providers and consumers***. To do this and regarding the heterogeneity in terms of protocols, we define a generic and simple approach based on software connectors, standards and resource-oriented principles [Romero *et al.*, 2010c, Romero *et al.*, 2010a, Romero *et al.*, 2010d]. Software connectors provide the elements for isolating communication concerns from context processing. As stated in the introductory chapter, the usage of standards supports reuse and portability of the application in different situations. Finally, the resource-oriented principles foster simplicity by focusing on the exchange data, *i.e.*, context information.

Our approach provides versatility in terms of communication mechanisms and supports the ***advertisement of context information as resources*** to deal with mobility. In particular, we propose a mechanism for accessing context information on demand. This information is advertised using QoC properties [Buchholz *et al.*, 2003] that context consumers use to select the most suitable provider according to their requirements [Romero *et al.*, 2010b]. We also make the ***concretization of the approach*** by incorporating it into the SCA component model. As we will present in Chapter 6, we benefit from the combination of SCA and our approach for building ubiquitous feedback control loops, which adapt context-aware applications. In this way, we summarize the contributions discussed in this chapter as follows:

1. **Metamodel Supporting the Notion of Context as a Resource**: The integration of context information requires the consideration of the heterogeneity and mobility issues already mentioned. In particular, we need to identify the different elements and concepts that will help us in the conception of our approach. Concepts such as context providers, context consumer, protocols and their relationships have to be considered. Therefore, regarding the relevance of context information in the adaptation of applications, we model this information as resources that can be provided and accessed by different kinds of devices. By using the notion of context as a resource, we are able to provide a simple but still elegant and complete solution for the integration of context information. Therefore, the metamodel represents the keystone of our proposal.

2. **Generic Connectors Enabling the Integration of Context Providers and Consumers**: Using the context as a resource metamodel, we introduce the notion of ubiquitous connectors, which encapsulate the integration responsibilities. These connectors have the particularity of employing standards and simple approaches for integrating context providers and consumers. In particular, we exploit REST principles and well defined protocols in their conception.

3. **Architecture for Interaction and Discovery of Context Resources**: By leveraging on the ubiquitous connectors, we define a generic and extensible architecture that includes and modularizes the different elements from the metamodel. This architecture is specialized by extending the SCA component model with the required functionality for supporting simple exchange of context information.

Chapter Organization

Considering the previous contributions, the rest of this chapter is organized as follows. We start by presenting the properties that our approach has to respect (cf. Section 5.1) before giving an overview about this approach (cf. Section 5.2). We continue with the keystone of the approach—*i.e.*, the metamodel that we use to support the notion of *Context as a Resource* (cf. Section 5.3). Then, we define the software connectors for dealing with the issues mentioned before (cf. Section 5.4), which bring support for QoC-based discovery (cf. Section 5.5). After that, we introduce the architecture associated with these connectors (cf. Section 5.6) as well as its incorporation into the SCA for supporting the context changes (cf. Section 5.7). Finally, we summarize the contributions of the Chapter (cf. Section 5.8).

5.1 Properties for Context Mediation

Before discussing our approach, we describe what we expect from a solution dealing with the exchange of context information. Therefore, regarding the heterogeneity and dynamism of the different entities that interact in the context-based adaptation, we identify the properties that our solution exhibits:

1. *Independent representation of context information*: The heterogeneity of the interacting entities requires the information exchange via different formats. However, the use of a specific representation should not impact producers implementation;

2. *Independent communication mechanisms*: The underlying protocols should be transparent for consumers as well as for producers;

3. *Loose-coupling between context producers and consumers*: Because of the dynamicity of ubiquitous environments, the producers should be able of interacting with different consumers and vice versa. This means that the hard-wired associations should be avoided. Furthermore, because context managers and context-aware applications focus on the exchange of context information, the association should be at the data level and not at the interfaces used to share this information. In fact, the interaction mechanisms should be selected according to resource constraints of computational entities;

4. *Flexibility and Extensibility*: This property is required in order to provide the previous ones. The heterogeneity in terms of technology and communication requires an adaptive solution allowing the inclusion of new context representations, communication protocols as well as the possibility of its customization. Figure 5.1 summarizes the goal, challenges (cf. Section 4.6) and properties searched in our solution.

Our approach satisfies these principles by combining existing standards and technologies. In particular, respecting the goals defined in Section 1.2, we apply CBSE and SOA principles as

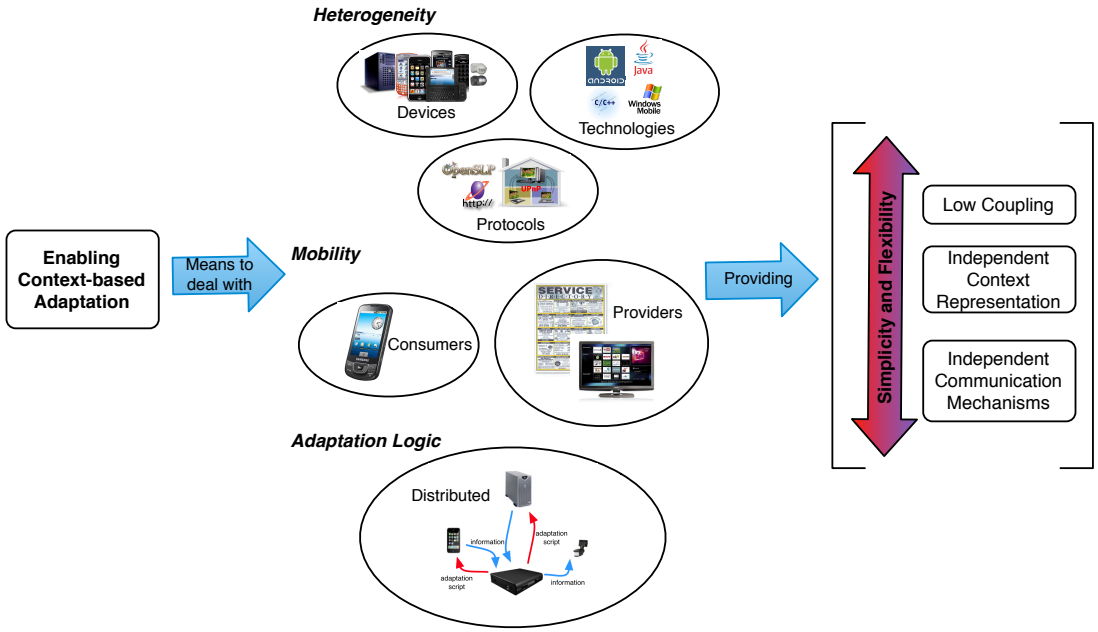


Figure 5.1: Properties Searched in the SPACES Conception

well as a resource-oriented approach. The combination of these paradigms allows us to provide a clear concern modularization, which is vital for providing a flexible and extensible solution. Regarding the existing middleware solutions (cf. Sections 3.2 and 4.3), we see that they precisely lack this flexibility.

On the other hand, several solutions provide support for the principal functionalities in context-awareness—*i.e.*, discovery, interaction and some kinds of adaptation (in the application or/and middleware levels). However, as discussed in Chapters 3 and 4, there are some limitations associated with these functionalities. For instance, several solutions fail to support the usage of new interactions and discovery protocols. Even some of them define their own protocols, restricting the incorporation of third part service providers. In a similar way, some approaches are not easily customizable. This means that it is not possible to have different personalities of the platform for deploying it on devices with distinct specifications and capabilities. Furthermore, the way these solutions are conceived makes the integration of services or applications that are not part of the platform difficult. Regarding the offered adaptation, some approaches do not provide support for reconfiguration at the application level. This means that the developer has to worry for exploiting the available context information in order to adapt the application using this information. Finally, almost all the solutions do not consider the relevance of context information, omitting the support for the QoC properties and different representations as important issues in the integration of this kind of information. The context consumers should be able to select the available information according to their needs and not just because it is available. Table 5.1 summarizes the satisfaction of the properties for context mediation by different state-of-the-art approaches. For more details about the different dimensions considered in this table see Section 3.2. In the rest of this chapter, we provide more details about our approach and how it deals with the integration issues. The contribution associated with the adaptation will be introduced in the next chapter.

	Dimension / Middleware Approach	Context Information		Loose Coupling		Independent Communication Mechanism		Adaptation		
		Support for Multiple Representations	QoC	Mobility Support	Dependency Definition	Interaction	Discovery	Static	Dynamic	Level
Autonomic Approaches	Agent-based Middleware for Context-Aware Services	✗	✗	✗	Component Interfaces	✗	✗	✗	✓	Middleware
	ANS	✗	✓	✗	QoC	✓	✓	✓	✓	Middleware
	Adaptation Platform for Autonomic Context-Aware Services	✗	✗	N/S	Component Interfaces	N/S	N/S	✗	✓	Application, Middleware
	Framework for Autonomic Context-Aware Service Composition	✗	✗	✗	Component Interfaces	N/S	✓	✗	✓	Application
	JADE	N/A	N/A	N/A	N/A	✗	N/A	✓	✓	Application
	MIDAS	✗	✗	✓	Proxies (Context elements)	✓	N/S	✓	✓	Application, Middleware
	AutoHome	✗	✗	✗	Client and Server Interfaces	✓	✓	✓	✓	Application, Middleware
	MUSIC	✗	✗	✓	Component Interfaces, Service Descriptions	✗	✓	✓	✓	Application, Middleware
	Rainbow	N/A	N/A	✗	System-Layer Infrastructure	✗	✗	✓	✓	Application (dynamic), Middleware (static)
	DACAR	N/A	N/A	✗	Component Interfaces	✗	✗	✗	✓	Application
Service Discovery Solutions	INDISS	✗	✗	N/A	Events	✗	✓	✓	✗	Middleware
	ReMMoC	✗	✗	N/A	WSDL	✓	✓	✓	✓	Middleware
	A multi-protocol framework for ad-hoc service discovery	✗	✗	N/A	Component Interfaces	✗	✓	✓	✗	Middleware
Middleware Solutions for Context-Awareness	GAIA	✗	✗	✓	Channels	✗	✗	✓	✓	Application, Middleware
	GAIA Microserver	✗	✗	N/A	J2ME Proxy/ Channels	✗	✗	✗	✓	Application
	MUSIC Peer-to-Peer	✗	✗	✓	Peer Group	✗	✗	✗	✗	N/A
	Cortex	✗	✗	✗	Sentient Objects	✗	✓	✓	✓	Middleware
	CARMEN	✗	✗	✗	Proxies (MAs)	✗	✗	✗	✓	Middleware
	Aura	✗	✗	✗	Environment Manager	✗	✗	✗	✓	Middleware
	CARISMA	✗	✗	✗	Profiles	✗	N/A	✗	✓	Middleware
	Cooltown	✗	✗	✓	Directory Module	✗	✓	✓	✗	Middleware
	MiddleWhere	✗	✓	✗	Adapters	✓	N/A	✗	✗	N/A
	MobiPADS	✗	✗	✗	Channel Service, Mobilets	✗	N/A	✓	✓	Application, Middleware
	SOCAM	✗	✗	✗	Service Locating Service	✗	✗	✗	✓	Application
	RCSM	✗	✗	✗	Channels (CTCs)	✓	✓	✓	✓	Application
	Large Scale Peer-to-Peer Context Dissemination Middleware	✗	✓	✗	Context-based groups	✗	✗	✗	✗	N/A
	CAPNET	✗	✗	✗	Messaging Component	✓	✗	✓	✗	Middleware
	Ubiquitous Feedback Control Loops	✓	✓	✓	Data	✓	✓	✓	✓	Application, Middleware

Table 5.1: Comparison of Different Approaches Regarding the Properties for Context Mediation

5.2 SPACES Overview

The three contributions presented in this chapter, *i.e.*, the metamodel, the connectors and the generic architecture, enable us to build a *flexible* and *simple* solution to integrate heterogenous resources in ubiquitous environments. We call the result of combining the three contributions SPACES. The simplicity of this solution rests on the goal of *not reinvent the wheel* by reusing and combining standards and widely accepted technologies (cf. Section 1.2). For its part, the flexibility leverages on three main keystones: *i)* separation of mediation concerns [Taylor *et al.*, 2009] enabling the customization of the solution, *ii)* application of a resource-oriented approach for focusing in what really matters, *i.e.*, the context integration and *iii)* the application of CBSE and SOA principles to reduce coupling and foster reuse of the different elements that make part of the solution.

The separation of concerns strategy fosters spliting the problem into parts as independent as possible. With this idea in mind and regarding the already mentioned problems associated with context-aware applications development, we identify the following issues as relevant in context mediation:

1. ***Discovery***: The discovery capability is important because we need to detect or find the available context sources in the environment. When we know in advance all the available sources, this capability is not required. However, as already stated in Chapter 3, the ubiquitous environments are highly dynamic and therefore we cannot assume the existence of a static infrastructure configuration. Thus, the possibility of becoming aware of the services that are currently available in the environment constitutes an important advantage in context mediation and therefore it is present in our solution.

2. ***Interaction***: Once the available context providers are discovered, the next step is to interact with them. This means the retrieval of the information from the sources using different kinds of representations according to the consumer needs. If we can not ask for or retrieve the information, we cannot used it in the adaptation process of context-aware applications.

3. ***Support for Several Representations***: As already stated, flexibility is key for solutions providing context integration. The flexibility encloses the provision of different ways for representing the information, which is an important property when dealing with heterogenous devices.

4. ***Context Processing***: Once the information is retrieved, it has to be processed to produce higher level information or make decisions related to the required adaptation. In SPACES, we delegate the context processing to existing context managers such as the COSMOS framework [Rouvoy *et al.*, 2008, Conan *et al.*, 2007]

5. ***Adaptation***: The final step in context-awareness is the dynamic reconfiguration of applications. The data associated with these reconfigurations (*e.g.*, scripts and new functionality) can be also considered as context information that requires to be exchanged. Therefore, we see the adaptation as a process where the context information flow is important. We use this idea to build Ubiquitous Feedback Control Loops enabling the adaptation of context-aware applications. We provide more details about these feedback control loops in Chapter 6.

In order to consider these different issues, in SPACES we propose a metamodel (cf. Section 5.3) that helps us to clearly extract and define the elements that make part of context mediation. In particular, we materialize the first four issues (*i.e.*, discovery, interaction, multiple representations and context processing) and include other relevant aspects, such as the QoC properties. Therefore, the objective of the so-called SPACES metamodel is to foster the modularization of the different responsibilities and concerns associated with the exchange of context information.

From the SPACES metamodel, we derive resource-oriented connectors, which focus in the exchange of context. These connectors expose the context as resources and hide the current implementations associated with context producers and consumers. To do that, the SPACES connectors promote the usage of simple and standard interfaces and leave the establishment of the

dependency between clients and servers at the data level. The connectors support a clear separation between the resources (*i.e.*, the context information), their representation (*i.e.*, the format used to exchange the information) and how they are retrieved (*i.e.*, the current operations and protocols). This separation favors the use of simple communication mechanisms, which can be selected regarding the variable characteristics of devices hosting the consumers as well as the presence of legacy applications. Thus, in SPACES, we offer the possibility of customizing our solution by allowing the incorporation of interaction mechanisms at design time and at runtime.

Considering the independence of different responsibilities that we can obtain of component based approaches, we propose a generic architecture that provides a detailed view of the different elements that conform the SPACES connectors. Then, and by benefiting from the loose-coupling and reuse promoted by SOA applications, we bring SPACES into the SCA component model. In this way we can keep the best of CBSE and SOA together and still provide a reusable and simple solution for context integration.

5.3 Modeling Context as Resources: The SPACES Metamodel

To meet the different properties introduced in Section 5.1, we apply in SPACES a resource-oriented approach. Solutions following this kind of approach consider resources as first class entities, which control interactions between resource producers and consumers. For context-aware systems, context is key [Coutaz *et al.*, 2005], but context also represents resources that need to be propagated. Therefore, we start the design of our solution by modeling context information as resources. In this section we present the metamodel that makes it possible.

Figure 5.2 depicts the SPACES metamodel representing context as a resource. This metamodel is inspired in the REST service metamodel introduced in Section 2.1.2. The metamodel provides the foundations to build our solution since it includes all the elements and their relationships, which we consider relevant in context mediation. Below we discuss the different elements.

1. ***Context Element:*** We introduce the notion of `Context Element` in order to identify the different entities participating in the context mediation that have to be considered as first class citizens. `Context elements` include `Context Provider`, `Context Information`, `Context Consumer` and `Representation`. The different context elements are characterized by their name and description. We also introduce other entities that are not context elements, but that we consider for providing a suitable solution.

2. ***Context Provider:*** In context mediation, there are always producers and consumers of information. In the SPACES metamodel, the `Context Provider` represents the producers—*i.e.*, entities generating relevant information for `Context Consumers`. In the REST metamodel, these entities are the *Services*. A `Context Provider` has QoS attributes [Aurrecoechea *et al.*, 1998, Comuzzi and Pernici, 2009] describing non-functional aspects of the producer as well as connectors for exchanging the information. We decided to reify the context providers in the metamodel in order to make clear the separation between these entities and the context mediation concerns. This consideration is relevant because the processing of context should be not impacted by the mechanisms for retrieving the information.

3. ***Context Consumer:*** This entity represents the clients of the context providers. In the REST as a service metamodel there is no entity associated with the `Context Consumer`. However, we include it to express the associations with other concepts of context integration. Furthermore, the consumers of context make necessary the development of solutions dealing with the exchange of information and therefore also motivate the contributions presented in this dissertation.

4. ***Context Information:*** This context element refers to the actual resources that need to be exchanged. The `Context Information` is associated with properties providing additional

descriptions, such as its quality (*i.e.*, the QoC attributes) or type. This information has a URI suffix attribute expressing the local identifier of the context information. This suffix in combination with the URL prefix from the `Connector` element represent the unique identifier associated with the context resource.

5. ***Properties:*** These properties are associated with QoC attributes (cf. Section 3.1) that are useful in the selection of context information. The `Property` element is also used to characterize Context Providers in terms of QoS information [Aurrecoechea *et al.*, 1998, Comuzzi and Pernici, 2009], which can be used for the establishment of Service Level Agreements (SLAs) [Berberova and Bontchev, 2009]. However, the property representation is generic enough for expressing other kinds of properties. In our metamodel, a property has a *nature* indicating if the value of the property can change at execution time—*i.e.*, if the property is static or dynamic. The `Measure` element associated with the property models how it is measured. In this way, this class contains a *type* that represents the property's data type (*e.g.*, long, integer, boolean). This measure also has a `Unit` that refers to the current unit used to measure the property. The units are related to `ConversionFormulas` allowing the conversion between them. The property representation used in the SPACES metamodel simplifies and generalizes concepts from the Amigo-S QoS [Ben Mokhtar *et al.*, 2006] ontology to represent any property (functional or not) of context providers. In particular, we modify the concept of *QoSParameter* introduced by the Amigo-S QoS ontology in order to define simple properties. We preserve the concepts of *Metric* (`Measure`, in our metamodel), *Unit*, and *ConversionFormula* from this ontology.

6. ***Representation:*** In order to provide a flexible solution dealing with devices heterogeneity (cf. Section 5.1), we consider the representations associated with context resources. The `Representation` element reifies these representations. Each representation is associated with a type, such as XML schema or a media type. Unlike the REST metamodel, in the SPACES metamodel we separate the representations from the context information via the `Connector` entity. This separation helps us to keep resources independent from concerns associated with context mediation.

7. ***Connector:*** In our metamodel we also find a `Connector` entity tying the Context Provider and Context Consumer and hiding the mediation concerns from them. This entity encapsulates the *discovery*, *interaction* and *support for different representations* responsibilities. The connectors enable us to reduce the impact on the processing of the context information.

8. ***Protocol:*** This element is the communication mechanism for enabling the information exchange. In context mediation, we consider protocols for interaction and discovery. As already stated in Section 2.4, the discovery protocols can offer the interaction protocol. To express this possibility, we create the *interactionProtocol* association. On the other hand, a protocol can be built on the top of different protocols. For this reason we specify the relationship *buildOn*. For example, the UPnP protocol uses SSDP, HTTP and GENA (cf. Section 2.4.1) as part of its protocol stack.

The presented metamodel states the different elements that should be reified and modularized in a solution providing flexibility in context mediation. In the next section we discuss how we build SPACES connectors by applying the notions considered by our metamodel.

5.4 SPACES Fundamentals

In this section, we define the *SPACES connectors* for dealing with integration in ubiquitous environments and conceived by derived from the metamodel.

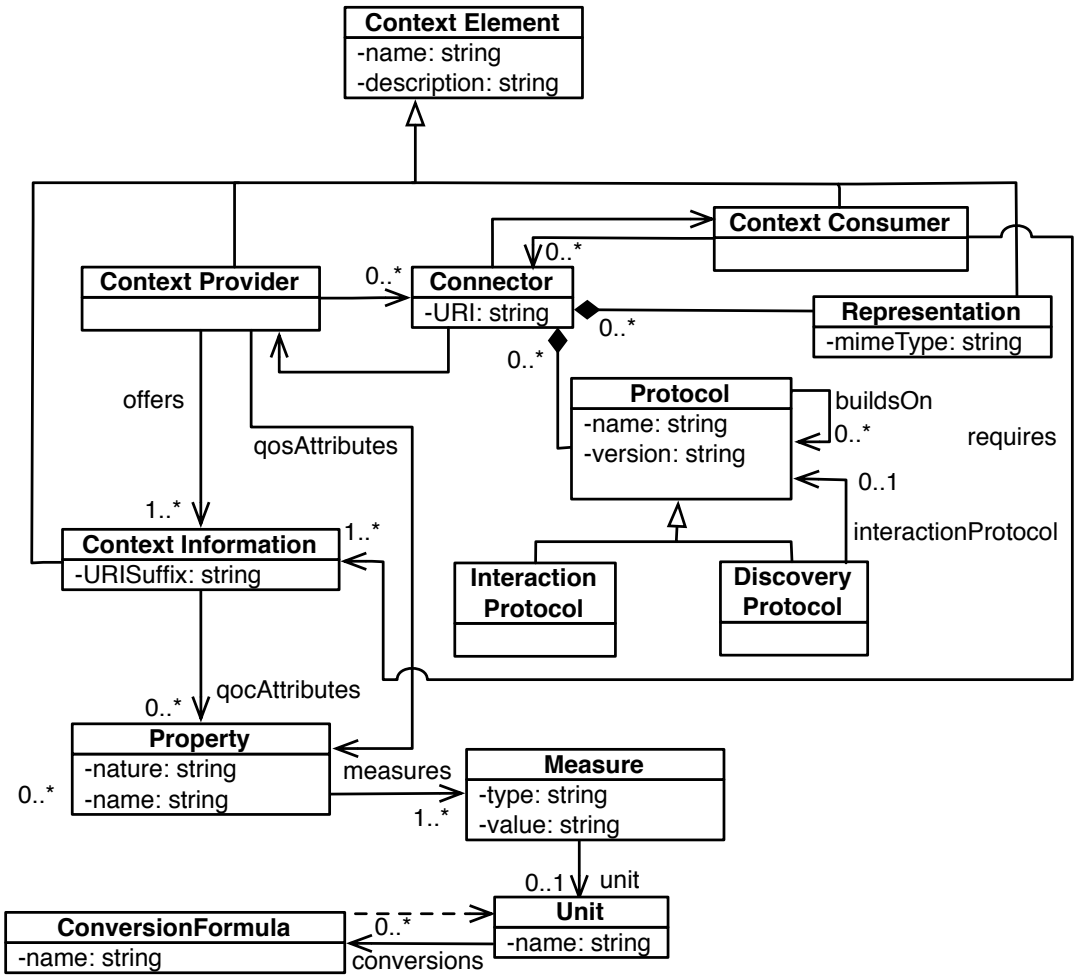


Figure 5.2: Context as a Resource Metamodel

SPACES Connectors

In CBSE, software connectors foster the separation and modularization of concerns. In particular, they encapsulate the transfer of control and data, and non-functional services (*e.g.*, persistency, messaging and invocation) [Taylor *et al.*, 2009, Crnkovic, 2002] helping to keep the application functionality focused on the domain specific concerns. Therefore, we leverage on this concept to support independence of context information and communication mechanisms as well as loose-coupling between producers and consumers (cf. Section 5.1) in our solution.

Following the context as a resource metamodel, the SPACES connectors (also called *ubiquitous connectors* in this dissertation) expose information as resources accessible via different protocols and formats and using logic identifiers. This means that the ubiquitous connectors separate the distribution concerns from the context management tasks but they still keep a clear division of the different responsibilities associated with such distribution. Therefore, by encapsulating the context mediation in these connectors, we do not impact the process of the context information.

Figure 5.3 illustrates the ubiquitous connectors. As it can be seen, these connectors follow a Client-Server style and thus they are composed by two entities:

1. *SPACES Server*: Associated with the context provider, the server exposes the context information as SPACES provided resources accessible via simple protocols. This element also

advertises the resources using different services discovery protocols and accepts subscriptions from interested clients.

2. *SPACES Client*: This entity allows a context consumer to retrieve or send context information. In other words, a SPACES Client produces requests for exchanging information (*i.e.*, the SPACES required resources). Furthermore, this element permits spontaneous communications by discovering the required context resources at runtime and the subscription to sources providing these resources.

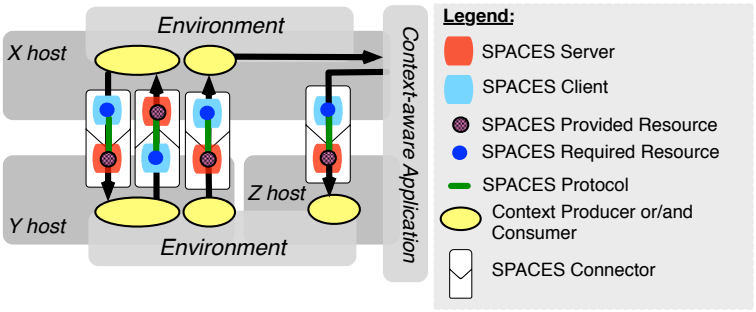


Figure 5.3: SPACES Connectors

In Figure 5.3, we also include the context providers and consumers from the SPACES metamodel. They are context-middleware or context managers (cf. Section 3.1). As observed, by means of our connectors an entity can be provider and consumer at the same time in a transparent way.

Besides providing a clean isolation between context distribution and business logic, SPACES connectors promote multiple implementations of the interaction mechanisms to deal with protocol heterogeneity in ubiquitous environments. For example, the SPACES connectors can support the *message-oriented* [Leclercq *et al.*, 2005] paradigm as well as a *peer-to-peer* [Hu *et al.*, 2007] middleware approach.

Promoting Low Coupling between the Interacting Entities

The modularization of mediation concerns in the SPACES connectors is a first step for uncoupling the interacting entities. We hide the remote interactions from the context processing in order to integrate entities in a transparent way. However, it is not enough. Analyzing the *context as a resource* metamodel presented in Section 5.3, we see that it is still necessary to define how we can use the context resources. In other words, we need to establish how to identify, find, access and represent these resources by means of the SPACES connectors. To do that, we combine the REST architectural style (cf. Section 2.1.2), the existing discovery mechanism (cf. Section 2.4) and the concept of QoC attributes (cf. Section 3.1), all referenced in our metamodel. In particular, we define SPACES nouns, verbs and context types, which are inspired in the triangle of nouns, verbs, and content types from REST, as well as the support for spontaneous communications. Below we discuss the first three issues and postpone the presentation of the discovery issues to Section 5.5.

SPACES Nouns. The exposition of context information as resources via the ubiquitous connectors requires the definition of *context identifiers*. Following the REST principles, in SPACES these identifiers are unique nouns described using the *Uniform Resource Identifier* (URI) format [Berners-Lee *et al.*, 2005]. Therefore, context identifiers include a *communication scheme*, a *server address*, a *context path*, and a sequence of *request parameters*:

```
scheme://context-server/context-path?request-parameters
```

The *communication scheme* describes the communication protocol used by SPACES connectors to transfer the resource representation between the hosting server and the requesting client. Examples of communication protocols include HTTP, FTP (*File Transfer Protocol*), RTSP (*Real Time Streaming Protocol*), and file. Then, the *server address* description is specific to a given scheme. For example, an HTTP server address can be specified using the syntax `user:password@host:port` to describe the web server host and port as well as the credentials for accessing the published resources. The *context path* identifies the context information corresponding to the published resource. Depending on the context provider, the resources can be hierarchically organized as context domains using the syntax `parent-domain/child-domain/context` (*e.g.*, in COSMOS [Conan *et al.*, 2007] or WildCAT [David and Ledoux, 2005]). Finally, the *request parameters* can be used to specify the representation of the requested context information using the MIME type syntax [IANA, 2007] as well the query to retrieve the context information if there is no *context path* or if this functionality is available.

SPACES Verbs. In SPACES connectors, we apply the REST verbs—*i.e.*, GET, PUT, POST, DELETE— to represent the operations typically offered by context providers. The usage of these simple and standard interfaces allows us to focus on the exchange of resources. Below, we present these operations and the associated REST verbs.

1. *Retrieve* (GET): This operation represents the pull mechanism to obtain spontaneously context information from a source. The operation can be synchronous or asynchronous.
2. *Notify* (PUT): The “notify" is used to push automatically context information to interested entities.
3. *Subscribe* (POST): The “subscribe" operation allows clients to show interest in context information and receive it via the notify operation.
4. *Unsubscribe* (DELETE): This operation stops the automatic notifications from context managers.
5. *Publish* (POST): This operation makes available a new context information.
6. *Unpublish* (DELETE): This is the inverse operation of “publish".
7. *Update* (PUT): The “update" operation enables the modification of context information.

As it can be noted, different context operations can be mapped to the same REST verb (*e.g.*, subscribe and publish). However, this is not a problem is SPACES because each operation and the associated resource have a unique and logic URI that avoids any ambiguity.

The following HTTP request shows an example for retrieving the context information *battery level* from a mobile device via ubiquitous connectors:

```
GET /mobile-info/battery_level HTTP/1.1
Host: device.inria.fr:8080
Accept: application/xml, application/json
...
```

Context information can also be pushed into a remote entity by using an HTTP `PUT` request. In this case, the HTTP request is sent to the server-side associated with the context consumer. For example, the HTTP request:

```
PUT /alice-mobile-info/adjust_bitrate HTTP/1.1
Host: server.inria.fr:8080
Content-Length: 1368
Content-Type: application/xml, charset=utf-8

<?xml version="1.0" encoding="utf-8"?>
<spaces:message name="adjust_bitrate"
               xmlns:spaces="http://www.spaces.org/XMLSchema"
```

```
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="www.spaces.org/XMLSchema
                      http://picoforge.int-evry.fr/projects/svn/cosmos/spaces/spaces.xsd">
  <spaces:chunk name="current" type="xsd:int">128</chunk>
  <spaces:message name="qoc">
    <spaces:chunk name="timestamp" type="xsd:int">0904081100</chunk>
    <spaces:chunk name="unity" type="xsd:string">kbps</chunk>
  </spaces:message >
</spaces:message >
...
```

notifies the context consumer that the uploaded XML document refers to a change of the context information *adjust bitrate*.

SPACES Content Types. SPACES connectors supports various types of content for representing the context information. These types are described using the MIME media types classification. The selection of the appropriated format depends mainly on the capabilities of the device that consumes it. In particular, the SPACES connectors implementation reported in this dissertation supports the *Java object serialization* (`application/octet-stream`), XML (`application/xml`) and JavaScript Object Notation-JSON [Crockford, 2006] (`application/json`) documents as resource representations.

The presented connectors provide a simple solution for context integration leveraging on a resource-oriented architectural style. They provide the mechanisms for accessing, addressing and representing the context resources as well as for modularizing the communications. However, because of the context providers and consumers mobility, we also need to consider the establishment of interactions at runtime. In the next section, we discuss this issue in our approach.

5.5 Supporting Spontaneous Communications in SPACES

The SPACES metamodel provides the notions for transparent context handling, which refers to the independence from representations and communication mechanisms (cf. properties for context mediation 1 and 2). By applying a resource-oriented paradigm in our approach, we are also promoting loose-coupling (property 3) between context producers and consumers since we focus the interaction in the required context resources. Furthermore, in the context as resource metamodel we also include discovery protocols and QoC information, which are necessary to satisfy the property for context mediation 3. These elements permit the discovery and selection of context providers at runtime when required. Thus, by including the discovery and QoC notions in the SPACES metamodel, we give the possibility of establishing spontaneous communications [Zhu *et al.*, 2005] to deal with mobility of services and clients in ubiquitous environments.

In SPACES connectors we promote the usage of existing discovery protocols such as UPnP, SLP and Bluetooth SDP (cf. Section 2.4). By using standard protocols, we make the location of providers easier since it is not necessary to develop new and complex discovery protocols. Furthermore, we foster interoperability with legacy services, which can be advertised with standard protocols in the environment. On the other hand, to benefit from the metadata describing the context information (*i.e.*, QoC attributes) and mantain the SPACES connectors flexibility (property 4) in terms of interactions, we consider three design aspects of the SDPs [Zhu *et al.*, 2005]:

1. *Provider invocation*: In general, the process for using located services at runtime has different steps, which include discovery, selection and access of remote providers (cf. Section 3.3). Regarding the access step, some protocols define the underlying communication mechanisms. For example, UPnP states SOAP [Box *et al.*, 2000] in order to invoke operations (*i.e.*, actions in the UPnP vocabulary) on the available services. However, the imposition of a

single communication mechanism in ubiquitous environments, where variability in terms of resources and protocols is the rule rather than the exception, limits the applicability of this kind of protocols. Hence, to face this lack of flexibility, SPACES connectors modify the discovery protocols by making context resources accessible via different interaction mechanisms. In this way, we offer the possibility to choose the most suitable protocols for exchanging in both, the consumer and provider sides. Furthermore, if the discovery protocol, such as SLP or Bluetooth SDP, does not define the communication mechanism, we complement it by using the supported protocols by the connectors.

2. *Description and attribute definition*: A provider description gives information about the type and operations supported by the service. Protocols, such as SLP, provide an additional service characterization by means of attributes. Therefore, in SPACES connectors we benefit from these attributes definition for expressing interaction protocols that can be used to access the provider as well as QoC information. This additional information is used in the provider selection phase.

3. *Provider selection*: In order to select the required service, SDPs apply a basic filter that consider the service type and name. Others protocols offer more specialized searches by defining restrictions on the attributes of the required providers. In ubiquitous connectors, we use these specialized searches to select providers by considering relevant properties for context information consumers—*i.e.*, QoC attributes. If the SDP does not offer the specialized search option, SPACES connectors include an additional filter to ensure that the discovered providers will satisfy the requirements.

Until now we have characterized the SPACES connectors in order to satisfy the properties introduced in Section 5.1. To do that, we have used the different elements and concepts specified by our context as a resource metamodel (cf. Section 5.3). Now, in the next section we materialize the connectors in a generic architecture for context mediation.

5.6 SPACES Connectors Detailled Architecture

In this section we provide a detailed view of SPACES connectors for client (*i.e.*, context consumer) and server (*i.e.*, context provider) sides. The discussed architecture represents the third contribution of the chapter and it is also conceived considering the simplicity and flexibility that are key for dealing with the issues associated with context integration such as heterogeneity and mobility.

Figure 5.4 depicts the abstract architecture of SPACES connectors. Following CBSE and REST principles we have encapsulated the responsibilities of discovery, resource representations and communication in different components. This modularity enables the use of several implementations of the components as well as it provides the flexibility to choose the required functionality in consumers and providers. In particular, the required and provided resources identified by SPACES connectors are reified in the architecture as the SPACES Published Context Request and the SPACES Published Context Resource components, respectively (cf. Figure 5.4).

SPACES Published Context Resource Architecture.

A Published Context Resource is a skeleton receiving and processing context requests sent by remote context consumers (right side in Figure 5.4). This component embeds a Verb Server component isolating the communication protocol used to exchange the context information. This component can have different implementations to support protocols, such as HTTP, FTP and file, and to use REST-based services such as Twitter [Makice, 2009]. The Verb Server also receives the context requests and delegates the processing of the context operation to the respective Handler. The different handlers implement the same interface and are associated with a specific SPACES

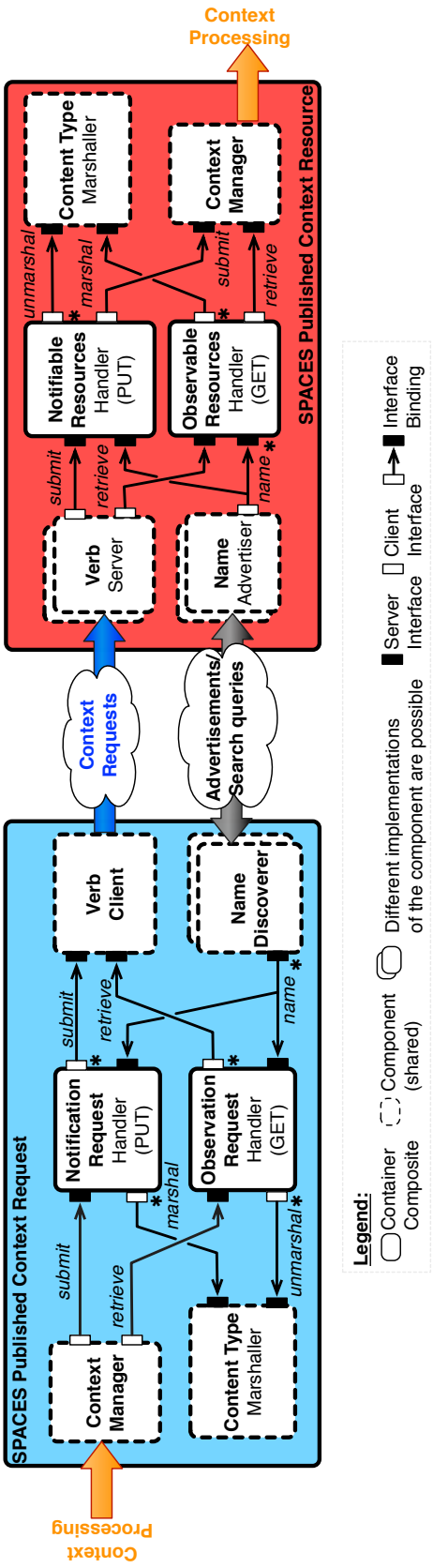


Figure 5.4: SPACES Published Resource-side and Request-side Architectures.

operation. For simplicity and clarity we include only the handlers for dealing with context observation and notification—*i.e.*, the Notifiable Resources Handler and the Observable Resources Handler.

When required, the Handler components use the Content Type Marshaller component to marshal (resp. unmarshal) the requested (resp. posted) context information. SPACES connectors implementation provides a distinct implementation of this component for each of the supported content types—*i.e.*, Java objects, XML, and JSON documents.

The different handlers delegate the context operation to the Context Manager. In the concrete case of the Observable Resources Handler component, it delegates the requested observation to Context Manager and returns the marshaled context information to the Verb Server. Similarly, the Notifiable Resources Handler component unmarshals the submitted context information before notifying the Context Manager. For its part, the Context Manager encapsulates the current context manager being used. This component maps the SPACES operations to the operations of the current context manager.

The Name Advertiser component advertises the available context information via standards service discovery protocols such as UPnP and SLP. This component adds QoC properties describing the context information and the supported access protocols into the advertisement messages.

SPACES Published Context Request Architecture.

A Published Context Request component is a stub generating and sending context requests to a remote context provider (left side in Figure 5.4). In this architecture, the Published Request is associated to a Verb Client component, which acts as a local representative of the resources accessible remotely via a specific communication protocol. The Handlers components are employed when the execution of a remote operation is required. In Figure 5.4 we include the Observation Request Handler and Notification Request Handler components, which enable remote observations and notifications, respectively. In particular, the Observation Request Handler component requests a context observation and unmarshals the received context information via the Content Type Marshaller component. On the other side, the Notification Request Handler component marshals the context information notified by the Context Manager before posting it to a remote resource.

Finally, the Name Discoverer locates the potential context providers in the environment. This component selects the providers regarding the QoC attributes and the supported access protocols.

5.7 Integrating SPACES Connectors into SCA

In the previous section we discussed the generic architecture for the ubiquitous connectors. In this section we complete the third contribution of the chapter by presenting the reification of such architecture by means of SCA [Open SOA, 2007b] (cf. Section 2.2.2).

As already stated, SPACES connectors promote loose-coupling between context managers and consumers by following a simple approach inspired by the context as a resource metamodel. This solution is suitable to enable the integration of context information regarding heterogeneity in terms of communications and representations. Our solution has been defined in a technology independent way, which means that connectors can be implemented with the required programming language. The interoperability of heterogenous implementations is granted because we are promoting the usage of standards interaction mechanism. However, the scope of the approach could be extended to allow the interaction of different kinds of applications, not only context-aware.

To complement our approach and enable the building of distributed entities implemented with different technologies, we extend the SCA standard. In particular, we benefit from the clear concerns isolation that promotes SCA, which is reflected in its independency from communications protocols and implementation technologies. In this way, we group the SPACES connectors

concepts in integration and discovery, which are materialized in SCA by means of two new kinds of bindings: `Resource-Oriented Bindings` and `Ubiquitous Bindings`. In the following sections we present these bindings.

5.7.1 Information Exchange between SCA Services: The case of the Resource-Oriented Bindings

In Section 5.4, we have presented the principles of the SPACES connectors. From this kind of connectors, we derive the `Resource-Oriented Bindings`, which support information exchange via different representations and communication protocols. These bindings therefore apply the SPACES metamodel and provide the same functionality of the ubiquitous connectors except for the discovery.

Service (server-side)

```
<composite name="context-producer.composite">
  <service name="battery-level-info" promote="battery-level-node/battery-level-info"/>
  ...
  <component name="battery-level-node">
    <implementation.java
     class="examples.smarthome.server.src.main.java.BatteryLevelChangeDetectorCO"/>
    <service name="battery-level-info">
      ...
      <resource.xml schema="battery-level.xsd"/>
      <resource.json schema="battery-level.json"/>
      ...
      <binding.http uri="/device_info/battery_level"/>
      ...
    </service>
      ...
  </component>
    ...
</composite>
```

Reference (client-side)

```
<composite name="context-consumer.composite">
  ...
  <reference name="mobile-battery-level-info" promote="mobile-device-resources-
                                              analyzer/mobile-battery-level-info">
  ...
  <component name="mobile-device-resources-analyzer">
    <implementation.java class="examples.smarthome.client.src.
                                main.java.ResourcesAnalyzerCO"/>
    ...
    <reference name="mobile-battery-level-info">
      <resource.xml schema="battery-level.xsd">
      ...
      <binding.http uri="http://device.inria.fr:8080/device_info/battery_level"/>
      ...
    </reference>
  </component>
    ...
</composite>
```

Figure 5.5: Example of an SCA Definition for Resource-oriented Bindings.

Figure 5.5 depicts a resource-oriented binding definition example for services (upper part) and references (lower part). Unlike the SCA typical bindings, the SCA services and references using resource-oriented bindings do not need to expose interfaces. Instead, the services (resp. references) specify the provided (resp. required) resources. To do this, we add a new element, `<resource.representation>`, that expresses the required or provided information including its representation (*e.g.*, XML, JSON) and type (defined by a schema). In the service-side, this element

describes the different formats used to offer the information. For its part, the `<binding.protocol>` element specifies the protocol supported to exchange the information as well as the path to access it (via the `uri` attribute).

In the reference-side, the `<resource.representation>` element defines the required information and the `<binding.protocol>` expresses how the device can retrieve it—*i.e.*, the supported communication protocol. The `uri` attribute in the `<binding.protocol>` element is optional (because the `uri` can be defined at runtime, for example, if discovery protocols are activated) and contains the address for retrieving the information. In this way, by encapsulating the resource retrieval as SCA bindings, we enable the development of applications following a resource-oriented approach and the integration of not only context information, but also of any kind of data (*e.g.*, streams and events).

Implementation of the Resource-Oriented Bindings in the FRASCATI platform

We have integrated our resource-oriented bindings into the FRASCATI platform [Seinturier *et al.*, 2009, Mélisson *et al.*, 2010a] (cf. Section 2.3.3). The FRASCATI selection is motived by two main reasons: *i)* the reflective capabilities that the platform introduces in the SCA programming model to allow dynamic introspection and reconfiguration of SCA based context consumers and producers, and *ii)* we can run the lightweight version of the platform (FRASCAME) on the mobile devices with limited capabilities [Romero *et al.*, 2010c].

We bring the resource-oriented bindings into FRASCATI following the `ContainerComposite` architectural pattern promoted by the HULOTTE platform [Loiret *et al.*, 2009, Loiret *et al.*, 2010b]. This pattern states the definition of a container for a functional component as a composite component. Therefore, we encapsulate the context-aware applications in a container composite hosting the components to intercept the operations (cf. Section 5.4) that require context mediation. In this way, we respect the SPACES connectors isolation of distribution concerns as well as the no impact of the context processing logic.

Figure 5.6 depicts the containers implementing the resource-oriented bindings for context provider and context consumer composites (cf. Figure 5.5). As it can be seen, we follow the SPACES connectors architecture introduced in Section 5.6 for *SPACES Published Context Requests* and *SPACES Published Context Resources*. The producer's container exposes context as REST resources, having the Published Context Resource role in the SPACES architecture. On the other hand, the container associated with the consumer provides the functionality to publish context requests in order to retrieve the required context information. By respecting the ubiquitous connectors architecture in container composites that implement resource-oriented bindings, we modularize the different SPACES principles—*i.e.*, representation, access, and addressing. In this way, we foster the components reuse, the flexibility to choose different component implementations, and the reduction of memory footprint.

5.7.2 Bringing Service Discovery in SCA: The case of Ubiquitous Bindings

The resource-oriented bindings promote loose coupling by enabling resource access via standard and simple operations. This loose-coupling is enhanced with the inclusion of discovery capabilities that permit spontaneous communications at runtime. However, we exclude the SPACES connectors responsibility associated with context discovery from resource-oriented bindings to encourage reuse of both functionalities independently. Therefore, we define `Ubiquitous Bindings` [Romero *et al.*, 2010b], which provide spontaneous communications in a transparent way. This new type of binding integrates state-of-the-art *Service Discovery Protocols* (SDPs) and enables the establishment of communication wires at execution time.

Figure 5.7 depicts the definition of ubiquitous bindings for services (upper part) and references (lower part). The definition of an `Ubiquitous Binding` has a filter attribute in the client-side. This attribute specifies a LDAP filter expressing restrictions of the required service in terms of its properties. For its part, in the server-side, the `Ubiquitous Binding` can have properties that provide additional information about the service, such as QoC attributes. Each property is

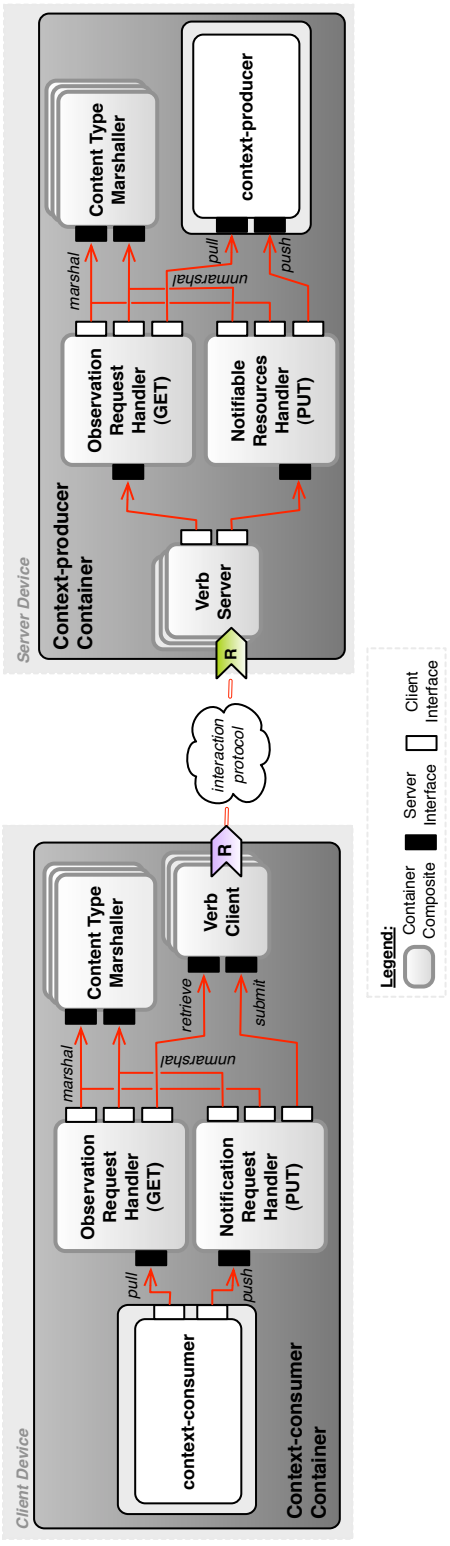


Figure 5.6: Resource-oriented Binding Integration into the FRASCATI Platform

described by a property element. Figure 5.7 shows examples of an `Ubiquitous Binding` with the SLP protocol. The *precision* and *probabilityOfCorrection* are QoC attributes that describe the context provided by the service (cf. Section 3.1). These attributes and the *contextType* property are used in the definition of the LDAP filter in the reference. The bindings in the service side correspond to the different communications that can be used to access the context information. Thus, with the definition of `Ubiquitous Bindings` in this way, we can support spontaneous communications with SCA services via different discovery protocols and then access these services using the most suitable interaction mechanism according to the application needs.

Service (server-side)

```
<composite name="context-producer.composite">
  <service name="battery-level-info" promote="battery-level-node/battery-level-info"/>
  ...
  <component name="battery-level-node">
    <implementation.java
        class="examples.smarthome.server.src.main.java.BatteryLevelChangeDetectorCO"/>
    <service name="battery-level-info">
      ...
      <resource.xml ..."/>
      ...
      <binding.http .../>
      ...
      <binding.slp>
        <property name="probabilityOfCorrection">medium</property>
        <property name="reputation">medium</property>
        <property name="contextType">batteryLevel</property>
      </binding.slp>
    </service>
      ...
  </component>
    ...
</composite>
```

Reference (client-side)

```
<composite name="context-consumer.composite">
  ...
  <reference name="mobile-battery-level-info" promote="mobile-device-resources-
                                                  analyzer/mobile-battery-level-info">
  ...
  <component name="mobile-device-resources-analyzer">
    <implementation.java class="examples.smarthome.client.src.
                                main.java.ResourcesAnalyzerCO"/>
    ...
    <reference name="mobile-battery-level-info">
      <resource.xml ...>
      ...
      <binding.http .../>
      ...
      <binding.slp filter="(&(probabilityOfCorrection=high)(reputation=medium)
                            (contextType=batteryLevel)(protocol=rest))"/>
    </reference>
  </component>
    ...
</composite>
```

Figure 5.7: SCA Definition of the Ubiquitous Bindings.

QoC-based selection

As already stated, in the provider selection we consider additional information about the service. In the particular case of context providers, we filter them regarding the QoC attributes (cf. Section 3.1).

In ubiquitous bindings, these QoC attributes are incorporated in context advertisements to enable the selection of suitable information. Protocols, such as SLP, support the addition of new attributes in a straightforward way and do not need any kind of modification. Other protocols (*e.g.*, UPnP) need to be extended to support the attribute definition. In order to support selection of context providers based on QoC attributes, we use the model introduced in Section 5.3.

Although our QoC model combined with LDAP filters allow selection of context providers according the customer needs, there is no guarantee that only one of the suppliers will satisfy the filter. Therefore, to deal with this issue, we assume that all the discovered context sources of the same information are equivalent—*i.e.*, any of them can be used as context provider. This assumption is valid regarding the limited capabilities of some mobile devices and the importance of the latency time for context-aware systems. In the current implementation of the ubiquitous bindings, we select the first context found and keep the others in a context pool to replace the selected one if it fails. Nevertheless, as we mentioned before, the ubiquitous bindings are flexible enough to allow more complex selection techniques if needed.

Implementation of the Ubiquitous Bindings in the FRASCATI platform

As with resource-oriented bindings, ubiquitous bindings are also integrated into the FRASCATI platform. Figure 5.8 depicts this integration. As it can be seen, we also follow the `containerComposite` pattern used in the resource-oriented binding definition (cf. Section 5.7.1). In the client side, the container provides discovery functionalities—*i.e.*, it is responsible for granting spontaneous access of services discovered at runtime. In other words, the container detects, selects and enables interactions with required services. In the server side, the container complements the discovery capabilities of clients by advertising the services whose bindings are declared as ubiquitous. If the encapsulated component has both roles—*i.e.*, client and server, the container will provide the discovery and advertisement functionalities.

The proposed architecture for these components modularizes different concerns of service discovery (*i.e.*, search, selection, and provider monitoring) and introduces some optional optimizations (in the client side). In this way we foster the reuse of the different components (in particular for selection of providers), the flexibility to use different implementations and choose the required components (not all the components are mandatory). Below, we present the detailed architecture of the container in the reference and service sides.

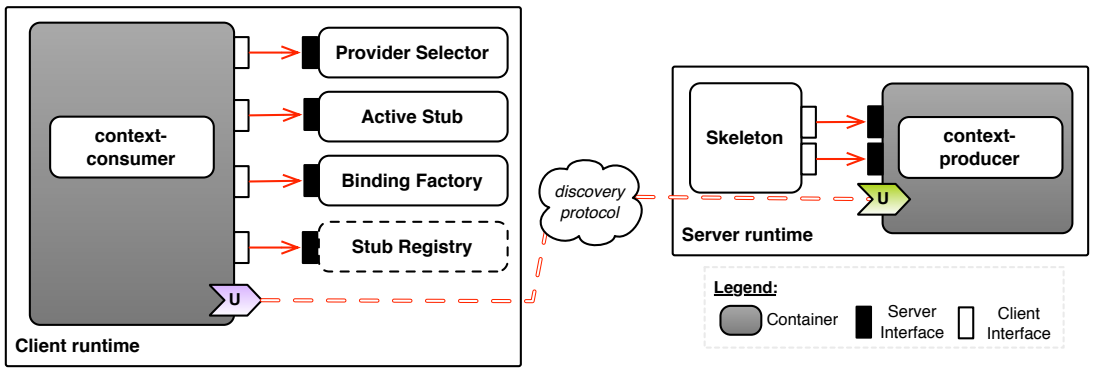


Figure 5.8: Ubiquitous Binding Integration into the FRASCATI Platform.

Container with Discoverer Role. When SCA references in a component are declared with ubiquitous bindings, the container will provide the required functionality for binding establishment at runtime. Benefiting from the different steps in the discovery process (cf. Section 2.4), we modularize and externalize the common functionalities to different container composites to reduce

the memory footprint. In particular, the implementations of different discovery protocols can share components for provider selection, active stubs (that encapsulate the communication with remote services) and the stub registry (which keeps a list of the stubs already instantiated). Thus, the container in the reference-side has a discoverer role in the context of SDPs.

A container with discoverer role (left side in Figure 5.9) has a Discoverer Orchestrator that coordinates the discovery of the requested SCA service. The Finder component sends the requests to detect the potential providers in the environment. If the filters with attributes are supported (*e.g.*, SLP), the Finder translates the LDAP filters to the protocol scheme. If the SDP does not support automatic service selection and it is needed (*e.g.*, UPnP), the Finder uses a provider selector for choosing the service. When the provider is selected, the Discovery Orchestrator disables the Finder and uses the Provider Monitor for monitoring the service availability. When the service is invoked the first time, the Discovery Orchestrator verifies in the stubs registry if there is a stub for the service provider. When this happens, the Discovery Orchestrator selects the registered stub as Active Stub. Otherwise, the Orchestrator uses the FRASCATI binding factory [Romero *et al.*, 2010c] (which is used to create wires and bindings in the platform) in order to instantiate and configure the Active Stub. When the provider becomes unavailable, the Provider Monitor notifies the Discovery Orchestrator that actives again the Finder and asks it to find a new provider.

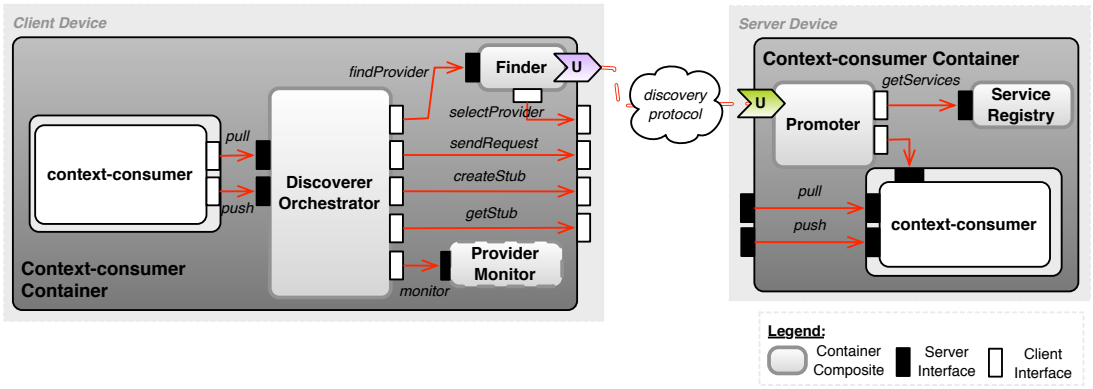


Figure 5.9: Discoverer and Advertiser Architecture.

Container with Advertiser Role. In the service-side, the container of an SCA component exposing services with ubiquitous bindings, has a Promoter component with the following responsibilities:

1. Advertise the available SCA services in the Service Registry. Each entry in the Service Registry contains the required information for the published service (*e.g.*, name, type) that is required to advertise the service. This information can be updated at runtime.

2. Listen and process discovery requests.

3. Notify events associated with the SCA component state.

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

5.8 Summary

In this chapter we introduce our solution to deal with context integration in ubiquitous environments. In our approach, we consider the importance of *independence from context representations* and *communication mechanisms*, *loose-coupling*, *flexibility* and *extensibility* (called properties for context mediation in this dissertation) to deal with heterogeneity in this kind of environments. We reach these properties by basing our solution in well-defined standards characterized by their simplicity and wide acceptance, promoting the separation of mediation concerns, applying a resource-oriented approach and, using principles and concepts from CBSE and SOA.

The three contribution that make part of our approach, which have been presented in this chapter, can be summarized as follows:

1. **SPACES Metamodel**: As the cornerstone of our approach we define a metamodel providing the elements for exposing context information as resources. In particular, in the metamodel we state the relationships between the principal concepts that have to be considered in context mediation. The SPACES Metamodel have two kinds of concepts: *i)* Context Elements that reify relevant entities in context mediation and, *ii)* Complementary elements that we consider important for defining a simple but complete solution. Thus, our SPACES Metamodel, inspired in the REST architectural style, provides the foundations to build solutions enabling the integration of context information considering the different participants and concerns as first class entities.

2. **SPACES Connectors**: From the previous metamodel, we propose SPACES connectors, which constitute our middleware solution for dealing with context integration. These connectors foster the separation of context mediation concerns and context processing tasks by applying concepts from CBSE and the REST architectural style.

3. **Generic Architecture for Context Mediation**: By following the principles stated by the SPACES Connectors, we introduce a generic architecture respecting the modularity from the context as a resource metamodel for context processing—*i.e.*, the discovery, addressing, access and representations. Then, we reified this architecture by extending the SCA standard. The combination of SPACES Connectors and SCA promotes the usage of our approach in the integration of different kinds of applications (not only context-aware). To achieve it, we classify the ubiquitous connectors functionality in two categories: *integration* and *discovery*, which are materialized in SCA as *resource-oriented bindings* and *ubiquitous bindings*, respectively. Both, this separation and the clean concerns isolation from SCA allow the combined or independent usage of these functionalities. Thus, by putting together SCA and our SPACES connectors, we provide support for mobility, heterogeneity and, context flow in context-based adaptation in ubiquitous environments.

In the next chapter we present ubiquitous feedback control loops, which illustrate a concrete application of SPACES connectors concepts and allow the building of adaptable context-aware applications.

Chapter 6

Building Ubiquitous Feedback Control Loops

Contents

In this dissertation we identify two main issues associated with the development of ubiquitous systems: context integration and context-based adaptation. In this chapter we focus in the second part of the contribution, which focuses on the adaptation issue.

Motivations

The context-aware applications have be adapted with a minimal intervention from users. The different adaptation tasks—i.e., the *context gathering*, the *context processing*, the *identification of required configurations* and the *execution of the adaptation* have to be as transparent as possible to the application users. As already stated in Chapter 4, Autonomic Computing provides the foundations for developing self-management systems. In this chapter we exploit such foundations for building Feedback Control Loops that enables transparent adaptation of context-aware applications and the same time helps developers un the development of such applications.

Chapter Contributions

In Chapter 5 we have discussed the SPACES Connectors architecture and its incorporation into SCA for dealing with context integration. Now, we need to employ it for making the adaptation of applications easier. To do that, we consider this kind of adaptation as a process where the integration of the context information is fundamental. This process starts with the data collection from context sources and finishes in the execution of the required configuration of applications. Throughout this process, context information is always flowing between the participants. SPACES Connectors can be applied to enable this flowing. Hence, the contributions of this chapter can be summarized as follows:

1. ***A general approach for dealing with the adaptation process***: Following the principles of the autonomic computing (cf. Chapter 4) and exploiting SPACES connectors, we define *Ubiquitous Feedback Control Loops* (Ubiquitous FCL), our approach supporting the adaptation of applications based on context information. This approach allows the mobility of adaptive applications as well as the dynamic incorporation of new services in the adaptation process. In the Ubiquitous FCLs, we include the notion of Local Feedback Control Loops, which support simple local decisions when the global (or ubiquitous) loop cannot make the adaptation decisions. Fostering simplicity and reuse (cf. Section 1.2), the different participants in FCLs are interconnected by respecting the SPACES Connectors principles.

2. ***A mechanism for determining the required adaptions***: By applying constraint programming techniques [Apt, 2003], we define a mechanism for determining the required configurations in Ubiquitous FCLs. The required reconfiguration is stated by regarding the context information as well as additional dimensions such as resource consumptions, Quality of Service (QoS) and adaptation cost. Therefore, the new application configuration is not only valid but also the most suitable for the current conditions and expectations of the costumer.

Chapter Organization

The remainder of this chapter is organized as follows. We start by presenting the properties expected in context-based adaptation (cf. Section 6.1) as well as our modeling of this kind of adaptation as a process (cf. Section 6.2). Then, we discuss the Ubiquitous FCL, our general approach for context-awareness (cf. Section 6.3). We continue providing details about how we deal with some parts of the Feedback Control Loop. In particular, we describe our proposal for determining the required configurations (cf. Section 6.4) and we give an overview of the mechanism to plan the actions for reaching such configurations (cf. Section 6.5). In a similar way, in Section 6.6 we explain the orchestration of the adaptation execution. Besides that, in Section 6.7 we complement the Ubiquitous Feedback Control Loops description by introducing the notion of Local Feedback Control Loops. Finally in Section 6.8 we provide a summary of the chapter.

6.1 Properties for Context-Based Adaptation

In ubiquitous environments, the variability in terms of computational resources and technologies makes the adaptation of context-aware applications a real challenge. Depending on devices capacities and required context information some adaptation decisions can be made locally. However, the adaptation decision can be complex—*i.e.*, information from several sources using different kinds of discovery protocols, interaction mechanisms and representations has to be retrieved and processed. Moreover, the available services in the environment must be considered in order to provide a better user experience. To deal with these issues, we can benefit from the most powerful devices for assigning the responsible entities of the different adaptation tasks. Therefore, to distribute the adaptation responsibilities we need to consider the following properties:

1. *Flexible context integration*: Context providers differ in their capabilities and therefore in the mechanisms used to advertise and access the information. Normally, providers select the

most convenient protocols and context representations according to their computational resources, such as available memory and battery level. Then, a solution enabling context-based adaptation should be flexible enough to deal with the integration of context information regarding this heterogeneity.

2. *Clear definition of the different adaptation roles and tasks*: The identification of different tasks and entities responsible in the adaptation process has several advantages. First, it is possible to establish who must do what. A well defined task attribution is key for the successful execution of a system. Second, the modularization of the different responsibilities fosters the improvement and evolution of adaptation functionalities. And third, this same modularization makes the detection of problems easier. Therefore, the proposed mechanism should enable a clear definition of the different tasks.

3. *Dynamic Detection of Adaptive Applications and Services*: The distribution of the adaptation concerns and the dynamism from ubiquitous environments mean that some of the process participants can join and leave at anytime. In particular, the adaptive applications running on mobile devices and some services satisfy this property. Hence, the solution supporting the adaptation should provide support for dynamic discovery.

4. *Suitable mechanism for identification of adaptation situations*: Once all the context information is collected, the approach enabling the adaptation has to provide a way for identifying the cases requiring the application adaptation as well as the determination of required actions to do it. The underlying mechanism has to consider, of course, the context information. Moreover, the adaptation should be optimal regarding additional dimensions that improve the user satisfaction. Therefore, the result of the adaptation has to produce a working application configuration that satisfies the user expectations.

5. *Simple execution mechanism of the required reconfigurations*: The offered approach should allow the execution of the adaptation respecting the flexibility principle for context integration. To do that, the reconfigurations can be considered as context information and therefore a similar mechanism for dealing with heterogeneity issues should be applied.

As stated in Section 4.4, the main drawbacks of approaches dealing with context-based adaptation are associated with the *lack of flexibility* for supporting reconfiguration at the application and runtime levels, *the low priority given to the information integration*, *the lack of relevance given to context information* and *the high dependency* between the different entities participating in the adaptation process. In order to face these drawbacks and satisfy the mentioned adaptation properties, we propose the concept of *Ubiquitous Feedback Control Loops*. These Ubiquitous loops enable the adaptation of context-aware applications in an semi-autonomic way. To do that, we leverage on the combination of SPACES Connectors and SCA presented in Chapter 5. Table 6.1 compares our Ubiquitous FCLs with different autonomic approaches in terms of context mediation properties and adaptation issues. In this table we also include the ubiquitous and service discovery approaches, because with the Ubiquitous FCLs we complete the contribution of this dissertation. As observed, the FCLs inherit the different advantages from SPACES Connectors. In the following sections, we provide more detail about these loops and how we meet the properties for adaptation.

6.2 Context-based Adaptation as a Process

In order to meet the property *"Clear definition of the different adaptation participants and tasks"* (cf. Section 6.1) we employ the MAPE-K model (cf. Section 4.1) as well as the notion of business process [Coalition, 1999, Coalition, 1999]. In particular, regarding the definition of process, we can see that the realization of context-based adaptation requires the execution of a series of tasks, in a specific order to meet the goal of changing the structure or/and behavior of applications according to the environment state. Furthermore, the distributed nature of the adaptation

	Dimension / Middleware Approach	Context Information		Loose Coupling		Independent Communication Mechanism		Adaptation		
		Support for Multiple Representations	QoC	Mobility Support	Dependency Definition	Interaction	Discovery	Static	Dynamic	Level
Autonomic Approaches	Agent-based Middleware for Context-Aware Services	✗	✗	✗	Component Interfaces	✗	✗	✗	✓	Middleware
	ANS	✗	✓	✗	QoC	✓	✓	✓	✓	Middleware
	Adaptation Platform for Autonomic Context-Aware Services	✗	✗	N/S	Component Interfaces	N/S	N/S	✗	✓	Application, Middleware
	Framework for Autonomic Context-Aware Service Composition	✗	✗	✗	Component Interfaces	N/S	✓	✗	✓	Application
	JADE	N/A	N/A	N/A	N/A	✗	N/A	✓	✓	Application
	MIDAS	✗	✗	✓	Proxies (Context elements)	✓	N/S	✓	✓	Application, Middleware
	AutoHome	✗	✗	✗	Client and Server Interfaces	✓	✓	✓	✓	Application, Middleware
	MUSIC	✗	✗	✓	Component Interfaces, Service Descriptions	✗	✓	✓	✓	Application, Middleware
	Rainbow	N/A	N/A	✗	System-Layer Infrastructure	✗	✗	✓	✓	Application (dynamic), Middleware (static)
	DACAR	N/A	N/A	✗	Component Interfaces	✗	✗	✗	✓	Application
Service Discovery Solutions	INDISS	✗	✗	N/A	Events	✗	✓	✓	✗	Middleware
	ReMMoC	✗	✗	N/A	WSDL	✓	✓	✓	✓	Middleware
	A multi-protocol framework for ad-hoc service discovery	✗	✗	N/A	Component Interfaces	✗	✓	✓	✗	Middleware
Middleware Solutions for Context-Awareness	GAIA	✗	✗	✓	Channels	✗	✗	✓	✓	Application, Middleware
	GAIA Microserver	✗	✗	N/A	J2ME Proxy/ Channels	✗	✗	✗	✓	Application
	MUSIC Peer-to-Peer	✗	✗	✓	Peer Group	✗	✗	✗	✗	N/A
	Cortex	✗	✗	✗	Sentient Objects	✗	✓	✓	✓	Middleware
	CARMEN	✗	✗	✗	Proxies (MAs)	✗	✗	✗	✓	Middleware
	Aura	✗	✗	✗	Environment Manager	✗	✗	✗	✓	Middleware
	CARISMA	✗	✗	✗	Profiles	✗	N/A	✗	✓	Middleware
	Cooltown	✗	✗	✓	Directory Module	✗	✓	✓	✗	Middleware
	MiddleWhere	✗	✓	✗	Adapters	✓	N/A	✗	✗	N/A
	MobiPADS	✗	✗	✗	Channel Service, Mobilets	✗	N/A	✓	✓	Application, Middleware
	SOCAM	✗	✗	✗	Service Locating Service	✗	✗	✗	✓	Application
	RCSM	✗	✗	✗	Channels (CTCs)	✓	✓	✓	✓	Application
	Large Scale Peer-to-Peer Context Dissemination Middleware	✗	✓	✗	Context-based groups	✗	✗	✗	✗	N/A
	CAPNET	✗	✗	✗	Messaging Component	✓	✗	✓	✗	Middleware
	Ubiquitous Feedback Control Loops	✓	✓	✓	Data	✓	✓	✓	✓	Application, Middleware

Table 6.1: Comparation of Ubiquitous FCLs with Different Approaches for Adaptation

and the consequent modularization of the adaptation responsibilities (*i.e.*, monitoring, analysis and execution) bring into play different participants as well as the roles that they need to hold. Therefore, to build our solution, we start by modeling this kind of adaptation as a process.

Figure 6.1 depicts this process following the BPMN notation [White, 2004]. As it can be seen, we identify four main roles: *Information Source*, *Context Provider*, *Adaptation Orchestrator* and *Application Client*. The *Information Source* role is held by entities in the environment providing relevant information for the adaptation process (cf. Section 3.1). Mobile devices and sensors are examples of entities having this role. For its part, the *Context Provider* role has responsibilities associated with the collection of data from the different sources, the processing of this data and the production of context information that will enable the *Adaptation Orchestrator* for determining the required configurations on the *Client Application*. Depending on how the adaptation is tackled, the roles of *Context Provider* and *Adaptation Orchestrator* can be held by the same entity. Consequently, the functionality for executing the adaptation can be distributed between different entities.

Figure 6.1 also shows the general tasks that make part of the adaptation process. These tasks are discussed below:

1. *Data Gathering*: Consists in the collection of raw data from information sources. Once the data is gathered, it is determined if there is a significant change in order to trigger the adaptation. The Data Gathering belongs to ***monitoring*** phase of the MAPE-K model.

2. *Context Processing*: In this task, the data collected in the *Data* or *Context Gathering* tasks is processed in order to produce high level information that will be used to decide the required reconfigurations. The context-based adaptation process can include multiple information sources and context providers, which means that the *Data Gathering* and *Context Processing* tasks can be execute several times before reaching the *Context Gathering* task. In the `Adaptation Orchestrator` the *Context Processing* task is optional.

3. *Context Gathering*: This task consists in the retrieval of the context information for identifying adaptation situations. The information can be explicitly requested from the context provider (*pull* mechanism) or the context consumer can be notified (*push* mechanism).

4. *Identification of the New Configuration*: Using the collected context information, this task identifies the new configuration that is required in the adaptive application. The *Context Processing*, *Context Gathering* and *Identification of the New Configuration* tasks constitute the ***analysis*** phase.

5. *Determination of the required actions to meet the configuration*: Once the required configuration is established, it is necessary to determine the changes to be done on the application in order to meet this new configuration. In other words, we need to decide what components must be added, deleted and/or replaced. The result of this task will be reconfiguration scripts. This task is part of the ***planning*** phase of the MAPE-K model.

6. *Adaptation Information Gathering*: This task allows the collection of reconfiguration scripts. The task makes part of the ***execution*** phase together with the *Adaptation Execution* task.

7. *Adaptation Execution*: This task is the final task of the adaptation process. The task consists in the reconfiguration of the context-aware application, *i.e.*, the execution of the reconfiguration script.

As in any process, in the context-based adaptation process the flow of information between the tasks is vital for its execution. In particular, in the context-based adaptation process we find two types of information: *context information* and *information associated with the adaptation*. As already stated in Section 3.1, context information is data characterizing the situation of a person, place or object, which are considered relevant in the interaction between users and applications [Dey, 2001]. Examples of context information include location, user preferences, and data

describing the environment state (*e.g.*, temperature, sound level and available bandwidth). On the other hand, the information associated with the adaptation is any information derived directly or indirectly from the context information. This information includes adaptation rules, reconfiguration scripts and the functionality that has to be deployed.

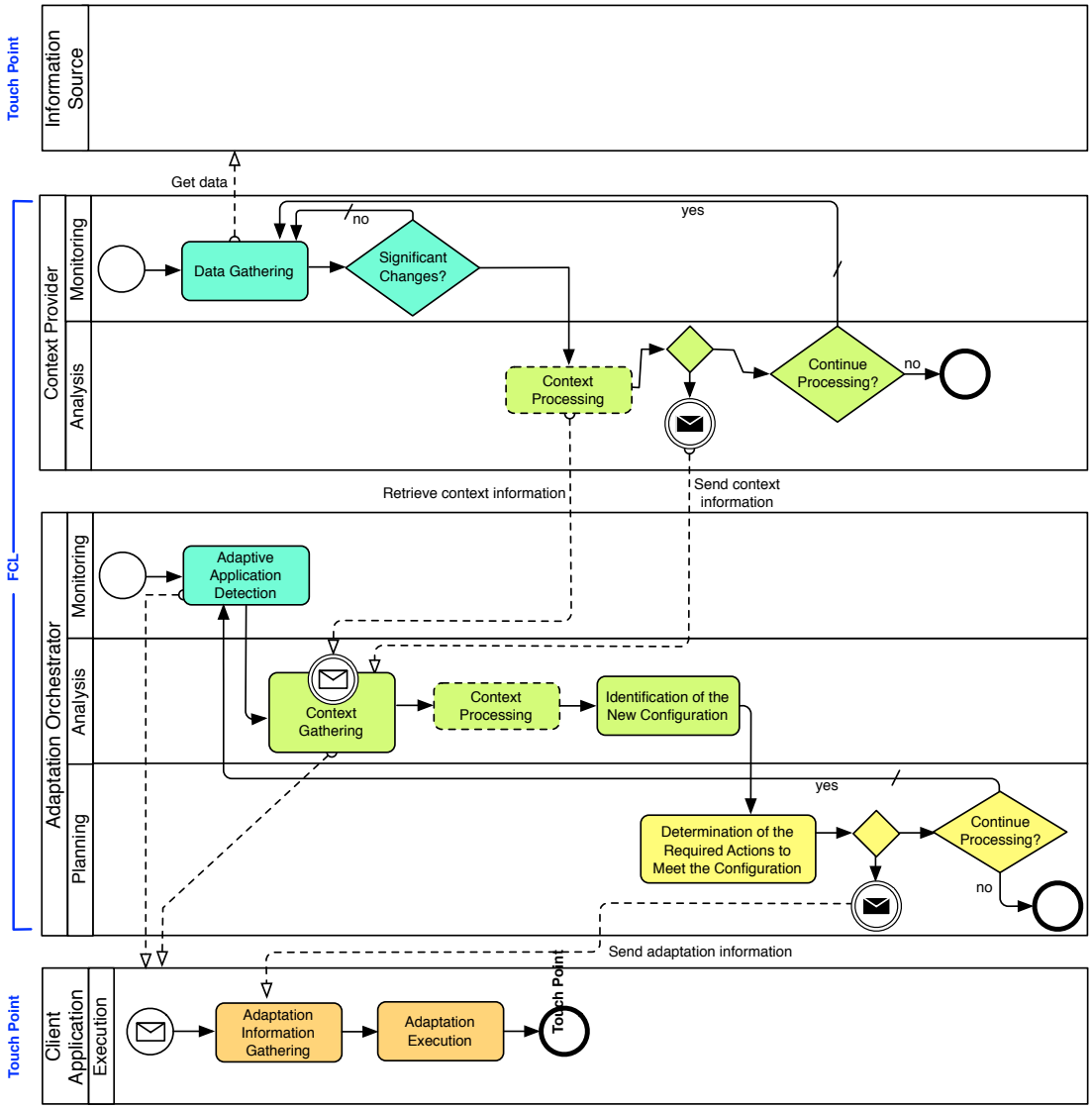


Figure 6.1: Context-Aware Adaptation Process Definition

6.3 Building Ubiquitous Feedback Control Loops

The model introduced in the previous section gives us the elements for defining Ubiquitous Feedback Control Loops. In particular, considering the different roles and responsibilities in the context-based adaptation process and the need for distributing those responsibilities between several entities, we need a mechanism enabling the flow of information. This mechanism should consider the heterogeneity at different levels in ubiquitous environments. Then, SPACES Connectors, our approach discussed in Chapter 5, is a suitable option to deal with this issue. Further-

more, to benefit from the variability in terms of services and support the mobility of applications, our Ubiquitous Feedback Control Loops exploit the discovery capabilities present in Ubiquitous Connectors. Therefore, this kind of control loop respects the *flexible context integration*, *clear definition of the different adaptation roles and tasks* and *dynamic adaptation of adaptive applications and services* properties discussed in Section 6.1.

In order to build the Ubiquitous Feedback Control Loops we also use the SCA component model (cf. Section 2.2.2). This means that the applications that we aim to adapt are conceived as SCA applications. As already stated, the SCA selection is motivated because it structures SOA applications keeping the advantages of this approach in terms of loose-coupling and reuse. Furthermore, as we will discuss in Section 6.4, by using a unified model approach, such as SCA, we provide support for adaptation at platform and application levels. The SCA usage also fosters the incorporation of the RESOURCE-ORIENTED BINDINGS (RBs) and the UBIQUITOUS BINDINGS (UBs) (cf. Sections 5.7.1 and 5.7.2, respectively), which bring discovery capabilities and a data-centric approach into SCA. In particular, these bindings support the notion of Context as a Resource (cf. Section 5.3). Now, given that the Software as a Service (SaaS) [Association, 2001, Nitu, 2009] becomes a reality and the widespread use of SOA, we benefit from our UBs and RBs to connect the different entities in the Ubiquitous FCLs. This means that we enable dynamic discovery of adaptive applications and services using standard protocols, such as UPnP and SLP, and access via the most suitable interaction protocols. This approach makes the integration of legacy applications as well as different kinds of service possible.

As already stated, our vision of the adaptation as a process helps us to conceive our Ubiquitous FCLs. In particular, we benefit from this vision in three aspects:

1. *Responsibilities Distribution*: In Section 6.2, we have identified the different roles considered relevant in the adaptation process as well as their associated responsibilities. This role definition together with the tasks modularization help us to decide who must do what in the Ubiquitous FCLs.

2. *Architecture Definition*: The clear definition of responsibilities allows us to conceive a simple architecture reifying the different phases of the control loop. This architecture modularizes the adaptation responsibilities fostering reuse and customization of the different elements.

3. *Usage of the SPACES Connectors Concepts*: The identification of the data that must flow through the adaptation participants makes the usage of concepts from SPACES Connectors easier. In particular, we can determine where the UBs have to be employed for enabling spontaneous interactions.

To illustrate our approach, we employ the generic example depicted in Figure 6.2. In this figure we use the SCA graphical notation introduced in Section 2.2.2. In particular, we have included three entities and several services that are available in the environment to show the distribution of the different responsibilities and roles of the adaptation process depicted in Figure 6.1. The `Entity 1` has two roles: *Decision Maker* and *Context Provider*. This means that the `Adaptive Server` contained by this entity has responsibilities associated with monitoring, analysis and planning phases. The monitoring responsibilities include the detection of the available context information, services and adaptive applications. This data is used by the `Adaptive Server` in order to determine the new application configuration for better exploiting the available services in the environment. More details about this analysis activity is provided in Section 6.4. Once the new configurations and the actions to reach them are defined, they are executed on the `Entity 2` and `Entity 3`, which hold the role of *Client Application* from Figure 6.1. The two entities provide the required services for fulfilling the planned actions on context-aware applications. The `Entity 2` contains an adaptive application and the `Entity 3` hosts an adaptive service. Additionally, `Entities 2` and `3` are *Information Sources* (cf. Figure 6.1) providing data about the application configuration.

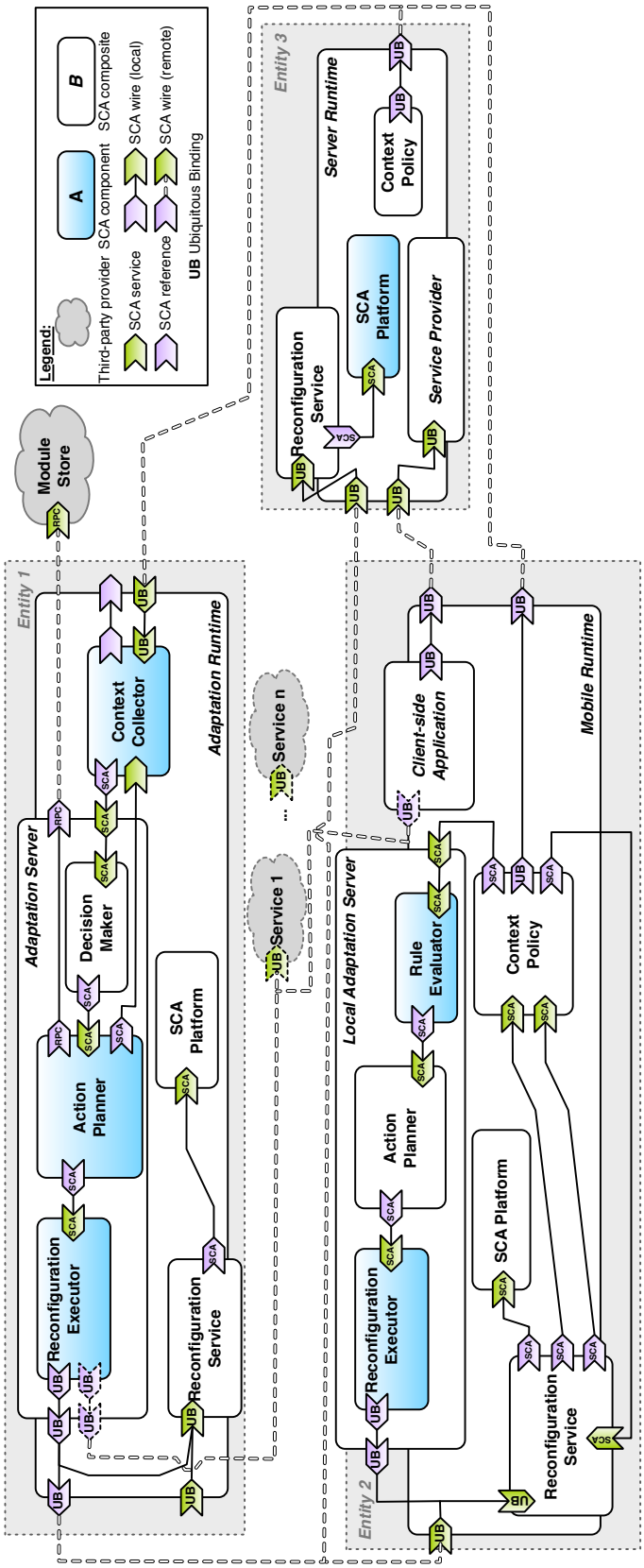


Figure 6.2: Example of a Ubiquitous Feedback Control Loop

6.4 Determining the Required Reconfiguration for the Applications

In the previous section we gave an overview of our Ubiquitous FCLs and the different elements that compose them. In this section, we focus on the analysis phase of the FCL, which is associated with the *Decision Maker* from Figure 6.2. In particular, we propose a mechanism based on Constraint Satisfaction Problems (CSPs) techniques [Apt, 2003]. This mechanism selects a new valid configuration regarding, for example, the cost associated with resource consumption (*e.g.*, memory or energy), the adaptation (*e.g.*, in terms of bindings that we need to add or remove) or QoS [Xiao, 2008] (*e.g.*, user satisfaction or response time). In this way, we provide adaptation considering not only the current context but also dimensions for providing an optimized application that guarantees a better user experience.

Our mechanism for selecting the most suitable configuration, inspired on [Beauvois *et al.*, 2007, Neema and Ledeczi, 2003], assumes that an application provides a set of functionalities, each of which is reified by one or several components. Some of these functionalities are mandatory, *i.e.*, they have to be always present in the application and therefore the components that implement them represent the *application kernel*. The optional functionalities are the *flexibility points* (or *variation points*) of the architecture. We exploit these variations points in order to determine the functionalities that have to be added or modified according to the context changes.

In order to make the decision related with the new configuration, we also require some information provided by entities holding the *Client Application* and *Decision Maker* roles. The former has to keep the list of flexibility points associated with the current application configuration. This information is deployed with the application and updated each time that it is reconfigured. The latter has the list of mandatory components that define the application kernel and the different component configurations associated with each flexibility point. Furthermore, the entity holding the *Decision Maker* also includes the list of dependencies between the flexibility points. These dependencies define *exclude* and *require* relationships.

In our mechanism, we associate with each adaptation situation a context policy that identifies a concrete need for changing a flexibility point or functionality in the application. The results of different context policies are aggregated for defining the new required configuration—*i.e.*, the variation points that need to be modified. In Ubiquitous FCLs, several policies can be associated with the same functionality. Such policies can be triggered at the same time by context changes and can select different implementations of the point. In these cases, we need to apply our mechanism in order to decide the final new configuration of the application. Thus, by modularizing the changes of each flexibility point in context policies we simplify the selection of the new configuration.

In the next section, we present a simple application example that we use for illustrating the usage of our selection mechanism.

6.4.1 Example: The MOBIHOME Application

MOBIHOME is a simple application that enables mobile devices for controlling and accessing different services in home environments. The application has a module for each service. The different modules can be added, removed or replaced depending on the service availability. Therefore, they represent the flexibility points of the application architecture.

Figure 6.3 depicts the architecture of the application. As observed, MOBIHOME includes the `View` and `Controller` components, which represent the kernel of the application. The architecture also presents two modules: `TV Control Module` and `Multimedia Module`. The former allows the mobile device to control a UPnP television via our ubiquitous bindings (cf. Section 5.7.2). This module only has a possible implementation. The latter accesses a multimedia server available in home. This component has an implementation allowing the display of multimedia content such as videos and photos (called MultimediaI1). This implementation exploits

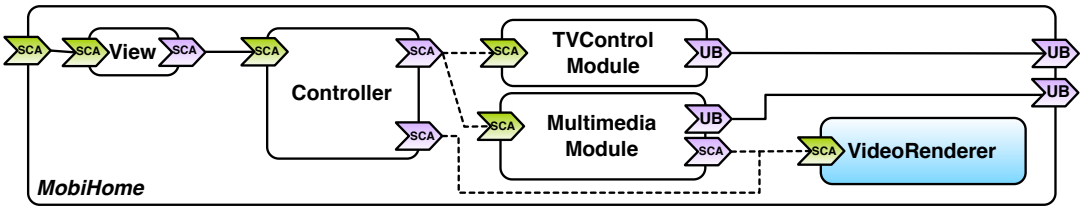


Figure 6.3: The MOBIHOME Application

the `Video Renderer` component for video processing. Other implementation of the component only permits the exploring of the multimedia files (called MultimediaI2).

In the following sections we use this example to present the modeling of our mechanism for selecting the most suitable configuration. In this model we consider the already mentioned assumptions regarding the application architecture as well as the different dimensions (resource cost, adaptation cost and QoS) for optimizing the configuration selection.

6.4.2 Modeling the Selection Problem

In this section we present the modeling that will enable the selection of the optimized configuration. To do that, we define $\mathcal{C} = \{C_1, \cdots, C_m\}$ as the set of configurations that can be stated from the different implementations of the flexibility points selected with the context policies. For example, in MOBIHOME we have the following configurations:

- $C_1 = \{ViewI, ControllerI\}$
- $C_2 = \{ViewI, ControllerI, TVControlModuleI\}$
- $C_3 = \{ViewI, ControllerI, VideoRendererI\}$
- $C_4 = \{ViewI, ControllerI, TVControlModuleI, VideoRendererI\}$
- $C_5 = \{ViewI, ControllerI, MultimediaI1, VideoRendererI\}$
- $C_6 = \{ViewI, ControllerI, MultimediaI2, VideoRendererI\}$
- $C_7 = \{ViewI, ControllerI, TVControlModuleI, MultimediaI1, VideoRendererI\}$
- $C_8 = \{ViewI, ControllerI, TVControlModuleI, MultimediaI2, VideoRendererI\}$

On the other hand, $\mathcal{FP} = \{FP_1, \cdots, FP_n\}$ denotes the flexibility points of the application and $\mathcal{I}_j = \{I_1, \cdots, I_o\}$ refers to the implementations associated with the FP_j flexibility point. In the context of our example we have two flexibility points associated with the two modules. For the `TV Control Module` and the `Video Renderer`, we have one implementation and for the `Multimedia Module` we have *MultimediaI1* and *MultimediaI2*. We also introduce FPC_i as a subsets of FP representing the selected variation points for the i^{th} configuration. For example $FPC_8 = \{TVControlModuleFP, MultimediaModuleFP, VideoRendererFP\}$ corresponds to the selected points in configuration 8 of MOBIHOME. The expressions in the set $SFP_i = \{sfp(FP_1), \cdots, sfp_(FP_n)\}$ indicates if the i^{th} flexibility point is used (1) or not (0) in C_i. For C_8 in MOBIHOME, $sfp(TVControlModuleFP) = 1$, $sfp(MultimediaModuleFP) = 1$ and $sfp(VideoRendererFP) = 1$. For its parts, IC_i^j represents the group of implementations that can be potentially chosen for the j^{th} flexibility point in the i^{th} configuration. The expressions in $\mathcal{SI}_j^i = \{si(I_1), \cdots, si(I_q)\}$ express the selected and discards implementations of the j^{th} flexibility point in configuration C_i.

To state *require* and *exclude* constrains between flexibility points, we define the sets $\mathcal{R} = \{(FP_i, FP_j) \in \mathcal{FP} \times \mathcal{FP} : FP_i \ requires \ FP_j\}$ and $\mathcal{E} = \{(FP_i, FP_j) \in$

$\mathcal{FP} \times \mathcal{FP} : FP_i$ *excludes* $FP_j\}$, respectively. In the MOBIHOME example, $\mathcal{R} = \{(MultimediaModuleFP, VideoRendererFP)\}$. The only element in this set indicates that the usage of the *MultimediaModuleFP* point in the configuration always needs the presence of one implementation associated with the *VideoRendererFP* point. On the contrary, there are not exclude relationships and therefore $\mathcal{E} = \emptyset$. Table 6.2 summarizes these definitions.

Definition	Explanation
(d1) $\mathcal{C} = \{C_1,..., C_m\},\ 1 \leq m$	Possible Configurations
(d2) $\mathcal{FP} = \{FP_1,..., FP_n\},\ 1 \leq n$	Flexibility Points
(d3) $\mathcal{I}_j = \{I_1,..., I_q\},\ 1 \leq q$	Implementations of the j^{th} variation point
(d4) $\mathcal{FPC}_i \subseteq \mathcal{FP}$	Selected flexibility points in the i^{th} configuration
(d5) $\mathcal{SFP}_i = \{sfp(FP_1),..., sfp(FP_n)\},\ 1 \leq n \wedge (sfp(FP_j) = 1 \Rightarrow FP_j \in \mathcal{FPC}_i) \wedge (sfp(FP_j) = 0 \Rightarrow FP_j \notin \mathcal{FPC}_i)$	Expressions indicating the selected and excluded variation points for the i^{th} configuration
(d6) $\mathcal{IC}_i \subseteq \mathcal{I}_j$	Selected implementations of the j^{th} variation point for the i^{th} configuration
(d7) $(d7)\ \mathcal{SI}_i^j = \{si(I_1),..., si(I_q)\},\ 1 \leq q \wedge (si(I_j) = 1 \Rightarrow I_j \in \mathcal{IC}_i) \wedge (si(I_j) = 0 \Rightarrow I_j \notin \mathcal{IC}_i)$	Expressions indicating the selected and excluded implementations of the j^{th} flexibility point for the i^{th} configuration
(d8) $\mathcal{R} = \{(FP_i, FP_j) \in \mathcal{FP} \times \mathcal{FP}: FP_i \text{ requires } FP_j\},\ i \neq j \wedge 1 \leq i \leq n \wedge 1 \leq j \leq n$	Require constrains between the flexibility points
(d9) $\mathcal{E} = \{(FP_i, VP_j\} \in \mathcal{FP} \times \mathcal{FP}: FP_i \text{ excludes } FP_j\},\ i \neq j \wedge 1 \leq i \leq n \wedge 1 \leq j \leq n$	Exclude constrains between the flexibility points
(d10) $\mathcal{RE} = \{R_1,..., R_q\},\ 1 \leq q$	Resources consumed by the implementations of a point
(d11) $\mathcal{QSD} = \{Q_1,..., Q_t\},\ 1 \leq t$	QoS dimensions offered by the implementations of a point
(d12) $\mathcal{RCONS}_{Vk} = \{rconsr_1(I_k),..., rconsr_q(I_k)\},\ 1 \leq q$	Consumption of the different resources by the I_k implementation
(d13) $\mathcal{QOS}_{Vk} = \{qosq_1(I_k),..., qosq_t(I_k)\},\ 1 \leq t$	QoS dimensions offered by the I_k implementation
(d14) $\mathcal{WEIGHT}_{exp} = \{weight_{exp}(exp_{element})\},\ (exp = rcons \Rightarrow element \in \mathcal{RE}) \wedge (exp = qos \Rightarrow element \in \mathcal{QSD})$	Importance associated with the resource consumption and the QoS parameters
(d15) $\mathcal{MVP} \subseteq \mathcal{VP}$	Mandatory flexibility points
(d16) $\mathcal{C}_i = \{\bigcup_{j=1}^{j=q} \mathcal{I}C_i^j\}$	i^{th} configuration

Table 6.2: Definitions for Modeling the Selection Problem as a CSP

As already said, we need to establish some criteria for choosing the new configuration, when different alternatives are possible with the current context changes. In our approach, we include the resource consumption, the adaptation cost and the QoS as selection criteria in the analysis mechanism. These criteria enable the optimization of the new configuration and improve the result of the adaptation process. For this reason, we need to introduce some expressions permitting the measure of such criteria. In particular, we define two new sets: *i)* $\mathcal{RE} = \{R_1, \cdots, R_q\}$ as the sets of resources, the use of which we can measure or know

and, *ii)* $\mathcal{QSD} = \{Q_1, \cdots, Q_t\}$ indicating quality of service dimensions. We also introduce the sets $\mathcal{RCONS}_{I_j} = \{rcons_{r_1}(I_j), \cdots, rcons_{r_q}(I_j)\}$, $QOS_{I_j} = \{qos_{q_1}(I_j), \cdots, qos_{q_t}(I_j)\}$ and $RCONF_{I_j} = \{reconf(I_0 \rightarrow I_k, \cdots, I_s \rightarrow I_k)\}$. The expressions in the first set denote the consumption of the resource r_k by the j_{th} implementation. The qos expression indicates the QoS offered by I_j. The expressions in the later set define the cost of replacing the u_{th} implementation by the k_{th} implementation. In this expression, the I_0 refers to the absence of an implementation for the variation point, which is valid when the flexibility point is *optional—i.e.*, the point is not part of the application kernel. In the current implementation of the `Decision Maker`, we parameterize the component with the values of the resource consumption and QoS dimensions. For its part, the reconfiguration cost is estimated in terms of the required operations for moving from one implementation to another.

So far, we have introduce the basic concepts and expressions that we use in the selection problem modeling. In the following sections we introduce the different functions and restrictions associated with the criteria that we consider in our approach, *i.e.*—QoS, resource consumption and reconfiguration cost.

6.4.3 Optimizing The Resource Consumption

In order to foster flexibility, in our Ubiquitous FCLs we provide different mechanisms for selecting the most suitable configuration. The simplest of the provided mechanisms searches for a valid configuration by using the implementations selected for each flexibility point. In this mechanism, restrictions are not considered and therefore the first valid configuration found will be chosen. If there is no a valid configuration, there will be no reconfiguration. On the other hand, we also give the possibility of applying three optimizations: *resource consumption*, *provided QoS* and *reconfiguration cost*. In this section we focus in the first optimization. The others will be addressed in the following sections.

Regarding the resource consumption, we define the following function to optimize:

$$\textbf{(F1)} \quad \min \left(\sum_{i=1}^{i=\#\mathcal{FP}} \sum_{j=1}^{j=\#\mathcal{I}_i} \sum_{k=1}^{k=\#\mathcal{RE}} (rcons_{r_k}(I_j) \times si(I_j) \times sfp(FP_i) \times weight_{rcons}(rcons_{r_k})) \right)$$

The previous function calculates the total resource consumption for a given configuration. We use the $d12$ definition for determining the consumption associated with each point implementation and we guarantee that it will be only considered if the implementation and, of course, the respective flexibility point make part of the considered application configuration (by using $d5$ and $d7$ definitions). Additionally, we use the $weight_{rcons}$ expression to include in the calculation the importance associated with the resource. These values are extracted from user preferences stating the relevance of different resources for the user, such as battery and memory consumption. If a resource is no relevant in the optimization process, its $weight_rcons$ should be 0.

Considering the exclude and include relations, for minimizing **(F1)** we have to satisfy the following restrictions:

(R1) $\{\forall\ FP \in \mathcal{MFP}\ : (\exists\ I \in \mathcal{C}_i /\ I \in \mathcal{I}_{FP} \wedge si(I) = 1)\}$: All the mandatory variation points—*i.e.*, the application core, make part of the configuration.

(R2) $\{\forall\ I_j, I_k \in \mathcal{C}_i :\ (I_j, I_k) \notin \mathcal{E}\}$: All the exclude constrains are respected.

(R3) $\{\forall\ (I_j, I_k) \in \mathcal{R}_i :\ I_j \in \mathcal{C}_i \rightarrow I_k \in \mathcal{C}_i\}$: All the require constrains are respected.

(R4) $\{\forall\ FP \in \mathcal{FP}\ : sfp(FP) = 1 \rightarrow \#\mathcal{IC} = 1\}$: Each selected flexibility point has one and only one implementation that has been choosen.

6.4.4 Optimizing The Provided QoS

Other dimension that we use for improving the result of the adaptation process is the QoS. Applying quality of service for selecting the new configuration is a suitable alternative since we search to improve the user experience by means of the context-aware adaptation. Therefore, it is logical that we try to maximize the value associated with the following expression:

$$\textbf{(F2)} \quad \max \left(\sum_{i=1}^{i=\#\mathcal{FP}} \sum_{j=1}^{j=\#\mathcal{I}_i} \sum_{k=1}^{k=\#\mathcal{QSD}} (qos_{q_k}(I_j) \times si_(I_j) \times sfp(FP_i) \times weight_{qos}(qos_{q_k})) \right)$$

In this function, we use the $d13$ expression to estimate the QoS offered by each variant. This value is only considered in the calculation if the implementation as well as the flexibility point are part of the configuration. Similarly to the minimization of resources, we use $d14$ to obtain the weight associated with each QoS dimension. In the optimization of the QoS we also apply the $R1$, $R2$, $R3$ and $R4$ restrictions introduced in Section 6.4.3.

6.4.5 Optimizing The Reconfiguration Cost

The last aspect that we use for selecting a new configuration is the reconfiguration cost. To calculate this cost we define the following function:

$$\textbf{(F3)} \quad \min reconf(C_c \rightarrow C_i)$$

Where $reconf$ represents the costs of changing the current configuration (C_c) by the configuration C_i. To calculate the reconfiguration cost we use the set difference $C_i - C_c$. With this difference we can obtain the implementations of the points that must be added. We also apply the operation $C_c - C_i$ for determining the implementations to remove. For example, to replace $C1$ by $C8$ in MOBIHOME (cf. Section 6.4.2), we have $C_8 - C_1 = \{TVControlModuleI, MultimediaI2, VideoRendererI\}$, which indicates that need to add the implementations for the `TV Control Module`, `Multimedia Module` and `Video Renderer` component. On the contrary we do not need to remove functionality because $C_1 - C_8$ is empty. As it will be presented in Section 6.5, we use these differences for choosing the required scripts that materialize the current reconfiguration. Depending on the components that have to be added and removed we determine the reconfiguration cost. We specific an arbitrary value for the *add* and *delete* operations of components in the architecture, which can be modified at runtime if required. In this optimization we also need to satisfy the $R1$, $R2$, $R3$ and $R4$ restrictions.

6.4.6 Decision Maker Architecture

In the previous sections, we modeled the configuration selection considering different aspects that we estimate relevant for context-based adaptations. In this section, we provide an overview of the architecture that reifies the previously introduced modeling.

In order to enable the selection of the new required configuration, we propose the simple architecture depicted in Figure 6.4. In this architecture, the `Context Policies` represent the different adaptation situations that require the modification or elimination of flexibility points implementations. The `Configuration Collector` component retrieves the decisions from these policies and delegates the configuration selection to the `Selection Orchestrator`. This last component uses a plugin-based mechanism to support the incorporation of several `Selectors` representing different selection criteria. In particular we define the `Selectors` for the optimization proposed in Sections 6.4.3, 6.4.4 and 6.4.5. Additionally, we introduce a `Simple Selector`, which selects the first valid configuration calculated from the point implementations stated by the context policies.

In our architecture, `Selectors` employ the `Restriction Manager` for retrieving the restrictions that must be fulfilled. This manager exploits different `Holders`, which are associated with one or more restrictions. Figure 6.4 includes a `Core Restriction Holder` component that guarantees the satisfaction of the $R1$, $R2$ and $R3$ restrictions. Furthermore, we provide the flexibility for incorporating other `Restriction Holders` (as well as different implementations for the `Core Restriction Holder`), which can be applied according to the `Selector` needs. The `Selectors` and `Holders` are optional, but at least one of them have to be present in the deployed `Decision Maker`. If no valid configuration is found, the resulting configuration is the current configuration of the application.

We propose this architecture because we want to modularize the different aspects associated with the constraint satisfaction problem. Unlike other solutions [Krishnakumar and Sloman, 2002, Neema and Ledeczi, 2003, Padmanabhuni *et al.*, 2006, Davidyuk *et al.*, 2008], we reified in our architecture the restrictions and optimizable dimensions in the `Holders` and `Selectors` respectively. With this modularization we enable the independent modification of each one and, when required, it is possible to easily incorporate new restrictions or optimizations. Thus, with the `Decision Maker` architecture we foster flexibility and extensibility, properties that we search in our solution allowing context-based adaptation (cf. Section 1.2).

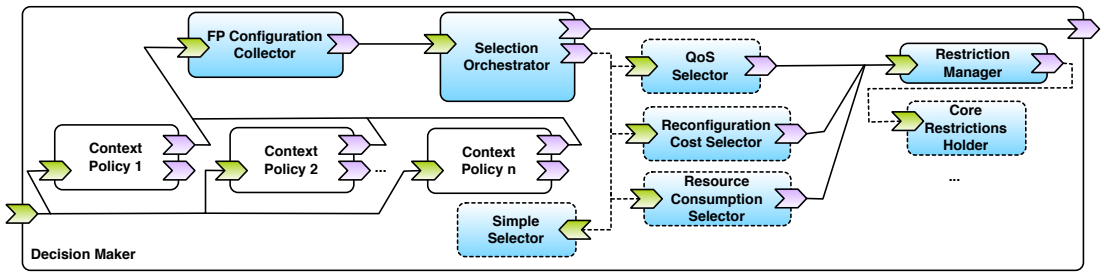


Figure 6.4: Architecture of the Decision Maker Component

6.5 Planning the Required Actions for Reaching the New Configuration

The output of the `Decision Maker` component is the new configuration as a list of implementations. Each implementation is associated with a specific flexibility point. In order to determine the required actions for reaching this configuration, we apply the set differences between the current and the new configurations. We use the difference $C_c - C_i$ to identify the optional flexibility points which implementation has to be removed. In a similar way, with $C_i - C_c$ we determine the implementation points to be added. Then, thanks to the isolation of flexibility points fostered by the context policies, we select the scripts that must be executed. In particular, each flexibility point is associated with a reconfiguration script that specifies the components to be added (resp. removed) for incorporating (resp. eliminating) a specific implementation. In the planning we also determine the order to apply the configuration. To do it, we consider the *require* relationships between the flexibility points for deciding what script should be applied first. In this analysis we assume that there is no loops in the require dependencies between the points. If this case appears, we consider it that it is a design problem in the application and therefore we can not guarantee the application consistency. Then, in the presence of loop dependencies, we do not execute any reconfiguration.

6.6 Instrumentation of the Adaptation in the FraSCAti Platform with the personalized SCA Bindings

The FraSCAti platform provides a run-time API for enabling the modification and introspection at runtime of SCA applications [Seinturier *et al.*, 2009]. The reconfiguration capabilities are based on the FScript language [David, 2005, David *et al.*, 2008], which enables the structural reconfiguration of component-based applications. Considering reconfigurations as transactions, FScript offers atomicity, state consistency, isolation, and termination as consistency criterions in the adaptation process. The satisfaction of these criterions ensures the behavioral integrity of applications after the reconfiguration. Therefore, the FScript usage in our approach guarantees that the application always will work in a suitable way after the Ubiquitous FCL execution.

The reconfiguration capabilities of the current FraSCAti platform version can be only accessed locally. However, our Ubiquitous FCLs foster the distribution of different tasks of the adaptation process. In particular, as already presented in Section 6.3, the analysis, planning and execution responsibilities are effectuated by different entities. Therefore, we provide a mechanism enabling the remote usage of the reconfiguration service. As we mention previously, our objective of keeping the approach simple (cf. property 5 in Section 6.1) motivates the SPACES connectors selection for the integration of the different entities in the FCL. Consequently, it is natural to keep a uniform mechanism and we give to the FraSCAti platform the possibility of offering its reconfiguration capabilities by applying SPACES principles.

To allow the remote access of the reconfiguration capabilities in FraSCAti, we introduce the `Reconfiguration Service` (cf. Figure 6.2). This service is deployed as an SCA application together with the FraSCAti runtime on the entity hosting the adaptive application. The `Reconfiguration Service` encapsulates the FScript Engine providing introspection and interception capabilities on the SCA applications. These functionalities are exposed by using the Resource-Oriented bindings presented in Section 5.4. In particular, this component has the following responsibilities:

1. *Processing the reconfiguration requests*: the `Reconfiguration Service` receives the SPACES requests indicating the execution of a script. These requests are specified using the POST interface from REST. The script is embedded in the REST request body as it can be seen below:

```
POST /alice-mobile-reconfiguration-service/app1/reconf1 HTTP/1.1
Host: server.inria.fr:8080
Content-Length: 1390
Content-Type: application/xml, charset=utf-8

<?xml version="1.0" encoding="utf-8"?>
<reconfiguration-service:message name="reconf1"
               xmlns:rs="http://www.spaces.rs.org/XMLSchema"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="www.spaces.org/XMLSchema
                     http://picoforge.int-evry.fr/projects/svn/cosmos/spaces/rs.xsd">
  <rs:adaptation-decision-time>100</rs:adaptation-decision-time>
  <rs:flexibility-point-list>
             ViewFP ControllerFP TVControllerModuleFP MultimediaModuleFP VideoRendererFP
  </rs:flexibility-point-list>
  <reconfiguration-service:script name="VideoRenderer">
   action add-video-renderer(mobihomeApp) {
      var vr = adl-new("mobihome.video.renderer.vr");
      controller = $mobihomeApp/child::controller;
      bind($controller/interface::videoRenderer, $vr/interface::videoRenderer);
   }
  </reconfiguration-service:script>
  <reconfiguration-service:script name="multimediaModule2">
   action add-multimediaI2(mobihomeApp) {
      var mm = adl-new("mobihome.multimedia.module.mm2");
```

```
        var tvm = adl-new("mobihome.tv.controller.module.tvm");
        controller = $mobihomeApp/child::controller;
        vr = $mobihomeApp/child::vr;
        bind($controller/interface::TVControllerModule, $mm/interface::multimediaModule);
        bind($controller/interface::multimediaModule, $mm/interface::multimediaModule);
        bind($controller/interface::videoRenderer, $mm/interface::videoRenderer);
    }
  </reconfiguration-service:script>
</reconfiguration-service:message >
```

In the two specified scripts we include the necessary actions for replacing configuration C_1 by C_8 in our MOBIHOME example (cf. Section 6.4.1 and 6.4.2). The `Reconfiguration Service` triggers the execution of these actions on the correct application by using the URI associated with the request. The actions are executed in the same order they are defined in the request. The adaptation decision time (used for triggering local loops execution) and flexibility point implementations list (already discussed in Section 6.4), are informations kept in the client-side for adaptation purposes.

2. *Processing the introspection requests*: Respecting the REST principles, the information associated with the structure and different states of the component from an SCA applications can be retrieved using the GET operation. Below, we define a request example for introspecting the `vr` component:

```
GET /alice-mobile/mobiHome/vr HTTP/1.1
Host: device.inria.fr:8080
Accept: application/xml, application/json
...
```

The associated response would look like that:

```
HTTP/1.0 200 OK
Date: Thu, 7 Dec 2010 14:24:44 GMT
Content-Type: application/xml
Content-Length: 621

<?xml version="1.0" encoding="utf-8"?>
<introspection-service:message name="c1_request"
               xmlns="http://www.osoa.org/xmlns/sca/1.0"
               xmlns:is="http://www.spaces.is.org/XMLSchema"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="www.spaces.org/XMLSchema
                     http://picoforge.int-evry.fr/projects/svn/cosmos/spaces/is.xsd">
   <is:component-info name="vr" status="active">
      <component name="vr">
          <implementation.java class="mobihome.renderer.lib.VideoRenderer"/>
          <service name="videoRenderer">
              <interface.java interface="mobihome.renderer.api.IVideoRenderer"/>
              <binding.sca/>
          </service>
      </component>
   </is:component-info>
</introspection-service:message>
```

3. *Management of the applications waiting for a reconfiguration*: The `Reconfiguration Service` component keeps the information of the applications waiting for a reconfiguration. Each configuration can be considered as a REST resource, which is accessible once the execution of the Ubiquitous FCL is triggered. Therefore, each resource has a unique identifier associated that guarantees the execution of the reconfiguration on the correct application. When the adaptation is executed, the resource becomes unavailable until the next execution of the loop.

4. *Collection of the adaptation decision times and new implementations of flexibility points composing an application*: When the `Adaptation Server` requests the execution of a reconfiguration, it sends the time that took the analysis and planning as well as the list of the implementations of the points that compose the application after the reconfiguration. This **adaptation decision time** is used for deciding the waiting time for the reconfiguration script (cf. Section 6.7). On the other hand, the list of implementations associated with each flexibility point is required for executing again the FCL.

By defining the `Reconfiguration Service` as an SCA application, we keep the clear separation of this concern from the adaptive applications and enable the use of the service by different Ubiquitous FCLs adapting the applications on the client entity. Furthermore, the application of REST in the last phases of the adaptation process allows us to keep a coherent approach, fostering simplicity and resuse of the different elements and mechanisms that make part of it.

6.7 Local Feedback Control Loops

In the previous section, we have introduced the mechanisms for supporting adaptation decisions in the Ubiquitous FCLs. The idea of having these FCLs is to benefit from the most powerful entities in the environment in order to determine the required reconfigurations considering the context information and available services in the environment. However, in these global control loops we need to consider the possible communication problems between the devices hosting the adaptive application and the entities deciding the reconfigurations. In particular, the following issues should be considered:

1. What happen if the context information cannot be sent to the Adaptation Server?
2. What happen if the adaptation takes a lot of time?
3. What happen if the reconfiguration service of the adaptive application is not more available when the required configuration is decided?
4. If the adaptation of several applications is managed at the same time, how to guarantee the correspondence between a determined reconfiguration and an application?

To deal with the first two issues we define Local Feedback Control Loops. These loops introduce a certain autonomy degree in the entities hosting the adaptive applications because they provide support for making local decisions based on reactive *ECA* (Event-Condition-Action) rules. The idea is to keep as simple as possible the local decisions because they are conceived as an auxiliary mechanism for the Global Feedback Control Loops. Furthermore, the entities hosting the applications have a limited knowledge about the environment. This means that these entities have not access to all the available context sources and are not aware of the available services. Therefore, the decision-based on an incomplete knowledge should not have an important impact on the application structure. The changes should be limited to component parametrization and the disabling and enabling of functionalities of the application.

The decisions that can be made by local loops include simple application recovery and application deactivation. The data used in these simple decisions are the resource information (*e.g.*, battery level, available memory), connection state and user preferences (if they are stored in the device). These informations are independently processed and there is no consolidation of the adaptation decision as in the ubiquitous FCLs. As already stated in Section 6.4, the global loops aggregate the decisions from the different context policies. These policies, running on the most powerful devices in the environment, collect information from different sensors, devices and consider the service availability in order to determine the application configuration. Therefore, the global loops have the capability of determining the addition, elimination or modification of the flexibility points that make part of the applications.

Figure 6.5 depicts a snippet of code for a simple ECA rule (upper part) and its associated script file (lower part). As it can be seen, the condition (line 7 and 8) includes the evaluation of the facts as well as the presence of some components that are involved in the script. On the other hand, the script file also shows that we are just disabling the components associated with multimedia functionalities in MOBIHOME.

```
1    @Service(IBatteryRule.class)
2    public class BatteryRules implements IBatteryRule{
3          ...
4          @Property private String scriptName;
5
6          public String batteryLowRule(){
7                  if(batteryLevel<50 &&
8                    validConfiguration()){
9                          return scriptName;
10                 }
11                 return null;
12         }
13
14         private boolean validConfiguration()
15         {
16                 return ids.contains("vr") &&
17                         ids.contains("mm");
18         }
19   }
```

```
1  action disable-multimedia-components(mobiHome) {
2      mm = $mobiHome/child::mm2;
3      vr = $mobiHome/child::vr;
4     controller = $mobiHome/child::controller;
5      stop($controller);
6      stop($vr);
7     unbind($controller/interface::multimediaModule[client(.)]);
8      unbind($controller/interface::videoRenderer[client(.)]);
9  }
```

Figure 6.5: Simple ECA Rule Example and its Associated Script

Entity 2 in Figure 6.2 contains a Local FCL. As it is depicted, a Local Control Loop follows the same structure of the global one. The monitoring part used in the global FCL is also exploited in the local one. In fact, entities hosting Local FCLs have additional functionality for determining when the decision is made locally. Figure 6.6 depicts the detailed monitoring part for these entities. The presented architecture is inspired by the context policy concept from the COSMOS context manager [Conan *et al.*, 2007, Rouvoy *et al.*, 2008]. In particular, we include the nodes that compose the core of the context policy[8]—*i.e*, all the required components for executing the adaptation. Below we describe these components.

1. `Adaptation Trigger`: Decides, based on the adaptation orchestrator availability and the waiting time for the reconfiguration decision in the global loop, if the decision has to be done locally. If the component delegates the adaptation decision to the global loop, it

[8]The context policy, the ECA rules and their reconfiguration scripts are deployed together with the applications on the host devices.

will send the information collected by the `Adaptation Info Aggregator` as well as the waiting time estimated by the `Delay Monitor`;

2. `Adaptation Info Aggregator`: The main responsibility of this node is the aggregation of the information required in the adaptation process. This information includes the current application configuration (*i.e.*, the different components that make part of the application), the flexibility points associated with the current functionality as well as other context information (*e.g.*, battery level, connection quality, and available memory);

3. `Delay Monitor`: This component supervises the time of the global loop execution. In other words, the Delay Monitor receives (from the `Adaptation Trigger`) the time when the request with the context information was sent and starts a timer for waiting the reception of the associated reconfiguration actions. If the timer expires, it notifies to the `Adaptation Trigger` component. The waiting time can be determined in three ways: *i)* Using an arbitrary value (that can me modified at runtime when required), *ii)* calculating the median historical value of the reconfiguration and *iii)* employing the application complexity. The first time that the ubiquitous FCL is successfully executed, there is no way to calculate the median historical value. The component estimating this value (the `Historical Estimator` component in Figure 6.6) therefore has a property for specifying a default media value. When several executions of the ubiquitous FCL are done, the `Historical Estimator` component uses the "adaptation decision time" values from reconfiguration requests to make the calculation. On the other hand, the `Complexity-Based Estimator` component employs the number of components multiplied by an arbitrary value representing the weight associated with each component. Because we apply a plugin-based mechanism for the waiting time calculation, new ways of doing it can be added easily. Finally, the `Delay Monitor` component has a property for specifying how to calculate the waiting time.

4. `Adaptation Server Monitor`: Verifies the presence of the entity orchestrating the Ubiquitous FCL activity.

5. `Application Configuration Monitor`: This component collects the information about the application structure upon demand. This information is required in the local FCL for deciding the new application configuration.

6. `Implementations Collector`: Stores the different flexibility points that characterize the current application configuration. As it was explained in Section 6.4, these points represent a set of functionality currently offered by the application. This information is vital in the ubiquitous FCL for determining the required actions in planning phase that will allow the reaching of the new application configuration.

The generic part of the context policy architecture serves to deal with the already mentioned `Adaptation Server` (cf. Section 6.2) unavailability and delays in the arrival of the global decision. However, associated with the last issue we find two additional problems: *i)* the local FCL can be in execution while the global adaptation decision arrives and, *ii)* the decided local adaptation may have already been executed or be in progress. For dealing with these two issues, we follow a simple strategy establishing that the first configuration that arrives is executed and the other one is discarded. This strategy is valid because the local or global decisions are valid for dealing with the identify adaptation situation. Furthermore, we guarantee that there will be no conflicts between the execution of the reconfigurations. To implemented the strategy, the `Adaptation Trigger` assigns a unique identifier following the URI format (called adaptation identifier) to the triggered adaption that will be used to receive the reconfiguration. This adaptation identifier is used in the `Reconfiguration Service` component for creating a REST resource enabling the execution of the adaptation. When the local or global reconfiguration arrives, this resource is deleted. In this way, only one of the reconfigurations will be executed.

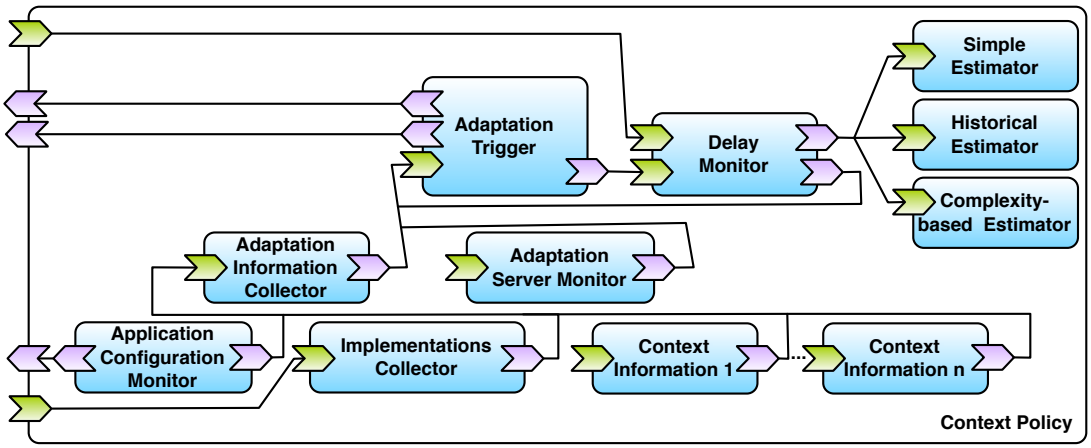


Figure 6.6: Context Policy for Entities Hosting a Local Feedback Control Loop

So far we have considered the adaptation issues associated with the entities hosting the local FCLs. However, as we already mentioned, it is also necessary to deal with the unavailability of the reconfiguration service of the application and the management of the adaptation of several applications. Both of them concern the entity orchestrating the Ubiquitous FCL—*i.e.*, the `Adaptation Server` in Figure 6.2. To deal with the unavailability of the reconfiguration service, the waiting time for the apparition of this service can be arbitrarily stated or calculated. The calculation is done by using the waiting time transferred with the adaptation information and the elapsed time between the information reception and the end of the adaptation decision. Therefore, if the service is not available, several attempts are done until the transfer is successful or the waiting time expires. For its part, the management of the several adaptive applications at the same time is not a problem because we are using unique adaptation identifiers following the REST principles. In fact, we use these identifiers in order to select the correct reconfiguration service.

6.8 Summary

In this chapter, we have described how we build *Ubiquitous Feedback Control Loops*, our approach for providing context-based adaptation. In this approach we apply the elements introduced by SPACES connectors in order to provide a simple solution for dealing with the context discovery and dissemination until the execution of the reconfigurations. The *Ubiquitous FCLs* consider the adaptation as a process, where the tasks composing it can be spread between several entities. The approach provides certain autonomy level for dealing with adaptation by enabling the discovery of new services and adaptive applications that can be integrated into the process. This autonomy is enhanced with the possibility of including *Local Feedback Control Loops* that make simple local decisions.

In order to deal with the core tasks of the adaptation process—*i.e.*, the *Identification of New Configurations* and the *Determination of the required actions to meet the configuration*, we define a strategy that applies CSP techniques. In particular, this strategy provides the flexibility to select the new application configuration by considering not only the context information but also additional dimensions to optimize such as QoS, reconfiguration cost and resource consumption. Our strategy provides the extensibility required for adding new optimization dimensions as well the associated restrictions.

In this chapter and in the previous one, we have introduced the major contributions of this dissertation—*i.e.*, **the SPACES metamodel**, **the SPACES Connectors concepts**, **the generic archi-**

tecture of these connectors and **the Ubiquitous FCLs** enabling context-based adaptation. In the following chapter we present different scenarios that we apply in order to provide the qualitative and quantitative evaluations of our contribution.

Part III

Validation

Chapter 7

CASE STUDIES

Contents

In this chapter, we introduce three case studies validating our proposal—*i.e.*, **SPACES Connectors** as well as the **Ubiquitous Feedback Control Loops**. The presented case studies enable us to confirm the suitability of our approach in ubiquitous environments by using different kinds of devices and communication protocols.

The first case study describes a **caching or off-loading** situation for choosing between two application configurations. This case shows the simplicity fostered by SPACES Connectors allowing the distribution of a context policy in a easy and transparent way.

We also introduce the **TRACK.ME** platform as second case study, a service-oriented platform for tracking the activities of mobile users. We use this platform for illustrating the advantages of applying SPACES Connectors concepts into SCA.

Finally, in the third case study we introduce the **DIGIHOME** platform, which allows the adaptation of mobile applications by exploiting the Ubiquitous FCL concepts. The last case study allows us to provide a validation of the complete approach.

Chapter Organization

In the rest of the chapter we introduce the **caching or off-loading** scenario in Section 7.1, **TRACK.ME** in Section 7.2 as well as the **DIGIHOME** platform in Section 7.3. In the three cases we present the experimentation and the advantages of our approach that we can confirm with it. Section 7.4 discusses the drawbacks associated with our approach. We conclude the chapter with a summary of the discussed case studies (cf. Section 7.5).

7.1 A Caching Off-Loading Situation

In this section we introduce a simple scenario that serves us as a starting point for the validation of our proposal. The experience from the SPACES Connectors implementation as well as the executed tests confirm the simplicity and flexibility of our approach.

7.1.1 Description

This case study, adapted from the scenario proposed in [Conan *et al.*, 2007], introduces some challenges in terms of device heterogeneity for deploying distributed context policies in ubiquitous computing environments.

In particular, we assume that the user of a mobile device executes a distributed application for analyzing the smartphone usage while roaming. The WiFi connection of the mobile device is subject to fluctuations. Thus, in order to tolerate such fluctuations, the middleware platform executing the application can be augmented with the capabilities *i)* of caching application entities into a software cache and *ii)* of off-loading application treatments on (more powerful) hosts of the wired network. In order to choose between caching and off-loading, we use the context policy depicted in Figure 7.1. This policy computes the memory capacity ④ as the sum of the average free memory ② plus the average free swap ①. The context policy also monitors the WiFi network connectivity. The context nodes configuration detects fluctuations and computes the adjusted bit rate (average bit rate during periods of strong connectivity) ⑤. When the memory capacity is sufficient, but the adjusted bit rate low, caching is preferred. When the memory capacity is low, but the adjusted bit rate sufficient, off-loading is preferred. In the two other cases, the end-user or the middleware platform gives their preferences (caching or off-loading) ③. Once the decision is taken, connectivity information is used to detect the activation instants for caching/off-loading when the connectivity mode evolves (from strongly connected to disconnected and *vice versa*).

In order to create a solution dealing with the caching or off-loading decision, we employ the COSMOS context manager. In the next section we provide an overview of this context manager before discussing how we employ the notion of SPACES Connectors in the case study.

7.1.2 COSMOS: COntext entitieS coMpositiOn and Sharing

COSMOS is a component-based framework for managing context information in context-aware applications [Conan *et al.*, 2007, Rouvoy *et al.*, 2008]. In particular, COSMOS identifies the contextual situations for which a context-aware application is expected to react. These situations are represented as *context policies* that are hierarchically decomposed into fine-grained units called *context nodes*. Figure 7.2 depicts the structure of a context node.

In COSMOS, a Context Node is context information modeled by a software component. The relationships between context nodes are *sharing* and *encapsulation*. The sharing of a context node—and, by implication, of a partial or complete hierarchy—corresponds to the sharing of part or all of a context policy. Context node leaves (the bottom-most elements, with no descendants) encapsulate raw context data obtained from collectors, such as operating system probes, sensors near the device, and user preferences in profiles. Context nodes should provide all the inputs necessary for reasoning about the execution context. This is why COSMOS considers user preferences as context information. Thus, the role of a context node is to isolate the inference of

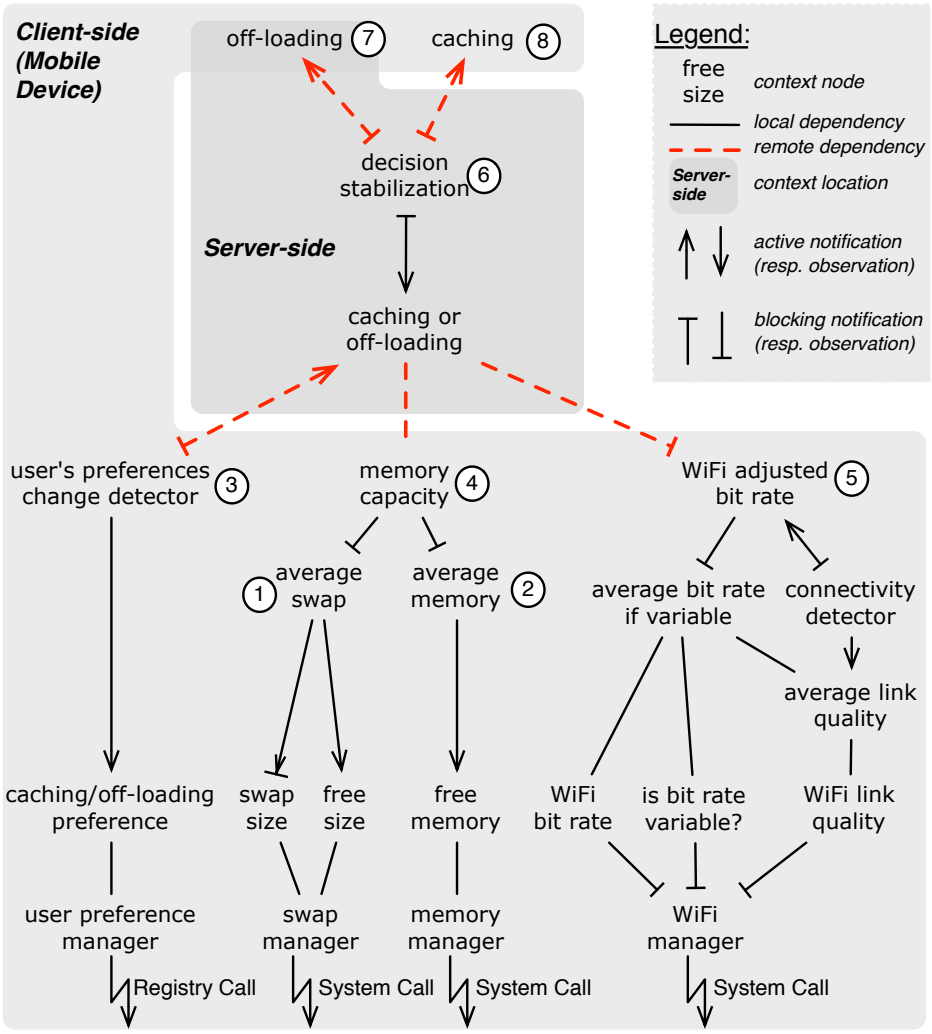


Figure 7.1: Overview of a Distributed Context Policy.

high-level context information from lower architectural layers responsible for collecting context information.

Each context node can tune its behavior by configuring the following properties:

Active/Passive. An active node is associated with a thread of control (attached to the Activity Manager). Typical examples of active nodes include a node in charge of the centralization of several types of context information, a node responsible for the periodic computation of higher-level context information, and a node to provide the latter information to upper nodes. A passive node obtains context information upon demand.

Observation/Notification. Communication into a context node's hierarchy can be top-down or bottom-up. The former—implemented by the interface `Pull`—corresponds to observations that a parent node triggers. The latter—realized by the interface `Push`—corresponds to notifications that context nodes send to their parents.

Pass-through/Blocking. Pass-through nodes propagate observations and notifications while blocking nodes stop the traversal. For observations, COSMOS transmits the most up-to-

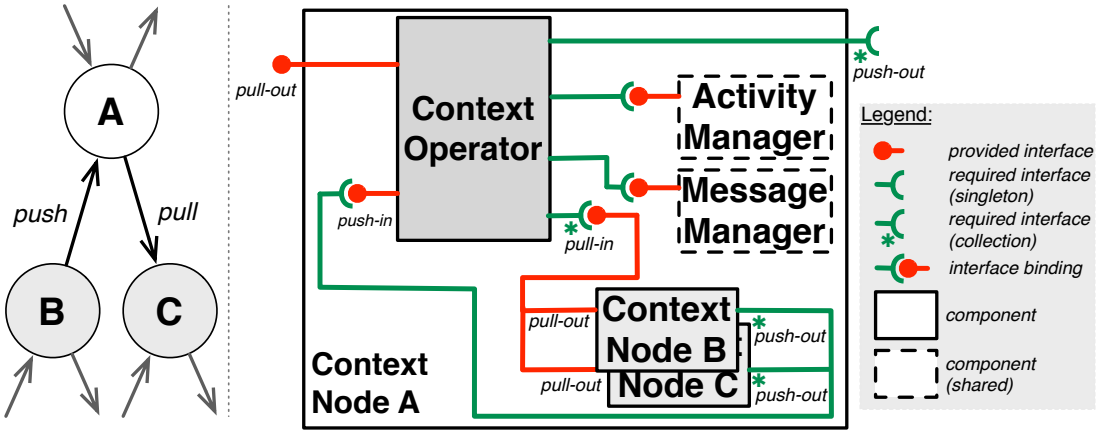


Figure 7.2: Architecture of a COSMOS Context Node.

date context information without polling child nodes. For notifications, COSMOS uses context data to update the node's state, but it does not notify parent nodes.

The context nodes exchange *messages* that are created and manipulated by the Message Manager. The context nodes send and receive these *messages* through the `Pull` and `Push` interfaces. Each `Message` is composed of a set of `Chunks` and encloses a set of sub-messages. Each chunk reflects a context information as a 3-tuple `(name,value,type)`.

7.1.3 Distribution of the Context Policy

The context policy defined in [Conan *et al.*, 2007] for the given scenario is based on the COSMOS framework. However, this framework lacks of distribution capabilities. Therefore, the context policy for the caching off-loading situation is defined and implemented as a local policy. To apply the SPACES Connectors concepts and in this way spreading the policy, we split the context nodes between two entities: the *application server* and the *mobile device*. As depicted in Figure 7.1, the mobile device (Client-side) exploits both the context information caching ⑦ and off-loading ⑧, while the application server only requires to access the context information off-loading in order to provision the appropriate resources for the mobile device when needed. Furthermore, in order to reduce the burden in the mobile device, we assign the decision stabilization mechanism ⑥ to the application server (Server-side). Thus, the mobile device has to send the context information to the application server, which will decide the best strategy to apply and will notify back the decision to the mobile device.

In order to implement the remote dependencies in the context policy in a transparent way, we concretize the generic SPACES connectors architecture presented in Section 5.6 following the notion of COSMOS context nodes (cf. Section 7.1.2). Figure 7.3 depicts the SPACES Connectors architecture as context nodes. As observed, we apply different concepts offered by COSMOS. In particular, benefiting from the Fractal Component Model support for component hierarchies and sharing, we include in the node operator the functionality for dealing with context requests and resources. Furthermore, to finely control the resource consumption we share the `Activity Manager` and `Message Manager` components. By defining the SPACES connectors in this way, we foster reuse and the clear separations between context processing and context dissemination.

Implementation details

We can materialize SPACES Connectors as an extension of any legacy web servers [Wikman and Dosa, 2006, Nokia, 2008, Pham and Gehlen, 2005, Srirama *et al.*, 2006,

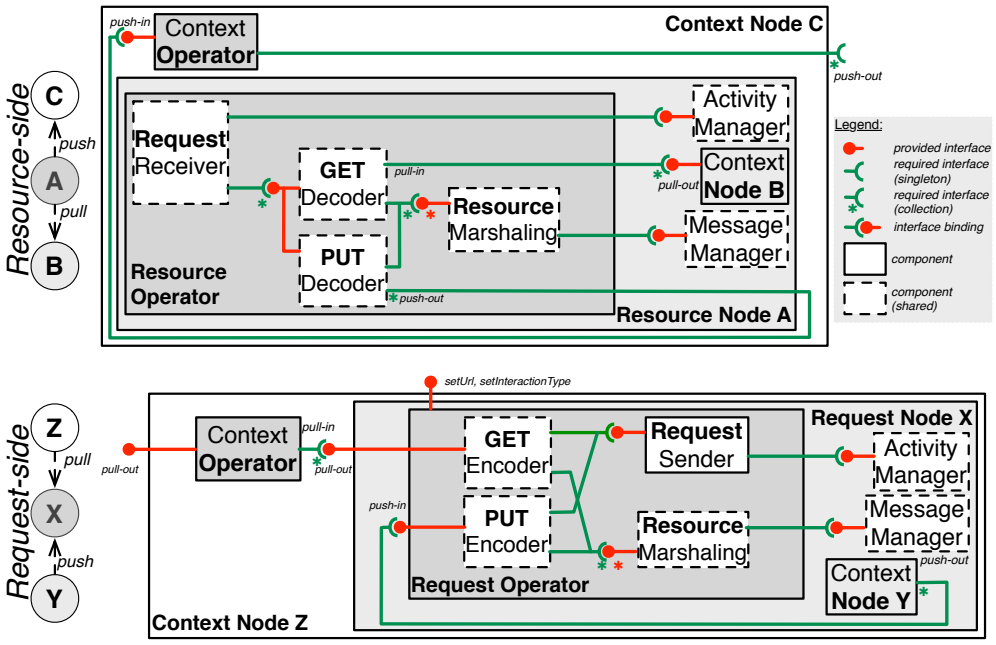


Figure 7.3: SPACES Architecture Based on COSMOS Context Nodes

The Apache Software Foundation,] (or any other communication stack). The implementation of SPACES Connectors reported in this case study is based on the COMANCHE[9] web server [Bruneton *et al.*, 2006b] and the COSMOS framework 0.1.5. Both COSMOS and COMANCHE are based on the FRACTAL component model and use the JULIA[10] implementation of the FRACTAL runtime environment [Bruneton *et al.*, 2006b]. The choice of these technologies provides an elegant way of integrating both frameworks at an architectural level by means of the FRACTAL ADL toolset [Leclercq *et al.*, 2007]. In this way, the configuration and the evolution of SPACES context policies consist in the selection, addition, or replacement of software components and the tuning of their associated parameters.

7.1.4 Quantitative Evaluation: Measuring the Performance of the Approach

The objective of the benchmark associated with this first case study is to show the cost introduced in the context mediation by the SPACES Connectors implementation based on Fractal Component Model and the COSMOS Framework. To do this, we have measured the time for context processing by executing the context policy with and without distribution. In the tests we employ laptops as well as mobile devices. Furthermore, we illustrate the versatility of our approach by executing the benchmark with different resource representations. In particular, we evaluate the performance by using the Java Object Serialization as well as JSON and XML documents.

Context Aggregation

To reduce the number of exchanged messages and measure the actual overhead introduced by SPACES Connectors, context nodes published with the same properties—i.e., active/passive observer or notifier, can automatically aggregate and separate in each device the context information they exchange. For testing purposes, we make all the context nodes in the context policy active observers (cf. Section 7.1.2). In this way, we need to introduce just two new context nodes

[9] COMANCHE web server: http://fractal.ow2.org/tutorials/comanche.html
[10] JULIA: http://fractal.ow2.org/julia

in the policy: *i)* The first one aggregates the context information user's preferences, memory capacity and adjusted bit rate information on the client-side and, *ii)* the second node separates this information in the server-side. The aggregated context information is therefore published instead of the individual context nodes.

Test Bed Configuration

We have tested several configurations of the scenario using two MacBook Pro laptops, with the following software and hardware configuration: 2.4 GHz processor, 2 GB of RAM, AirPort Extreme card, Mac OS X 10.5.6 (kernel Darwin 9.6.0), Java Virtual Machine 1.6.0, and JULIA 2.5.2. The mobile client is a Nokia N800 Internet Table with 400 Mhz, 128 MB of RAM, interface WLAN 802.11 b/e/g, Linux Maemo (kernel 2.6.21), CACAOVM Java Virtual Machine 0.99.4, and JULIA 2.5.2. We used the libraries XStream 1.3.1[11] and JSON-lib 2.2.3[12] to serialize context information as XML and JSON documents, respectively.

Measurement Collection

Table 7.1 summarizes the results we obtained when executing the configurations of the context policy we introduced in Section 7.1.3. The reported values correspond to the observation delay—*i.e.*, using the *pull* mechanism—for retrieving the context information off-loading. The values represent the average of 1000 successful observations.

Providers Configuration	Provider	Server	Retrieval Latency (ms)		
			Objects	JSON	XML
a) No Provider	N/A	MacBook Pro	3		
b) No Provider	N/A	N800	8.8		
c) 1 Local Provider	N/A	MacBook Pro	8.25	8.33	10.6
d) 1 External Provider	MacBook Pro	MacBook Pro	37.5	42.25	48.25
e) 1 External Provider	N800	MacBook Pro	317.4	391.3	N/A

Table 7.1: Overhead Introduced by SPACES Connectors in the Context Mediation

In each test, two messages are actually exchanged between the client and the server. The first message aggregates the context information user preferences, memory capacity, and WiFi adjusted bit rate according to the already discussed context aggregation. For its part, the second message corresponds to the result of evaluating the entire context policy.

7.1.5 Results Discussion

In configurations *a* and *b* we execute the policy locally using a laptop and a mobile device respectively. From these first two evaluations, we observe that the mobile device almost triples the processing of the context policy. We exploit these informations for determining the actual overhead introduced by our SPACES Connectors implementation. In particular, considering that the context processing is locally performed in $3ms$ in configuration *a*, we can observe that the overhead introduced by SPACES for mediating context information costs around $2.6ms$ per message compared to configuration *c* with the java serialization. This time includes the marshalling and unmarshalling of the message as well as its transmission.

[11] XStream: http://xstream.codehaus.org
[12] JSON-lib: http://json-lib.sourceforge.net

We also observe that the network introduces an additional transport cost of 350% approximately regarding the configuration *c* and *d*. The former uses two different virtual machine instances, one of them having the server role and the other the provider role. On the other hand, we notice that the deployment of the distributed context policy on a mobile device (configuration *e*, object representation) introduces an additional overhead of 750% compared to the deployment on two laptops (configuration *d*, object representation). This overhead is mostly due to the limited processing capacity of the mobile device as illustrated by the deployment of the context policy in configuration *b*. Furthermore, in this experimentation, the deployment of the XML version of the distributed context policy was not possible due to a limitation of the Java Virtual Machines used on the mobile device (CACAOVM & JamVM).

Although the overhead introduced by SPACES is reasonable, the flexibility offered by the proposal supports the interoperability of context policies with a wide range of mobile devices, such as Symbian, Android, and iPhone platforms. Furthermore, the support for multiple resource representations leverages the integration of context-awareness support into legacy systems. In particular, XML documents generated by SPACES Connectors can easily be processed by tools, such as XQuery or XSLT to query or transform the context information requested by an application.

7.1.6 Qualitative Evaluation

The realization of the SPACES connectors for this first study case enables us to confirm the advantages of our approach. In particular, respecting the SPACES architecture, we provide isolation, simplicity, flexibility, extensibility, reusability and transparent context integration. These advantages are discussed below.

1. **Functionality isolation based on CBSE and REST principles**: As several proposals presented in Chapter 3, we apply in the SPACES Connectors architecture the *encapsulation*, *abstraction* and *modularity* promoted by CBSE [Taylor *et al.*, 2009]. In practice, thanks to these properties, the concrete SPACES architecture discussed in Section 7.1.3 provides a clear separation of the REST triangle of nouns, verbs and representations. This separation is a keystone for problem detection as well as for reuse purposes.

2. **Simplicity, flexibility and extensibility**: the core of our solution is based on REST. Therefore, it is logic that the concrete implementation of SPACES reflects the **simplicity** associated with this architectural style. Some middleware platforms [Román *et al.*, 2002b, Bellavista *et al.*, 2003, Gu *et al.*, 2004, Gu *et al.*, 2005, Davidyuk *et al.*, 2004, Yau *et al.*, 2004, Yau *et al.*, 2002] promote the usage of specialized and complex architectural elements that make their application difficult. Unlike these platforms, and respecting the objective of "*consider what exists*" (cf. Section 1.2) in SPACES we foster the application of existing standards and elements, which facilitates not only its use for context mediation but also its development. Due to this, in the SPACES implementation based on Fractal Component Model we have reused existing functionalities for dealing with the context request and the marshalling and unmarshalling of the context information. On the other hand, the functionality isolation brings **flexibility** in our approach. In particular, according to the context mediation needs, it is possible to customize the protocols and marshallings that will be integrated in SPACES Connectors in their deployment. In a similar way, the well definition of responsibilities between the different components of the architecture supports the integration of new interaction ways and representations.

3. **Reusable solution**: the SPACES connectors based on the notion of context nodes do not depend on the context policy of the case study. These connectors have been conceived for being used in different kinds of context policies requiring distribution as well as integration with legacy systems or applications.

4. **Transparent context integration**: the realization of SPACES as context nodes does not impact at all the development of context policies. In fact, the SPACES implementation based

on COSMOS enables the distribution of context nodes without affecting the logic dealing with the context processing. The application of the context node concepts makes the application of SPACES natural in this specific kind of context manager.

7.2 The TRACK.ME Platform

In the second case study we present TRACK.ME, a service-oriented platform that allows scientists to easily set up tracking experiments. With this platform we want to illustrate the advantages associated with the introduction of the SPACES Connectors concepts into the SCA component model. Furthermore, TRACK.ME is conceived to deal with the already mentioned heterogeneity in terms of information, data representation and implementation technologies from ubiquitous environments.

7.2.1 Platform Description

TRACK.ME is a distributed platform designed as a modular assembly of services, called TRACK.ME *app*s, to foster reuse and sharing. Based on the SCA component model and FraSCAti (cf. Section 2.3.3), the services of the platform and their assembly can be customized regarding the scientific needs and dynamically reconfigured along the experiments. Below we describe the generic services for collecting activity traces that make part of the service-side of the platform.

1. **Trace database**: Represents the root *app* of the TRACK.ME platform since it is in charge of ensuring the durability of the experiences. This *app* is configured with the XML schema used during the experience (*e.g.*, GPX [TopoGraphix,] or KML [Open Geospatial Consortium,]) and creates a new database upon deployment. This database will be fed by the activity traces of mobile users and then accessed by the scientist to consult the collected datasets.

2. **Trace manager**: Is a configuration *app*, which can be used to create a new experiment. In particular, this *app* is used to configure the time and geographic filtering mechanisms, which are used in the platform to preserve the privacy of the participants.

3. **Trace resource**: This *app* is the entry point of the participants involved in a TRACK.ME experiment. The *trace resource app* is remotely exposed as a REST resource supporting various representations of activity traces to be posted. Thus, this *app* is in charge of *i)* converting the activity trace into its XML representation if needed and *ii)* checking the conformance of the submitted document before delegating its processing to another *app*.

4. **Trace anonymizer**: Represents an example of *trace processing app* that preserves anonymity of the collected activity traces. This includes the actual filtering along the time and space dimensions and the clearing from non-relevant informations. Although the privacy of the participants can be ensured by client-side filtering mechanisms, we also provide a dedicated *app*, which can be used when the mobile device only offers limited processing capabilities.

5. **Trace visualizer**: This *app* is used by the scientist to visualize the activity traces, which have been already collected by the platform. Depending on the type of activity traces, this *app* offers various visualizations of the information. For example, when considering GPX or KML activity traces, the *trace visualizer app* can represent the activity traces using the *Google Map API*.

6. **Trace exporter**: Provides an interface to the scientist for extracting a dataset from the trace database. In particular, this *app* exploits XQuery to extract a document containing a subset of the collected activity traces. Given the variety of scientists using the TRACK.ME platform, standard extraction queries are also provided by the *app* to extract the activity traces

according to participant identifiers, date interval, period of the day, and geographic area parameters.

7. **Trace archiver**: Is an *app* for archiving a given dataset of activity traces in a public archive. Therefore, this *trace archiver app* makes the connection with third party archives like the CRAWDAD initiative [Kotz and Henderson,].

8. **Trace mutation**: This *app* is an example of *exploitation app* that uses genetic algorithms to mutate the collected activity traces. Mutated activity traces are useful to evaluate scientific models and algorithms against alternative scenarios randomly generated from real activity traces.

9. **Trace appstore**: Provides a private *appstore* service to leverage on the deployment of client-side tracking applications. This approach avoids the publication of the client-side tracking applications on dedicated platforms (*e.g.*, the AndroidMarket platform) and rather provides a private *app factory* to easily install the tracking applications on the participant devices. Once installed, the client-side application can download the experiment configuration exposed by the *trace resource app*.

Figure 7.4 depicts the SCA-based architecture of the TRACK.ME platform. In this architecture, Track.Me instances are hosted by the Track.Me cloud. Each instance reifies the environment provided to the scientist for creating and managing the tracking of activity traces. Therefore, a Track.Me instance is an SCA composite isolating the TRACK.ME *apps* selected by the scientist. These *apps* are designed and implemented as SCA components, which expose their services as SCA bindings. In particular, the proposed architecture offers the Trace Resource, Trace Visualizer and Trace Manager services as REST resources by using the resource-oriented bindings introduced in Section 5.7.1.

TRACK.ME Client-Side

Considering the variability in terms of devices providing the trace information and benefiting again from the COSMOS context framework concepts we introduce *Activity Trace Policies*. These policies are composed by three parts: the Device Part, the Generic Part and the Experiment Part. Figure 7.5 illustrates these distinct parts. The Device Part contains the data collectors, which are specific to the tracking device, and can be selected by the scientist to describe the activity traces. The Generic Part corresponds to the specification of a generic piece of software deployed on the tracking device to aggregate, store, encode, and upload the collected data. Finally, the Experiment Part includes the configuration built by the scientist when creating a new tracking experiment and specifies the data to be collected by the participants.

The Activity Trace Policy example in Figure 7.5 shows a configuration where the scientist is interested in tracking the position of the participants as well as the list of wireless networks and bluetooth devices available in their surroundings. This configuration is deployed within the smartphone for describing how the collected data is composed. In the TRACK.ME case, local processing can be exploited to filter out the undesired activity traces on the smartphone prior to their upload on the TRACK.ME platform server-side to preserve the privacy of the participants. The Activity Trace Policy specifies also whether the data is computed through periodic observations or spontaneous notifications of the client device. In Figure 7.5, upon change in the physical position of the participant ①, the GPS coordinates of the device are filtered according to space and time constraints ②. If the collected data are not discarded by the filters, they are aggregated with the list of networks and Bluetooth devices available in the surrounding ③ before being transmitted to the TRACK.ME platform ④. This transmission is performed asynchronously in order to tolerate network disconnections. Thus, the applied strategy aggregates and stores locally the dataset and waits upon the availability of the TRACK.ME platform. When the platform is available, the collected data are encoded according to the selected representation and sent over the network.

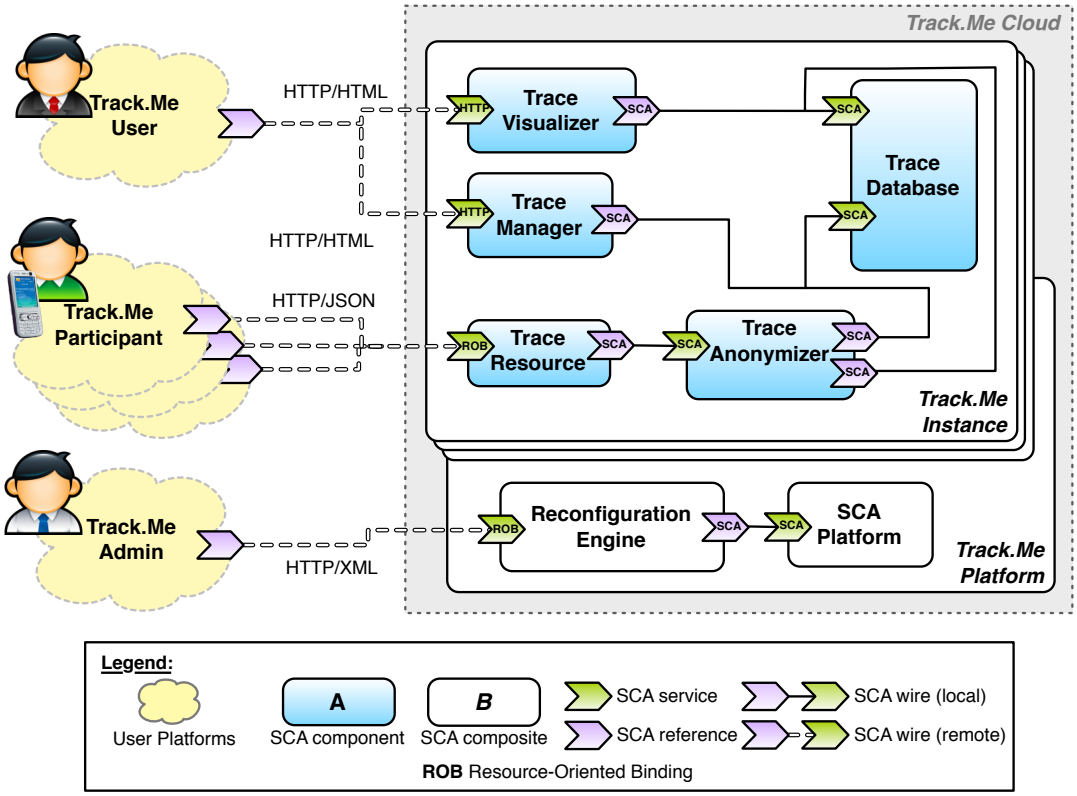


Figure 7.4: TRACK.ME Server-Side Architecture

Implementation Details

We have implemented the Activity Trace Policy in Figure 7.5 by using the various types of Android components. In particular, we define the different nodes in the trace policy as *Android services*, which can be shared at execution time by different Android applications. Besides, this choice preserves the modules defined in the different parts of the trace policy. For its part, the server-side of the TRACK.ME Platform has been developed using the FraSCAti platform version 1.2 and the resource-oriented bindings presented in Section 5.7.1.

7.2.2 Quantitative Evaluation

In order to measure the efficiency of our approach and confirm its suitability when it is incorporated into the FraSCAti platform, we execute the tests by using two configurations: *i)* local with the traces provider and TRACK.ME server running on the same laptop and, *ii)* distributed including the usage of an Android phone as provider and a laptop as server. In both sides the location information is manipulated using the GPX format and it is exchanged as JSON documents.

With the performance tests we want to confirm that the integration of SPACES Connectors concepts into FraSCAti does not impact the efficiency of the approach and we also get all the advantages associated with SCA and of course the FraSCAti platform.

Test Bed Configuration

For testing the TRACK.ME platform we employ the ADT Plugin for Eclipse 0.9.7 and BaseX 6.1 [Grün *et al.*, 2009]. In the testbed we include the Android application, the TRACK.ME server

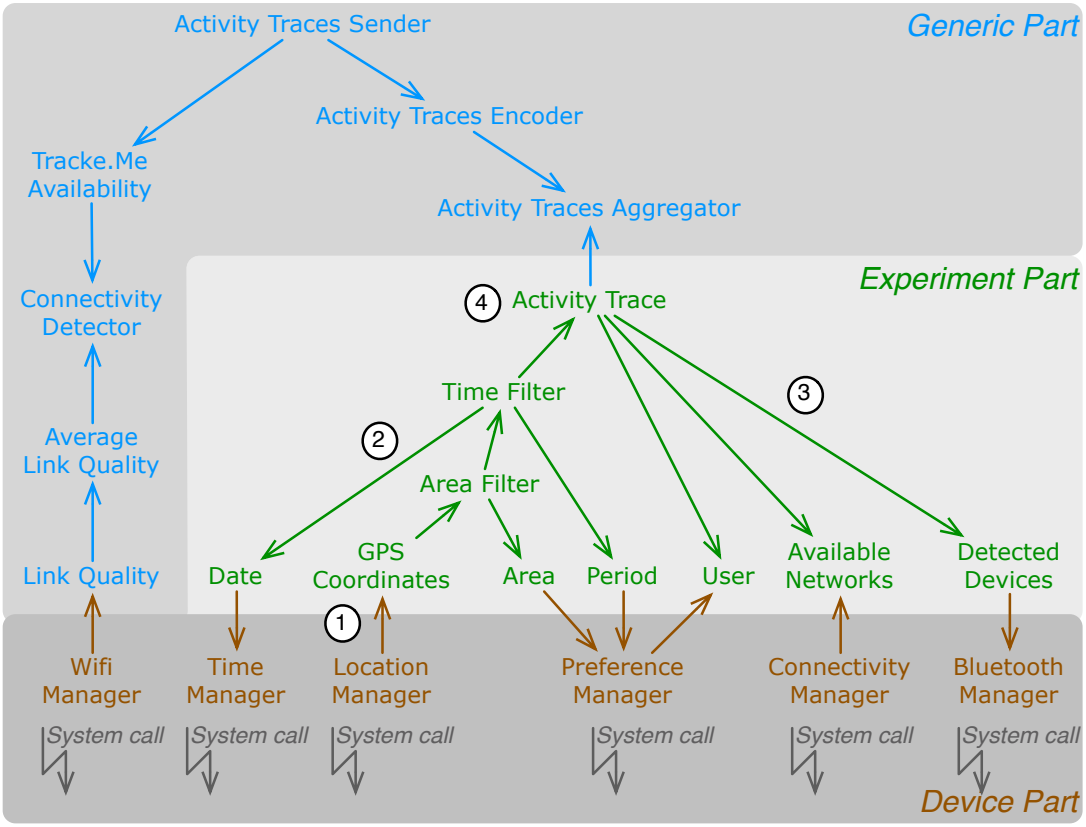


Figure 7.5: Example of an Activity Trace Policy

and the database, which run on a MacBook Pro Laptop with 2.4 GHz processor, 2 GB of RAM, AirPort Extreme card, Mac OS X 10.5.6 (kernel Darwin 9.6.0) and Java Virtual Machine 1.6.0. We have also executed some tests on a HTC Magic Android phone based on the version 1.5 of the platform (API level 3) with 528 MHz, 192 MB of RAM and interface WLAN 802.11 b/g.

Measurement Collection

In order to measure the TRACK.ME performance, we have performed 100 tests on the laptop running the client and the server. We have obtained an average cost of $772.48ms$ of which $419.2ms$ corresponds to the information processing (collection, filtering and storage) and $353.28ms$ to communication cost (using HTTP). The average size of the exchanged messages is 122 bytes, including the location information (longitude and latitude), the timestamp and the user name associated with the client application.

In a similar way, we ran the same tests using the HTC Magic Android phone to execute the TRACK.ME client application. With this configuration, we got an average cost of $793.47ms$, with $432ms$ for information processing and $361.47ms$ for communication.

7.2.3 Results Discussion

Comparing the results from the first case study including mobile devices (cf. Section 7.1), configuration *e* with the JSON representation), and the performance results with TRACK.ME we observe that we keep approximately the same efficiency of SPACES when it is integrated in FraSCAti.

Furthermore, regarding the two kinds of test developed with TRACK.ME, the usage of the mobile device introduces a minor overhead in the processing and the communication costs. Thus, the two testbeds demonstrate the suitability of the platform and of course of our approach, in particular when it is used with mobile devices.

7.2.4 Qualitative Evaluation

The *Caching Off-Loading Situation* introduced in Section 7.1 allowed us to illustrate the suitability of SPACES concepts for building an elegant and effective solution in order to integrate context information by using context policies. Although the different components of the implementation isolate the responsibilities fostering reuse, the usage of the global solution becomes impractical if we do not require to apply COSMOS as context manager. In the concrete case of TRACK.ME, the application of the COSMOS-based implementation would require the definition and usage of COSMOS nodes in both sides. However, in this platform we require a loose coupling between the client and server sides. Furthermore, we do not want to impose the usage of an specific context manager because the heterogeneity of mobile devices. Therefore, to deal with this limitation we bring the SPACES concepts into SCA.

By combining SPACES and SCA we still preserve the already mentioned advantages of the approach in terms of **reusability**, **transparent context integration** and **concerns isolation**. But we also achieve the generalization of the solution, which can be now used in the conception of different kinds of application based on SCA. The TRACK.ME platform is a suitable use case for illustrating the generalization of the approach. In the TRACK.ME implementation, we focused only on the development of the different services. The exposition of these services was transparent because we exploit the already existing resource-oriented bindings in the FraSCAti platform. Moreover, by benefiting from our approach, the TRACK.ME platform can interact with different kinds of clients, running on heterogenous devices. The fact of applying standards and a simple solution permits that client and server can interact without knowing the concrete implementation of each other.

On the other hand, the modularization of concerns makes the changes in terms of business logic and communication independent. Furthermore, the reflexion capabilities of the FraSCAti platform can be exploit to modify the application at runtime. Therefore, the so conceived solutions can be easily adapted at design as well as at runtime.

7.3 The DIGIHOME Service-Oriented Platform

In the third case study, we propose a scenario based on smart home environments. The objective of the case study is to illustrate the usage and versatility of all our proposal—*i.e.*, the Ubiquitous Feedback Control Loops. We do that by building DIGIHOME, a service-oriented platform that enables the adaptation of mobile applications for controlling home appliances as well as changes in room configuration.

7.3.1 Smart Home Scenario Description

A smart home generally refers to a house environment equipped with several types of computing entities, such as *sensors*, which collect physical information (temperature, movement detection, noise level, light, etc.), and *actuators* changing the state of the environment [Abowd *et al.*, 2003]. In this scenario, we consider a smart home equipped with occupancy, smoke detection, and temperature sensors. These tiny devices have the ability to collect context information and to communicate wirelessly with each other, in order to identify the context situation of the environment. In addition to that, we can also use actuators to physically control lights, TV, and air conditioning. Figure 7.6 illustrates the integration of these sensors and actuators in our scenario. As depicted in this Figure, the different entities use heterogeneous protocols to interact. In the scenario, the

smartphones provide information about the user preferences for the home configuration. Conflicts between the user preferences are resolved by giving priority to the person who arrived first to the room. The mobile devices also have an application, callled MOBIHOME (cf. Section 6.4.1), that enables the control of the actuators present in the different rooms. This application has several modules (one for each appliance) that are activated or deactivated according to the current *battery level*, the *battery saving* preference and *activation of modules* preference (these informations are also provided by the mobile device). The modules also are installed/uninstalled regarding the changes in the appliance configurations. Finally, there is a *Controller*, which is able to gather information, and interact with the other co-located devices.

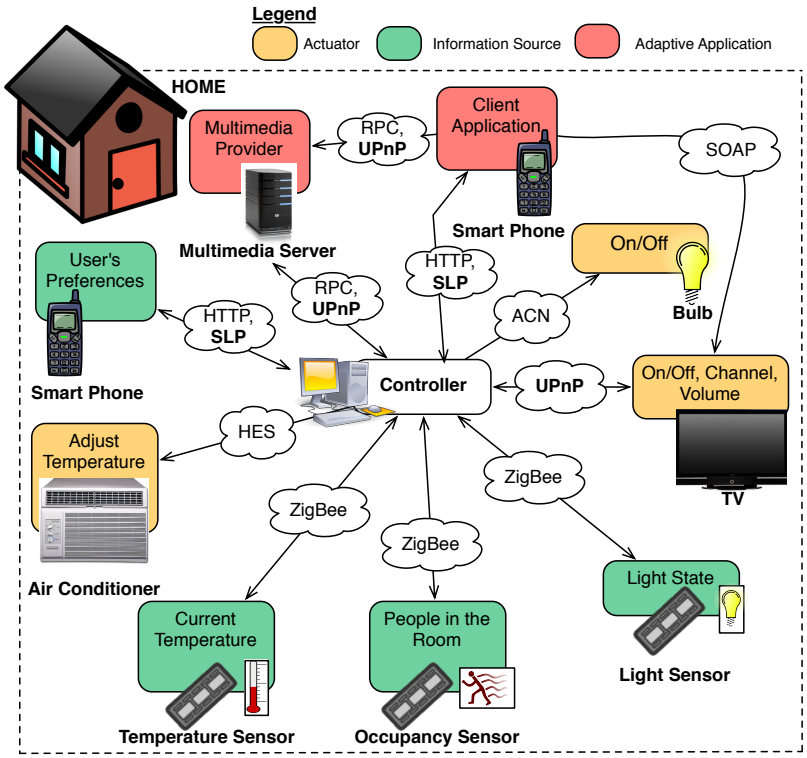


Figure 7.6: Interactions Between the Smart Home Devices.

To show how the different elements of our scenario interact, we present four different situations:

Situation 1: Alice arrives to the living room. The occupancy sensor detects her presence and also notifies the Controller that the room is occupied by somebody, which in turn tries to identify the occupant by looking for a profile in her mobile device. When Alice's profile is found, the Controller loads it and adjusts the temperature and lightening level of the room according to Alice's preferences.

Situation 2: The sensors detect smoke and notify the Controller, which using the occupancy sensor, detects that the house is empty. The Controller therefore sends an MMS to Alice, including a picture of the room captured using the surveillance camera. After checking the picture, Alice decides to remotely trigger the sprinklers using her mobile device. She also tells the system to alert the fire department about the problem. If Alice does not reply to the Controller within 5 minutes, the system activates automatically the sprinklers and alerts the fire department.

Situation 3: Alice installs a new TV in the bedroom. The Controller detects the presence of the new device, identifies it, and downloads the corresponding control software from an Internet repository. The platform tries to locate the available mobile devices, using a discovery mechanism, and finds Alice's mobile device. The Controller proposes to update the mobile device with the components for controlling the new TV.

Situation 4: Alice uses her mobile device to look for stored videos in her `Multimedia Server`. She receives a call and then she has to leave home unexpectedly. Outside, the application running on the mobile phone continues trying to access the `Multimedia Server` unsuccessfully. The mobile device has the capacity to detect this irregular behavior and stops the module in the application accessing the `Multimedia Server`.

The previous situations allow us to identify several key challenges. In particular, we can see that the ***integration of multi-scale entities*** is a key issue in the scenario. The mobile devices and sensors have different hardware and software capabilities, which make some devices more powerful than others. Therefore, the integration of these entities requires a flexible and simple solution that supports multiple interaction mechanisms and considers the restricted capabilities of some devices.

The ***mobility of entities*** represents another issue that have to be tackled. In the scenario, computational entities join and leave constantly. In particular, mobile devices providing user profiles are not always accessible (they can be turned off or the owner can leave the house with them). In a similar way, the actuators can be replaced or new ones can be added. Thus, we need to discover new entities dynamically as well as to support device disconnections.

Finally, we need to deal with the ***information processing and adaptation***. In order to support adaptation, we first need to identify the situations in which the adaptation is required. We have a lot of information that is generated by the different devices in the environment. Therefore, we need to define which part of this information is useful to identify relevant situations and react accordingly. In our scenario, those situations include the load of Alice's profile and the adjustment of the temperature, the sending of an alert via MMS in case of an emergency, the adaptation of Alice's mobile device to control the new TV in her bedroom and the absence of the Multimedia Server.

The previous issues are related to the already identify challenges in Chapter 6. Therefore, we can deal with these issues by applying the Ubiquitous FCLs. To do that we propose the DIGIHOME platform for dealing with context-based adaptation in smarthomes.

7.3.2 Platform Description

The DIGIHOME platform [Romero *et al.*, 2011, Romero *et al.*, 2010a] is a simple but efficient service-oriented middleware solution to facilitate context-awareness in ubiquitous environments. Conceived by applying our Ubiquitous Feedback Control Loops approach (cf. Chapter 6), DIGIHOME provides support for the *integration*, *information processing* and *adaptation* of the context-aware applications. This means that the platform enables the integration of heterogeneous computational entities by relying on the SCA model, REST principles and, standard discovery and communication protocols. As already stated in Chapter 6, the combination of SCA and REST fosters reuse and loose-coupling between the different services that compose the platform.

Figure 7.7 depicts the architecture of the DIGIHOME platform regarding the introduced smart home scenario. As it can be seen, we respect the structure of the Ubiquitous FCLs. In particular, the `Controller` governs the FCL execution by means of the `DigiHome Core`, which extends the `Adaptation Server` (cf. Chapter 6) with the components encapsulating the actuators control. Following a plug-in mechanism, these components grant access to the available actuator services in the environment. This means that the different actuator components are optional and deployed according to the current service configuration. For its part, `DigiHome Objects` are SCA components providing and/or consuming context information to/from others `DigiHome`

`Objects`. In our Ubiquitous FCL, a `DigiHome Object` represents an application. In our scenario, the mobile device executes a `DigiHome Object` enabling the control of home appliances (that also consumes context information indirectly in order to be adapted). The `DigiHomeME Core` is a lightweight version of the `DigiHome Core` allowing simple reconfigurations on the mobile device when the global loop is not available, as for example, in the situation 4 of the scenario. In this way DIGIHOME deals with the kind of situations presented in Section 7.3.1.

Implementation Details.

We built a prototype of the DIGIHOME platform based on FraSCAti 1.2. As already stated in Section 2.3.3, the selection of this platform is motivated by two main reasons: *i)* The platform brings reflection and reconfiguration capabilities at runtime into SOA systems and, *ii)* The FraSCAti customization capabilities according to the developer needs. The former is necessary in order to enable the dynamic adaptation of DIGIHOME applications. The latter allows us to easily deploy lightweight versions of DIGIHOME for executing them on devices with restricted capabilities, such as the mobile devices in the smarthome scenario. In order to implement the ubiquitous bindings, we have used CYBERLINK for Java version 1.7[13] for UPnP and the jSLP library[14] for SLP. Once the services are discovered, the DIGIHOME platform uses the resource-oriented bindings for interacting with them. As stated in Chapter 5, these bindings follow a RESTful approach in order to exchange information and use the HTTP protocol. Finally, in the `Decision Maker` (cf. Section 6.4.6 for a detailed description of the architecture of this component) implementation we benefit from the JaCoP (Java Constraint Programming) solver [Kuchcinski and Szymanek, 2010] version 3.1 for implement the logic associated with the selection problem. The selection of this library is motivated because of its simplicity and spread usage in the scientist community.

Social Bindings.

In addition to the bindings discussed in Chapter 5, we also introduce in SCA another kind of binding, the *Social Bindings* [Mélisson *et al.*, 2010b], in order to provide several benchmarks with DIGIHOME for validating our approach. These bindings are based on the micro-blogging services, which provide the sharing of information in real-time by reaching a lot of people in just a few seconds. One of the main advantages of these services is the short and simple nature of the posted messages. Furthermore, there are no restrictions on the subject associated with the messages. In general, this kind of informal communications was conceived for broadcasting simple events in the daily life, such as what people are doing, thinking, and experiencing [Zhao and Rosson, 2009, McFedries, 2007]. However, nowadays the micro-blogging services are also used for collaborative work in organizations [Zhao and Rosson, 2009], publicity purposes, and even for broadcasting real-time news updates for recent crisis situations. Therefore, the simplicity and flexibility as well as the real-time notification property of micro-blogs make them another suitable option to enable context exchange between the participants of the DIGIHOME platform.

The social bindings allow the notification of situations identified by the system that may require human intervention, *e.g.*, the detection of an intruder in home. To do that, the defined social bindings benefit from a simple but widely used micro-blog service: *Twitter*. Furthermore of the advantages of any micro-blogging service, the Twitter messages (so called *tweets*) can be posted using different formats (*e.g.*, JSON, XML, RSS and ATOM) and a simple RESTful API [Makice, 2009]. For purposes of our validation, we employ these bindings for exchanging the context information between the mobile device and the Controller.

[13] CYBERLINK for Java: `http://cgupnpjava.sourceforge.net/`
[14] jSLP: `http://jslp.sourceforge.net/`

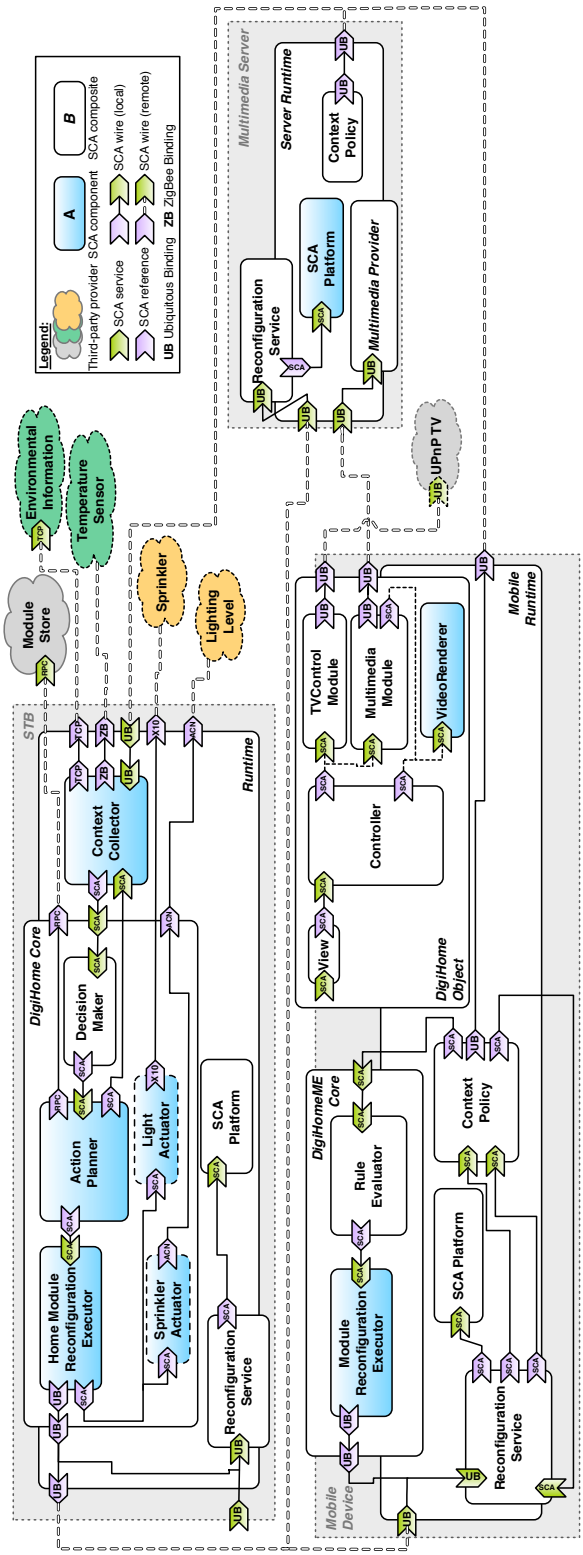


Figure 7.7: Conception of DIGIHOME as a Ubiquitous Feedback Control Loop.

Config.	Number of Providers	Retrieval Latency- Local Providers (ms)			Retrieval Latency- Remote Providers (ms)			Discovery Latency- Local Providers (ms)		Discovery Latency- Remote Providers (ms)	
		Objects	JSON	XML	Objects	JSON	XML				
a)	1	38	31	39	101	108	112	8	11	34	45
b)	2	46	45	62	160	174	179	17	22	58	75
c)	5	65	90	93	361	377	380	39	52	119	148
d)	10	145	149	309	418	425	444	77	102	238	315
e)	50	504	556	622	1954	1997	2039	361	481	1060	1410
f)	100	933	1449	1500	2269	2286	2305	730	974	2194	2912
g)	1 (N800)	N/A	N/A	N/A	340	376	N/A	N/A	N/A	129	136
Communication Mechanism		**HTTP Protocol**						**SLP**	**UPnP**	**SLP**	**UPnP**

a)

Config.	Number of Providers	Retrieval Latency (ms)	
		JSON	XML
a)	1 (Local)	948.2	951.55
b)	1 (Remote)	965.2	971.44
Communication Mechanism		**Twitter**	

b)

Table 7.2: Performance of the DIGIHOME Platform Using *a)* the Resource-Oriented and Ubiquitous Bindings and, *b)* the Social Bindings

7.3.3 Quantitative Evaluation

Table 7.2 reports the media latency for the context dissemination and discovery in the DIGIHOME platform. We have executed 10000 successful tests, of which the first 100 were considered as part of the warm-up. In this setup, we retrieve the user preferences from multiple local and distributed providers and use multiple formats for the context information (*i.e.*, XML, JSON, and Java Object Serialization). We also measured the delay for discovering the information provided by the sources. For discovery, we selected the UPnP and SLP protocols (cf. Section 2.4). In the tests, the platform aggregates the user's preferences to reduce the number of messages exchanged between the provider and the consumer. The measured time corresponds to the context policy processing, the exchange of messages as well as the marshalling/unmarshalling of the information. We have used REST messages (cf. Table 7.2 *a*) and tweets (cf. Table 7.2 *b*) for exchanging the context information. The cost of executing others protocols, such as ACN and ZigBee was not considered. More information about the overhead introduced by these protocols can be found in [Zigbee Alliance, 2007].

In order to show the overhead introduced by our Ubiquitous FCL in terms of analysis in the decision making, we separately measure the cost of this responsibility. Table 7.3 summarizes the results. We applied of course, the strategy introduced in Section 6.4. In particular, we optimized the QoS dimension. In the different tests, we varied the mandatory and optional flexibility points as well the number of QoS dimensions and required constrains to determine how they impact the performance of our solution. For the optional flexibility points we also use a different number of implementations. In the case of the mandatory flexibility points, we associated only one implementation.

Config.	Mandatory FPs	FPs	Number of Implementations by FP	Number of QoS Dimensions	Number of Required Constrains	Decision Making Latency (ms)
a)	2	3	1	5	1	0.882
b)	10	10	10	5	10	6.446
c)	2	**10**	1	5	1	0.887
d)	2	**100**	1	5	1	2.838
e)	2	**1000**	1	5	1	77.066
f)	**10**	1000	1	5	1	83.44
g)	**100**	1000	1	5	1	86.119
h)	**1000**	1000	1	5	1	268.767
i)	100	1000	1	5	**10**	90.213
j)	100	1000	1	5	**100**	98.749
k)	100	100	**10**	5	100	542.372
l)	10	10	**100**	5	10	715.837
m)	10	10	100	**10**	10	715.869
n)	10	10	100	**100**	10	724.059
o)	10	10	100	**1000**	10	725.882

Table 7.3: Performance of the Context-Based Adaptation

7.3.4 Results Discussion

Regarding the tests including REST messages (cf. Table 7.2 *a)*), we can see that there is a linear increase of the latency with the different formats. This is a good characteristics of our solution, because we can integrated several entities with an acceptable overhead. We also observe that there is not a big variation in the communication cost between the different formats when the number of providers is low (until approx. 10). As expected, when the providers are increased, the context exchange with object serialization is more efficient than the JSON and XML representations. Furthermore, the network usage introduces an overhead of approximately 300%.

In the case of the Social Bindings (cf. Table 7.2 *b*), the indirection introduced by the microblogging service increments the event exchange by approximately 30 times compared to the configuration *a* for the two mechanisms with the JSON representation. However, despite of this overhead, the simplicity of this interaction mechanism makes it still a suitable alternative to share context information in an asynchronous way. Furthermore, the context notification rests less than one seconds, which is a reasonable overhead to communicate relevant situations in smart homes environments.

On the other hand, the experiences with the mobility support (that includes the discovery time as well as the cost associated with the ubiquitous binding configuration), evidence that the discovery cost is negligible compared to the context information retrieval when there are not many providers (cf. **Table 7.2 *a)*** configurations *e)*, *f)*, *g)*, with the HTTP, SLP and UPnP protocols). Our tests show that the usage of SLP or UPnP for discovery does not have a big impact on the discovery time. In a similar way to the retrieval case, the measures including the network are bigger. Furthermore, we tested our solution using a Nokia Internet Table (N800) as a preference provider (cf **Table 7.2 a)** and **b)**, configuration *c)*). As it can be seen, the use of this mobile device introduces a considerable overhead for discovery and information exchange. This additional cost is mostly due to the limited resources of this kind of devices.

Analyzing the adaptation costs provided by Table 7.3, we see that in configuration *a)* we obtained a cost of $0.882ms$ for the MOBIHOME application and in configuration *b)*, $6.446ms$. From

these results, we observe that our approach has a good performance for simple applications having a reasonable quantity of constrains, flexibility points and implementations. On the other hand, considering more complex applications, we see that the overhead increases considerably. In particular, comparing the time for reconfiguring a simple application such as in configuration *b)* with a more complex application as in the reconfiguration *k)*, we observe that the analysis cost increases 84 times approximately. This increment is mostly due to the number of required constrains and implementations by optional points, which impose more restrictions on the constrain satisfaction problem as also confirmed by configurations *i*, *j*, *k*, *l*. However, considering that DIGIHOME is mainly focused on the adaptation for mobile applications which tend to be simple, applications with too many restrictions are difficult to find. Finally, regarding the impact of the QoS dimensions, which utility function we are optimizing, we see that they do not have a relevant impact on the analysis cost. This is confirmed by the results obtained in configurations *m*, *n* and *o*, where we only vary the quantity of QoS dimensions.

7.3.5 Qualitative Evaluation

In the previous section we have reported the efficiency of our approach—*i.e.*, the Ubiquitous FCLs, in the particular case of smart homes. Now, in this section we describe the advantage of the approach in terms of its design and usability.

A key advantage of our approach is the ***openness***. Below we analyze the Ubiquitous FCLs in terms of the *integration*, *extensibility* and *modification* dimensions stated by [Anvaari and Jansen, 2010, Sim *et al.*, 2006] to clarify the openness of our approach.

1. **Integration**: According to [Anvaari and Jansen, 2010], in the context of *openness*, the integration of a software solution refers to the "usage of its components by means of, for example, APIs, service calls and source code inclusions". Applying this notion to the Ubiquitous FCLs, we can see that different elements provided for monitoring, analysis, planning and execution can be integrated for adapting new applications. The different bindings (resource-oriented, ubiquitous, social, etc.) as well as the reconfiguration service used in DIGIHOME are generic and therefore reusable. Furthermore, the exposition of the different functionalities by means of well defined and accepted mechanisms enables other applications (that are not part of the loop) to benefit from the services offered by entities that compose the FCL.

2. **Extensibility**: This concept is associated with enhancing the functionality of the components that are part of the solution [Anvaari and Jansen, 2010]. Because we conceive the Ubiquitous FCLs by using standards and hiding the service implementation with SCA, we can easily incorporate other services that are not originally part of the infrastructure. In the DIGIHOME case, we can add new services exposed by means of standard protocols such as UPnP.

3. **Modification**: Is the replacement or change of components of the solution [Anvaari and Jansen, 2010]. The isolation of the different responsibilities of the adaptation thanks to the autonomic computing, SOA and component paradigms guarantees a loose-coupling between the different elements in the Ubiquitous FCLs architecture. Therefore, in DIGIHOME we can replace all the functionality associated with a specific phase of the feedback control loop or some parts of it.

Besides this openness,, the possibility of defining Local Feedback Control Loops provides certain ***self-healing*** (cf. Section 4.1) degree in DIGIHOME. This property is offered because simple adaptation situations can be managed even if the orchestrator of the global loop is not available. In the concrete case of the situation 4 in the scenario (cf. Section 7.3.1), in the absence of the Controller, the mobile device has still the capacity of dealing with the unavailable Multimedia Server. Furthermore, as stated in Section 6.7, depending on the defined ECA rules, the triggered reconfigurations can correct simple problems associated with the application behavior by means

of the start, stop, activation or deactivation of the application components. In the situation 4 of our scenario we stop the `Multimedia Server`, which makes that the application tries to access a service that is not more available. Therefore, the global control loops rest on the local ones when the first ones are unable to make the decisions because of communication issues.

7.4 Limitations of the Approach

In this section we present the shortcomings of our approach, which have been identify when working in the presented case studies.

Mobility Support and Adaptation

Our Ubiquitous Feedback Control Loops are conceived by applying the SPACES Connectors concepts. They provide support for the mobility of the adaptive applications as well as the services that are considered in their adaptation. In the particular case of DIGIHOME, we support the mobility of the applications running on the smartphones and services using standard discovery protocols such as the UPnP TV. However, even if we are able to identify the presence of new services in the environment and reconfigure the mobile application for using these services, we still need to develop the modules to exploit this functionality. In other words, we need beforehand to know what specific services potentially will arrive to the environment. Therefore, it will be useful a mechanism for retrieving the components that provide the client functionality for different kinds of services. This mechanism could be an SCA repository available on Internet. However the challenge with the different client components is that they should be generic enough for being reused in any kind of SCA applications that can require them.

Scope of the Local Feedback Control Loops

As already explained, the local feedback control loops in our approach have a support role inside the Ubiquitous FCLs. This decision has been mainly made for resource saving purposes on the mobile devices. This results in a limitation for the autonomy degree of the applications. To overcome this shortcoming, the local feedback control loops need to have a more participative role in the adaptation process. In particular, these local loops should take more complex decisions and be executed only if the device has enough resources (battery, memory, etc.) to do it. For its part, the global loops now will have a monitoring role of the local ones in two ways. The former includes the update of the adaptation rules on the mobile device according to the current service configuration. This means that the mobile device does not have to know explicitly all the available services. The latter refers to the observation of the local feedback control loop execution. If it is not executed, the global loop has the responsibility of adapting the application running on the device.

7.5 Summary

In this chapter, we have introduced three case studies: a ***caching or off-loading situation***, the **TRACK.ME** platform and the **DIGIHOME** platform. We use these case studies for illustrating the usage concrete of SPACES and the Ubiquitous FCLs that we build by applying its concepts. The three cases are complementary since they enable the evaluation of different aspects of the approach. Below we summarize these approaches.

1. The ***caching or off-loading situation*** represents a proof of concept of SPACES. In this situation, we define a context policy for deciding what to do with the collected information—*i.e.*, if (remotely) processing it or (locally) storing it. The context policy is distributed between

two entities, a client, providing information related to preferences, device resources and network connection, and a server making the decision. By means of the context policy distribution, we shows the *flexibility* and *simplicity* of our approach.

2. The **TRACK.ME** platform exemplifies a concrete application of SPACES requiring its generalization. With the platform we provide the possibility of defining experiences by means of traces collected from mobile devices. The tracking of activities associated with mobile users serves us like an elegant situation where the modularity and loose-coupling from the combination of SOA and CBSE as well as flexibility in terms of integration are required.

3. With the last case study, the **DIGIHOME** platform, we provide a global evaluation of the approach including the realization of Ubiquitous FCLs. This scenario integrates different situations including the processing of context information and its associated reconfigurations, heterogeneity and mobility.

Part IV

Conclusions and Perspectives

Chapter 8

Conclusions

Contents

In this concluding chapter we provide an overview of the entire dissertation, including the main contributions and the publications resulting of our work.

8.1 Summary of the Dissertation

Ubiquitous environments provide several computational resources that can be exploited in order to build adaptive applications improving the user experience. These resources include sensors, actuators, laptops, mobile devices and other electronic devices using different kinds of protocols and implementation technologies in order to offer functionalities and information associated with the environment state (so called context information). The mentioned adaptive applications, also knows as context-aware applications, benefit from the current environment configuration for changing their behavior. However, the developers of such kind of applications must face several issues before creating suitable context-aware applications, including:

- ***Heterogeneity*** in terms of devices capabilities, communication mechanisms and implementation technologies. This heterogeneity is critical because different applications select the most suitable standards according to their capabilities, making the **integration** of the different entities a real challenge.
- ***Mobility*** of devices and therefore of the services that they provide. The consideration of the issue is key in the design of context-aware applications because, by definition, these adaptive applications are potentially roaming between different environments. Furthermore, the current environment configuration can change anytime, meaning the join and leave of new services and context sources used in the adaptation process. Then, the realization of context-aware application requires to consider this variability.
- ***Distribution of Adaptation Concerns*** between several entities. This issue arises because of two reasons: *i)* the limited capabilities of some entities in the environment and *ii)* the possibility of exploiting the most powerful devices. Therefore, different participants in the adaptation process have to be integrated regarding, of course, the heterogeneity and mobility issues.

In order to face these issues and make the work of context-aware application developers easier, we propose a simple but still complete solution based on well accepted and defined technologies and standards. In particular, we propose a solution at the middleware level, called SPACES, to integrate context and adapt applications using this information. SPACES proposes a metamodel reifying the most relevant elements in context mediation, which are exploited for defining SPACES connectors. These connectors enable the integration of heterogeneous context providers and consumers in ubiquitous environments. Then, we exploit in the autonomic computing the elements and concepts that allow us to model the context-based adaption as a process where the context flow becomes a keystone. We use the different tasks identified in this process for modularizing the adaptation responsibilities in Ubiquitous Feedback Control Loops, which have an SCA-based architecture that supports the adaptation of context-aware applications. The proposed architecture fosters the reuse of their different elements as well as their customization. Finally, in the adaptation process we apply constraint programming techniques for selecting the required reconfigurations when multiple options are eligible according to the current context. This paradigm can optimize the resulting adaptation regarding dimensions that provide a better user experience. In the next section we present an overview of the contributions associated with our middleware solution.

8.2 Contributions of the Dissertation

In this dissertation we have considered the adaptation of applications as a process, where the different roles and tasks are held and executed by different entities in the environment. Regarding the **integration** of the different participants as a critical aspect before the realization of the adaptation process, we propose a first part of the contribution dealing with this aspect. In particular, our approach for context integration—*i.e.*, **SPACES**, is divided in the following contributions:

1. **SPACES Metamodel**: This metamodel introduces the concept of context as a resource (cf. Section 5.3) by reifying relevant elements and concepts in context mediation. Context Information, Context Consumer, Context Provider, Context Representations and QoC are defined as first class entities. The clear establishment of the different integration concerns composes the foundations of our approach for context mediation—*i.e.*, SPACES Connectors.

2. **SPACES Connectors**: These architectural artifacts encapsulate the distribution concerns in order to provide a clear separation between context processing and context dissemination. By following a data centric approach, the SPACES Connectors define the principles for exposing and using the context information as resources (cf. Section 5.4). To do that, we propose the usage of simple mechanisms enabling the advertisement and retrieval of the information, which are integrated in a generic architecture. Then, this architecture is integrated into the SCA component model [OASIS Open CSA, 2007] to be used in different kinds of applications, not only context-aware applications (cf. Section 5.7). The usage of standards and existing paradigms fosters simplicity, extensibility and flexibility in our approach.

By using SPACES and considering the different data exchanged by participants in the adaptation process as context information, we introduce the second part of the contribution enabling such adaptation. Specifically, this part of the contribution is composed by *Ubiquitous Feedback Control Loops* that represents an SCA-based solution for dealing with mobility of the adaptive applications. This solution is summarized below.

3. **Ubiquitous Feedback Control Loops**: We apply concepts from the autonomic computing for identifying the different tasks associated with the context-based adaptation. Then, we use the modularization and loose-coupling from the SCA component model in order to conceive Ubiquitous Feedback Control Loops (cf. Section 6.3) supporting the mobility of adaptive applications. These FCLs also allow the incorporation at runtime of new services

into the adaptation process. In the FCLs, we use context policies to recognize adaptation situations (cf. Section 6.4), which are also implemented as SCA components. The usage of an unified component model makes natural the customization, extension and reuse in the Ubiquitous FCLs. Moreover, in this approach we also provide a certain degree of autonomy in devices hosting the adaptive applications by defining Local Feedback Control Loops (cf. Section 6.7) for making simple decision (when there is some communication or latency issues associated with the global or Ubiquitous FCLs).

In the Ubiquitous FCLs we based the adaptation of applications on the retrieved context information. However, considering that multiple application configurations can be suitable for the new environment configuration, we apply constraint programming techniques in order to optimize the selected configuration (cf. Section 6.4). The selection is optimized considering the cost associated with the adaptation, the resource consumed or the QoS offered by the new application configuration. Respecting the principles of flexibility and extensibility, the Ubiquitous FCLs can be easily configured for including different optimization dimensions.

Finally, we validate our contribution by applying three case studies with different complexity levels:

a) *Caching or off-loading situation*: A simple scenario for resource saving on mobile devices. In particular, in this example we define a simple context policy requiring distribution and therefore we apply a Fractal-based implementation of the SPACES Connectors to deal with the integration concerns. This case study allow us to confirm the flexibility of simplicity of our approach;

b) TRACK.ME *Platform*: A platform helping scientists to set up tracking experiments. In this scenario, we exploit the integration of SPACES Connectors into SCA. In particular, we use the resource-oriented bindings (cf. Section 5.7.1) for integrating the collected traces. With TRACK.ME we show the versatility of our approach and its generalization by means of its incorporation into the SCA model;

c) DIGIHOME *Platform*: With this case study we provide the validation of the overall approach—*i.e.*, the *Ubiquitous Feedback Control Loops*. DIGIHOME adapts applications running on mobile devices and changes room configuration according the user preferences. The case study enable us to show the suitability of our mechanism for selecting the new configuration via constrain programming techniques.

8.3 Perspectives

In this section, we discuss some perspectives associated with this dissertation. In particular we present short and long term perspectives.

8.3.1 Short Term Perspectives

- *Increase the flexibility in terms of mobility*: In Section 7.4, we discussed the drawback of our approach associated with the mobility support. We stated that the offered discovery capabilities in ubiquitous FCLs are limited because we need to know in advance what potential services will arrive and then develop the components accessing them. Therefore, it will be useful to have a mechanism enabling the use of services that are not completely expected. For example, we can benefit from service descriptions for identifying the service type and then selecting, according this type, the component(s) allowing the access to the service. The interesting aspect to consider is the development of generic components that can be reused in different kinds of applications and situations. Then, these components will be published in a repository service. By using the SCA component model, we have several facilities to achieve this task.

- *Recovery from simple failures even if there are not defined policies associated with them*: In some cases, we can benefit from the service monitoring and the service type (obtained from service descriptions) for recovering from issues associated with service availability. In particular, we can search for equivalent services in the environment and then apply scripts enabling the service replacement. In this way, we provide a certain self-healing degree for applications.

- *Improve the exploitation of the Local Feedback Control Loops*: As stated in Section 6.7, the local FCLs only provide support when there is a problem with the global ones (*e.g.*, delay in communications or mobility of the adaptive application loosing the connection with the global loop orchestrator). Therefore, simple decisions can be only made, for example, for activating or deactivating functionalities. In order to improve our approach and regarding the increasing capabilities of mobile devices hosting the adaptive application, more responsibilities can be delegated to local loops. In particular, local FCLs can be used for deciding on the different adaptations if the device resources allow it. For its part, the global or ubiquitous loops will monitor the service configuration in the environment and consequently they will adapt the context policies of local FCLs—*i.e.*, they adapt now the local loops and not directly applications. If the device does not support the decision making because of resource restrictions, the ubiquitous FCL keeps the analysis and planning responsibilities.

- *Bring behavioral adaption support*: In our Ubiquitous FCLs, the planned actions are materialized in reconfigurations scripts (cf. Section 6.5). These scripts reify compositional adaptation [Smith, 1982] of applications. However, it would be interesting to complement the action planning in our approach for enabling behavioral adaptation [Smith, 1982] on applications. This support provides a better customization of applications regarding the user expectations as well as other context information.

- *Incorporate notions from Software Product Lines (SPLs) in Ubiquitous FCLs*: In Section 6.4, we proposed a simple modeling for representing the different configurations of applications in terms of their flexibility points as well as the relationships between such points. However, we can improve this modeling by exploiting concepts from SPLs engineering [Pohl *et al.*, 2005]. In particular, we can represent the different configurations as members of a product family, where flexibility points are associated with features and implementations of the different points with variants. The establishment of these relations enables us to benefit from the strategy proposed by [Parra, 2011] for dealing with the making decision. In this strategy are combined SPL Engineering and Aspect Oriented Modeling (AOM) principles for deciding the new configuration of the application. The changes of different variation points are modularized in aspects simplifying the realization of the adaptation. The problem with this strategy is that reconfigurations are not applied when several implementations can be chosen for the same variation point. Therefore, with our CSP mechanism for selecting the new configuration, we can complement this strategy and incorporate it in our Ubiquitous FCLs.

8.3.2 Long Term Perspectives

- *Support Adaptation of Ubiquitous FCL*: In ubiquitous environments, we are dealing with the dynamic adaptation of context-aware applications. Then, we can also consider Ubiquitous FCLs as entities requiring the adaptation according to the sense information. In particular, the resulting adaptation can become context information for adapting the ubiquitous FCL. For example, we can determine the frequency of use of different policies, implementations associated with flexibility points and communication bindings. This analysis can be used to optimize the decision making and action planning by removing unnecessary evaluations and communications in the adaptation process. Furthermore, we can benefit from the modularity of the decision making to identify cases in which we should remove, replace or add

new selection mechanisms. Thus, it would be interesting to explore the notion of *FCLs for FCLs*.

- *Support for Large-scale adaptations*: Large-scale Information Systems are difficult to adapt regarding new business conditions, such as process evolution. The usage of our Ubiquitous FCLs can be explored in order to make the adaptation of such systems easier. In particular, considering the complexity of large-scale information systems, it is possible to conceive hierarchies of FCLs that must be orchestrated in order to adapt the whole system successfully. Furthermore, the ubiquitous FCLs should provide different granularity levels for reconfigurations. In other words, it should be possible to make from high level service composition to low level resource-tuning. Finally and regarding the distributed nature of large-scale information systems, a solution for adapting such system should provide support for load balancing in terms the different FCLs responsibilities. These different concerns already start to be studied as part of the SALTY project [UNS *et al.*,].

- *Support for Security*: The realization of large-scale adaptations require the consideration of security issues. In particular, the identity of the different FCLs to be orchestrated has to be confirmed. If the systems being adapted are managed by different organizations, different authentication alternatives have to be explored such as certificated-based identifications. Even in the smart home case, the identity issue have to be considered. For example, biometric [Ratha *et al.*, 2001] and IMEI-based authentications [Europe, 2003] can be suitable in this case. Other aspect to consider is the privacy issue. In the orchestrated Ubiquitous FCLs for large-scale systems, the decision making can required the usage of sensible information. This information should not be shared or accessed by not authorized entities. In the best case, the sensible information does not need to be exchanged. However, it is necessary to very if there are situations that required such exchange. If such situations exist, the application of encryption mechanisms ensuring the information privacy has to be explored. This have to be done considering that the execution time of the large-scale adaptations can not be highly impacted.

Bibliography

[Abowd *et al.*, 2003] Gregory Abowd, Keith Edwards, and Beki Grinter. Smart homes or homes that smart? *SIGCHI Bull.: suppl. interactions*, 2003:13–13, 2003. 122

[Alarcón and Wilde, 2010] Rosa Alarcón and Erik Wilde. Restler: crawling restful services. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *WWW*, pages 1051–1052. ACM, 2010. 18, 19

[Andrieux *et al.*, 2005] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, September 2005. 56

[Antila and Mantyjarvi, 2009] Ville Antila and Jani Mantyjarvi. Distributed restful web services for mobile person-to-person collaboration. *Next Generation Mobile Applications, Services and Technologies, International Conference on*, 0:119–124, 2009. 44

[Anvaari and Jansen, 2010] Mohsen Anvaari and Slinger Jansen. Evaluating architectural openness in mobile software platforms. In *ECSA '10: Proceedings of the Fourth European Conference on Software Architecture*, pages 85–92, New York, NY, USA, 2010. ACM. 129

[Apt, 2003] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003. 6, 8, 88, 95

[Arsanjani, 2004] Ali Arsanjani. Migrating to a service-oriented architecture with web services: Beyond the hype. *Web Services, IEEE International Conference on*, 0:xxxi, 2004. 4

[Association, 2001] Software & Information Industry Association. Software as a service: Strategic backgrounder. Technical report, Software & Information Industry Association, 2001. 93

[Aurrecoechea *et al.*, 1998] Cristina Aurrecoechea, Andrew T. Campbell, and Linda Hauw. A survey of qos architectures. *Multimedia Syst.*, 6(3):138–151, 1998. 71, 72

[Beauvois *et al.*, 2007] Mikaël Beauvois, Djamel Belaïd, and Guy Bernard. A planning framework for dynamic configuration in mobile environments. In *GIIS'07: 1st International Workshop on Seamless Services Mobility (SSMO)*, 2007. 95

[Bellavista and Corradi, 2006] Paolo Bellavista and Antonio Corradi. *The Handbook of Mobile Middleware*. Auerbach Publications, Boston, MA, USA, 2006. 3

[Bellavista *et al.*, 2003] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli. Context-aware middleware for resource management in the wireless internet. *IEEE Trans. Softw. Eng.*, 29:1086–1099, December 2003. 6, 40, 117

[Ben Mokhtar *et al.*, 2006] Sonia Ben Mokhtar, Anupam Kaul, Nikolaos Georgantas, and Valérie Issarny. Efficient semantic service discovery in pervasive computing environments. In *Middleware '06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 240–259, New York, NY, USA, 2006. Springer-Verlag New York, Inc. 72

[Berberova and Bontchev, 2009] Diana Berberova and Boyan Bontchev. Design of service level agreements for software services. In *CompSysTech '09: Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, pages 1–6, New York, NY, USA, 2009. ACM. 72

[Berners-Lee *et al.*, 2005] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. http://www.ietf.org/rfc/rfc3986.txt, January 2005. 74

[Beth Schultz, 2009] Beth Schultz. Context-aware mobility: What is it and how will it change the business world? http://www.computerworld.com/s/article/9134587/Context_aware_mobility_What_is_it_and_how_will_it_change_the_business_world_, July 2009. 3

[Blair *et al.*, 2004] Gordon S. Blair, Geoff Coulson, and Paul Grace. Research directions in reflective middleware: the lancaster experience. In *ARM '04: Proceedings of the 3rd workshop on Adaptive and reflective middleware*, pages 262–267, New York, NY, USA, 2004. ACM. 37

[Booth *et al.*, 2004] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Mike Champion, Christopher Ferris, and David Orchard. Web services architecture. World Wide Web Consortium, Note NOTE-ws-arch-20040211, February 2004. 16

[Bottaro *et al.*, 2007] André Bottaro, Johann Bourcier, and Clement Escoffier. Autonomic context-aware service composition. In *4th IEEE International Conference on Pervasive Services (ICPS'07)*, Istanbul, Turkey, july 2007. 53, 58

[Bouchenak *et al.*, 2006] Sara Bouchenak, Noel De Palma, Daniel Hagimont, and Christophe Taton. Autonomic Management of Clustered Applications. In *IEEE International Conference on Cluster Computing*, Barcelona, Spain, sept 2006. 52

[Bourcier *et al.*, 2010] Johann Bourcier, Ada Diaconescu, Philippe Lalanda, and Mccann Julie. AutoHome: an Autonomic Management Framework for Pervasive Home Applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2010. 55

[Box *et al.*, 2000] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, May 2000. 76

[Box, 1998] Don Box. *Essential COM*. Addison-Wesley, 1998. 24

[Boyer *et al.*, 2005] Fabienne Boyer, Daniel Hagimont, Vivien Quema, and Jean-Bernard. Stefani. Architecture-based autonomous repair management: Application to j2ee clusters. In *ICAC'05: Proceedings of the Second International Conference on Automatic Computing*, pages 369–370, Washington, DC, USA, 2005. IEEE Computer Society. 52

[Briclet *et al.*, 2004] Frédéric Briclet, Christophe Contreras, and Philippe Merle. OpenCCM : une infrastructure à composants pour le déploiement d'applications à base de composants CORBA. In IMAG/LSR, editor, *DECOR04*, ISBN : 2-7261-1276-5, pages 101–112, 2004. 57

[Bromberg and Issarny, 2005] Yérom-David Bromberg and Valérie Issarny. Indiss: interoperable discovery system for networked services. In *Middleware '05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, pages 164–183, New York, NY, USA, 2005. Springer-Verlag New York, Inc. 42

[Bruneton *et al.*, 2006a] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36:1257–1284, September 2006. 24

[Bruneton *et al.*, 2006b] Éric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The FRACTAL component model and its support in Java. *Software: Practice and Experience – Special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12):1257–1284, August 2006. John Wiley & Sons. 4, 115

[Bu *et al.*, 2006] Yingyi Bu, Tao Gu, Xianping Tao, Jun Li, Shaxun Chen, and Jian Lu. Managing quality of context in pervasive computing. In *QSIC '06: Proceedings of the Sixth International Conference on Quality Software*, pages 193–200, Washington, DC, USA, 2006. IEEE Computer Society. 8, 34

[Buchholz *et al.*, 2003] Thomas Buchholz, Axel Küpper, and Michael Schiffers. Quality of context: What it is and why we need it. In *OVUAÕ03: Proceedings of the 10th International Workshop of the HP OpenView University Association (HPOVUA 2003), Geneva, Switzerland*, 2003. 8, 34, 66

[Capra *et al.*, 2003] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. Softw. Eng.*, 29:929–945, October 2003. 37

[Cervantes and Hall, 2004] Humberto Cervantes and Richard S. Hall. Autonomous adaptation to dynamic availability using a service-oriented component model. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 614–623, Washington, DC, USA, 2004. IEEE Computer Society. 53

[Chakraborty *et al.*, 2002] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin. GSD: A Novel Group-based Service Discovery Protocol for MANETS. In *In 4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN*, pages 140–144, 2002. 43

[Chan and Chuang, 2003] Alvin T. S. Chan and Siu-Nam Chuang. Mobipads: A reflective middleware for context-aware mobile computing. *IEEE Trans. Softw. Eng.*, 29:1072–1085, December 2003. 38

[Chan *et al.*, 2005] Ellick Chan, Jim Bresler, Jalal Al-Muhtadi, and Roy Campbell. Gaia microserver: An extendable mobile middleware platform. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 309–313, Washington, DC, USA, 2005. IEEE Computer Society. 36

[Chen and Kotz, 2000] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Dartmouth College, Hanover, NH, USA, 2000. 3

[Christensen, 2009] Jason H. Christensen. Using restful web-services and cloud computing to create next generation mobile applications. In *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 627–634, New York, NY, USA, 2009. ACM. 44, 45

[Coalition, 1999] Workflow Management Coalition. Workflow management coalition terminology glossary, 1999. 89

[Cohen *et al.*, 2004] Norman H. Cohen, James Black, Paul Castro, Barry Leiba Maria Ebling, Archan Misra, and Wolfgang Segmuller. Building context-aware applications with context weaver. Technical report, IBM Research, oct 2004. 4, 35

[Comuzzi and Pernici, 2009] Marco Comuzzi and Barbara Pernici. A framework for qos-based web service contracting. *ACM Trans. Web*, 3(3):1–52, 2009. 71, 72

[Conan *et al.*, 2007] Denis Conan, Romain Rouvoy, and Lionel Seinturier. Scalable Processing of Context Information with COSMOS. In *Proceedings of the 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'07)*, volume 4531 of *LNCS*, pages 210–224, Paphos, Cyprus, June 2007. Springer. 70, 75, 104, 112, 114

[Consortium, 1999] Salutation Consortium. Salutation architecture specification, version 2.0c. http://www.osgi.org/Specifications, 06 1999. 29

[Coulson *et al.*, 2004] Geoff Coulson, Gordon Blair, Paul Grace, Ackbar Joolia, Kevin Lee, and Jo Ueyama. A component model for building systems software. In *In Proc. IASTED Software Engineering and Applications (SEA '04)*, 2004. 43

[Coulson *et al.*, 2008] Geoff Coulson, Gordon Blair, Paul Grace, Francois Taiani, Ackbar Joolia, Kevin Lee, Jo Ueyama, and Thirunavukkarasu Sivaharan. A generic component model for building systems software. *ACM Trans. Comput. Syst.*, 26:1:1–1:42, March 2008. 4

[Coutaz *et al.*, 2005] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Commun. ACM*, 48(3):49–53, 2005. 4, 34, 35, 71

[Cowan, 2005] John Cowan. Restful web services presentation. http://home.ccil.org/~cowan/, 2005. 18

[Cremene *et al.*, 2008] Marcel Cremene, Michel Riveill, and Costin Miron. Adaptation platform for autonomic context-aware services. In *Proceedings of the 2008 IEEE International Conference on Automation, Quality and Testing, Robotics - Volume 01*, pages 298–303, Washington, DC, USA, 2008. IEEE Computer Society. 54

[Crnkovic, 2002] Ivica Crnkovic. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002. 73

[Crockford, 2006] Douglas Crockford. RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON). IETF RFC, 2006. 76

[Czerwinski *et al.*, 1999] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 24–35, New York, NY, USA, 1999. ACM. 29

[Dan *et al.*, 2003] Asit Dan, Heiko Ludwig, and Giovanni Pacifici. Web Services Differentiation with Service Level Agreements, 2003. 56

[David and Ledoux, 2005] Pierre-Charles David and Thomas Ledoux. WildCAT: a generic framework for context-aware applications. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC'05)*, pages 1–7, Grenoble, France, 2005. ACM. 75

[David *et al.*, 2008] Pierre-Charles David, Thomas Ledoux, Marc LÃ©ger, and Thierry Coupaye. FPath & FScript: Language support for navigation and reliable reconfiguration of Fractal architectures. *Annals of Telecommunications: Special Issue on Software Components – The Fractal Initiative*, 2008. 101

[David, 2005] Pierre-Charles David. *Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation*. PhD thesis, Université de Nantes / École des Mines de Nantes, July 2005. 101

[Davidyuk *et al.*, 2004] Oleg Davidyuk, Jukka Riekki, Ville-Mikko Rautio, and Junzhao Sun. Context-aware middleware for mobile multimedia applications. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 213–220, New York, NY, USA, 2004. ACM. 39, 117

[Davidyuk *et al.*, 2008] Oleg Davidyuk, Istvan Selek, Jon Imanol Duran, and Jukka Riekki. Algorithms for Composing Pervasive Applications. *International Journal of Software Engineering and Its Applications*, 2:71–94, 2008. 100

[Debaty *et al.*, 2005] Philippe Debaty, Patrick Goddi, and Alex Vorbau. Integrating the physical world with the web to enable context-enhanced mobile services. *Mob. Netw. Appl.*, 10(4):385–394, 2005. 6, 40

[Depalma *et al.*, 2008] Noel Depalma, Sara Bouchenak, Fabienne Boyer, Daniel Hagimont, Sylvain Sicard, and Christophe Taton. Jade : un environnement d'administration autonome. *Technique et Science Informatiques*, 27(9-10):1225–1252, 2008. 52

[Dey, 2001] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001. 4, 34, 91

[Dubus and Merle, 2006] Jérémy Dubus and Philippe Merle. Applying omg d&c specification and eca rules for autonomous distributed component-based systems. In *MoDELS Workshops*, pages 242–251, 2006. 57

[Entertainment Services and Technology Association (ESTA),] Entertainment Services and Technology Association (ESTA). Architecture for Control Networks (ACN). http://www.engarts.eclipse.co.uk/acn. 31

[Escoffier *et al.*, 2007] Clément Escoffier, Richard S. Hall, and Philippe Lalanda. ipojo: an extensible service-oriented component framework. In *IEEE SCC*, pages 474–481, 2007. 56

[Europe, 2003] GMS Europe. GMSE Proposals Regarding Mobile Theft and IMEI Security, june 2003. 139

[Fielding, 2000] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. 7, 9, 18

[Fillinger *et al.*, 2006] Antoine Fillinger, Stéphane Degré, Imad Hamchi, and Vincent Stanford. The nist smart data flow system ii multimodal data transport infrastructure. In *Proceedings of the 8th international conference on Multimodal interfaces*, ICMI '06, pages 128–128, New York, NY, USA, 2006. ACM. 53

[Floch *et al.*, 2006] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven. Using architecture models for runtime adaptability. *IEEE Software*, 23:62–70, 2006. 56

[Flores-Cortés *et al.*, 2006] Carlos A. Flores-Cortés, Gordon S. Blair, and Paul Grace. A multi-protocol framework for ad-hoc service discovery. In *MPAC '06: Proceedings of the 4th international workshop on Middleware for Pervasive and Ad-Hoc Computing*, page 10, New York, NY, USA, 2006. ACM. 43

[Foundation, a] The Apache Software Foundation. Apache Felix. `http://felix.apache.org`. 21

[Foundation, b] The Eclipse Foundation. Eclipse Equinox. `http://eclipse.org/equinox`. 21

[Foundation, 2010] Apache Software Foundation. Apache tuscany. `http://tuscany.apache.org/home.html`, 2010. 22

[Ganek and Corbi, 2003] Alan G. Ganek and Thomas A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003. 5, 8, 49

[Garlan *et al.*, 2000] David Garlan, Robert T. Monroe, and David Wile. Acme: architectural description of component-based systems. In Gary T. Leavens and Murali Sitaraman, editors, *Foundations of component-based systems*, pages 47–67. Cambridge University Press, New York, NY, USA, 2000. 57

[Garlan *et al.*, 2004] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37:46–54, 2004. 56

[Goland *et al.*, 1999] Yaron Y. Goland, Ting Cai, Paul Leach, and Ye Gu. Simple service discovery protocol/1.0. http://quimby.gnus.org/internet-drafts/draft-cai-ssdp-v1-03.txt, 1999. 30

[Grace *et al.*, 2005] Paul Grace, Gordon S. Blair, and Sam Samuel. A reflective framework for discovery and interaction in heterogeneous mobile environments. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(1):2–14, 2005. 43

[Gribble *et al.*, 2001] Steven D. Gribble, Matt Welsh, Rob von Behren, Eric A. Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, B. Zhao, and Robert C. Holte. The ninja architecture for robust internet-scale systems and services373423. *Comput. Netw.*, 35:473–497, March 2001. 29

[Grossman, 2009] Robert L. Grossman. The case for cloud computing. *IT Professional*, 11(2):23–27, March 2009. 10

[Group, 2006a] Object Management Group. Corba component model 4.0 specification. Specification Version 4.0, Object Management Group, April 2006. 4, 38

[Group, 2006b] Object Management Group. Corba component model 4.0 specification. Technical Report Version 4.0, Object Management Group, April 2006. 54

[Group, 2006c] Object Management Group. Deployment and Configuration of Distributed Component-based Applications Specification, Version 4.0, april 2006. 57

[Grün *et al.*, 2009] Christian Grün, Sebastian Gath, Alexander Holupirek, and Marc H. Scholl. XQuery Full Text Implementation in BaseX. In *6th International XML Database Symposium on Database and XML Technologies (XSym)*, volume 5679 of *LNCS*, pages 114–128. Springer, 2009. 120

[Gu *et al.*, 2004] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A middleware for building context-aware mobile services. In *IEEE Vehicular Technology Conference*, 2004. 34, 39, 117

[Gu *et al.*, 2005] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl.*, 28(1):1–18, 2005. 34, 39, 117

[Guttman *et al.*, 1999] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608 (Proposed Standard). http://tools.ietf.org/html/rfc2608, june 1999. 7, 29, 30, 43

[Hadley, 2006] Marc J. Hadley. Web application description language (wadl). Technical report, Sun Microsystems, Inc., Mountain View, CA, USA, 2006. 18

[Hallsteinsen *et al.*, 2004] Svein O. Hallsteinsen, Jacqueline Floch, and Erlend Stav. A middleware centric approach to building self-adapting systems. In *SEM*, pages 107–122, 2004. 56

[Hariri *et al.*, 2006a] Salim Hariri, Bithika Khargharia, Houping Chen, Jingmei Yang, Yeliang Zhang, Manish Parashar, and Hua Liu. The Autonomic Computing Paradigm. *Cluster Computing*, 9(1):5–17, 2006. 6

[Hariri *et al.*, 2006b] Salim Hariri, Bithika Khargharia, Huoping Chen, Jingmei Yang, Yeliang Zhang, Manish Parashar, and Hua Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17, 2006. 8, 50

[Hermann *et al.*, 2000] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. Deapspace: transient ad-hoc networking of pervasive devices. In *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '00, pages 133–134, Piscataway, NJ, USA, 2000. IEEE Press. 29

[Hirschfeld *et al.*, 2008] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented programming. *Journal of Object Technology, March-April 2008, ETH Zurich*, 7(3):125–151, 2008. 34

[Hu *et al.*, 2007] Xiaoming Hu, Yun Ding, Nearchos Paspallis, Pyrros Bratskas, George A Papadopoulos, Paolo Barone, and Alessandro Mamelli. A Peer-to-Peer based infrastructure for Context Distribution in Mobile and Ubiquitous Environments. In *Proceedings of 3rd International Workshop on Context-Aware Mobile Systems (CAMS'07)*, Vilamoura, Algarve, Portugal, November 2007. 41, 74

[Huebscher and McCann, 2005] C. Huebscher and A. McCann. An adaptive middleware framework for context-aware applications. *Personal Ubiquitous Comput.*, 10:12–20, December 2005. 54

[Huebscher *et al.*, 2007] Markus C. Huebscher, Julie A. McCann, and Asher Hoskins. Context as autonomic intelligence in a ubiquitous computing environment. *Int. J. Internet Protoc. Technol.*, 2:30–39, December 2007. 54, 58

[IANA, 2007] IANA. MIME Media Types. `http://www.iana.org/assignments/media-types`, March 2007. 75

[Inc., 2009] Apple Inc. Bonjour. `http://www.apple.com/support/bonjour/`, 2009. 29

[Kell, 2008] Stephen Kell. A survey of practical software adaptation techniques. *J. UCS*, 14(13):2110–2157, 2008. 4

[Kephart and Chess, 2003] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, January 2003. 8, 25, 50

[Klein *et al.*, 2008] Cornel Klein, Reiner N. Schmid, Christian Leuxner, Wassiou Sitou, and Bernd Spanfelner. A survey of context adaptation in autonomic computing. In *ICAS*, pages 106–111, 2008. 52

[Kotz and Henderson,] David Kotz and Tristan Henderson. Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD). http://crawdad.cs.dartmouth.edu. 119

[Krishnakumar and Sloman, 2002] Krish T. Krishnakumar and Morris Sloman. Constraint based network adaptation for ubiquitous applications. In *Proceedings of the 6th International Enterprise Distributed Object Computing Conference*, pages 258 – 269, Washington, DC, USA, 2002. IEEE Computer Society. 100

[Kuchcinski and Szymanek, 2010] Krzysztof Kuchcinski and Radoslaw Szymanek. Jacop - java constraint programming solver, 2010. 125

[LaMarca *et al.*, 2005] Anthony LaMarca, Jeff Hightower, Ian Smith, and Sunny Consolvo. Self-mapping in 802.11 location systems. In Michael Beigl, Stephen Intille, Jun Rekimoto, and Hideyuki Tokuda, editors, *UbiComp 2005: Ubiquitous Computing*, volume 3660 of *Lecture Notes in Computer Science*, pages 87–104. Springer Berlin / Heidelberg, 2005. 45

[Laws *et al.*, 2010] Simon Laws, Mark Combellack, Raymond Feng, Haleh Mahbod, and Simon Nash. *Tuscany in Action*. Manning Publications, 2010. 22

[Leclercq *et al.*, 2005] Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware. *IEEE Distributed Systems Online (DSO)*, 6(9):1–12, September 2005. 74

[Leclercq *et al.*, 2007] Matthieu Leclercq, Ali Erdem Özcan, Vivien Quéma, and Jean-Bernard Stefani. Supporting Heterogeneous Architecture Descriptions in an Extensible Toolset. In *ICSE'07: Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*, pages 209–219, Minneapolis, USA, May 2007. ieee. 115

[Loiret *et al.*, 2009] Frédéric Loiret, Michal Malohlava, Ales Plsek, Philippe Merle, and Lionel Seinturier. Constructing Domain-Specific Component Frameworks through Architecture Refinement. In *35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 375–382, 2009. 81

[Loiret *et al.*, 2010a] Frédéric Loiret, Romain Rouvoy, Lionel Seinturier, Daniel Romero, Kevin Sénéchal, and Ales Plsek. An Aspect-Oriented Framework for Weaving Domain-Specific Concerns into Component-Based Systems. *Journal of Universal Computer Science (J.UCS)*, 10 2010. To appear. 10

[Loiret *et al.*, 2010b] Frédéric Loiret, Lionel Seinturier, Laurence Duchien, and David Servat. A Three-Tier Approach for Composition of Real-Time Embedded Software Stacks. In *13th International SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*, LNCS, pages 37–54. Springer, June 2010. 81

[Makice, 2009] Kevin Makice. *Twitter API: Up and Running Learn How to Build Applications with the Twitter API*. O'Reilly Media, Inc., 2009. 77, 125

[Manzoor *et al.*, 2008] Atif Manzoor, Hong-Linh Truong, and Schahram Dustdar. On the evaluation of quality of context. In *Proceedings of the 3rd European Conference on Smart Sensing and Context*, EuroSSC '08, pages 140–153, Berlin, Heidelberg, 2008. Springer-Verlag. 8, 34

[Marinos *et al.*, 2010] Alexandros Marinos, Erik Wilde, and Jiannan Lu. Http database connector (hdbc): Restful access to relational databases. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 1157–1158, New York, NY, USA, 2010. ACM. 45

[Marsal, 2009] Katie Marsal. Augmented reality in iphone 3.1; new snow leopard build. http://www.appleinsider.com/articles/090724/augmented_reality_in_iphone_3_1_new_snow_leopard_build.html, 2009. 45

[Mascolo *et al.*, 2002] Cecilia Mascolo, Licia Capra, and Wolfgang Emmerich. Middleware for mobile computing (a survey). In E. Gregori, G. Anastasi, and S. Basagni, editors, *Neworking 2002 Tutorial Papers*, volume 2497 of *lncs*, pages 20–58. springer, 2002. 34

[Mattern, 2004] Friedemann Mattern. Wireless future: Ubiquitous computing. In *Proceedings of Wireless Congress 2004*, Munich, Germany, nov 2004. 34

[Mattern, 2005] Friedemann Mattern. *Ubiquitous Computing: Scenarios from an informatised world*, pages 145–163. Springer-Verlag, 2005. 34

[McFedries, 2007] Paul McFedries. Technically speaking: All a-twitter. *Spectrum, IEEE*, 44:84–84, 2007. 125

[McKinley *et al.*, 2004] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing adaptive software. *Computer*, 37:56–64, July 2004. 4

[Mélisson *et al.*, 2010a] Rémi Mélisson, Philippe Merle, Daniel Romero, Romain Rouvoy, and Lionel Seinturier. Reconfigurable Run-Time Support for Distributed Service Component Architectures. In *Automated Software Engineering, Tool Demonstration*, pages 171 – 172, Antwerp Belgique, 09 2010. 11, 23, 81

[Mélisson *et al.*, 2010b] Rémi Mélisson, Daniel Romero, Romain Rouvoy, and Lionel Seinturier. Supporting Pervasive and Social Communications with FraSCAti. In *3rd DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services*, Amsterdam Pays-Bas, 06 2010. 11, 125

[Menascé and Kephart, 2007] Daniel A. Menascé and Jeffrey O. Kephart. Guest editors' introduction: Autonomic computing. *IEEE Internet Computing*, 11(1):18–21, 2007. 8, 50

[Microsystems, 2000] Sun Microsystems. Service location protocol administration guide. http://dlc.sun.com/pdf/806-1412/806-1412.pdf, 2000. 30

[Microsystems, 2003] Sun Microsystems. Jini technology core platform specification. http://www.sun.com/jini/specs/jini1_1spec.html, 2003. 29

[Microsystems, 2005] Sun Microsystems. Java transaction api (jta). http://www.oracle.com/technetwork/java/javaee/tech/jta-138684.html, 2005. 22

[Mohyeldin *et al.*, 2005] Eiman Mohyeldin, Michael Fahrmair, Wassiou Sitou, and Bernd Spanfelner. A generic framework for context aware and adaptation behaviour of reconfigurable systems. In *16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC'05)*, 2005. 55

[Moore, 2000] Gordon E. Moore. Cramming more components onto integrated circuits. *Readings in computer architecture*, pages 56–59, 2000. 34

[Neema and Ledeczi, 2003] Sandeep Neema and Akos Ledeczi. Constraint-guided self-adaptation. In *Proceedings of the 2nd international conference on Self-adaptive software: applications*, IWSAS'01, pages 39–51, Berlin, Heidelberg, 2003. Springer-Verlag. 95, 100

[Nitu, 2009] Nitu. Configurability in saas (software as a service) applications. In *ISEC '09: Proceedings of the 2nd India software engineering conference*, pages 19–26, New York, NY, USA, 2009. ACM. 93

[Nokia, 2008] Nokia. Mobile Web Server, 2008. http://wiki.opensource.nokia.com/projects/Mobile_Web_Server. 115

[OASIS Open CSA, 2007] OASIS Open CSA. Service Component Architecture (SCA), March 2007. http://www.oasis-opencsa.org/sca. 4, 9, 136

[Open Geospatial Consortium,] Open Geospatial Consortium. OGC KML Standard - Version 2.2. http://www.opengeospatial.org/standards/kml. 118

[Open SOA, 2007a] Open SOA. *SCA Policy Framework*, March 2007. Version 1.0. 26

[Open SOA, 2007b] Open SOA. Service Component Architecture Specifications, November 2007. http://www.osoa.org/display/Main/Service+Component+Architecture+Home. 4, 9, 20, 79

[OSG, 2004] OSGi Alliance. *Listeners Considered Harmful: The Whiteboard Pattern*, August 2004. 25

[Padmanabhuni *et al.*, 2006] Srinivas Padmanabhuni, Bijoy Majumdar, Mohit Chawla, and Ujval Mysore. A constraint satisfaction approach to non-functional requirements in adaptive web services. In *Proceedings of the International Conference on Next Generation Web Services Practices*, pages 109–116, Washington, DC, USA, 2006. IEEE Computer Society. 100

[Parashar and Hariri, 2005] Manish Parashar and Salim Hariri. Autonomic Computing: An Overview. *Unconventional Programming Paradigms*, pages 257–269, 2005. 6

[Parra, 2011] Carlos Parra. *Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptation*. PhD thesis, Université Lille 1, march 2011. 138

[Pham and Gehlen, 2005] Linh Pham and Guido Gehlen. Realization and Performance Analysis of a SOAP Server for Mobile Devices. In *Proceedings of the 11th European Wireless Conference*, volume 2, pages 791–797, Nicosia, Cyprus, April 2005. VDE Verlag. 115

[Pohl *et al.*, 2005] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. 138

[Preuveneers and Berbers, 2007] Davy Preuveneers and Yolande Berbers. Architectural back-propagation support for managing ambiguous context in smart environments. In *UAHCI'07: Proceedings of the 4th international conference on Universal access in human-computer interaction: ambient interaction*, pages 178–187, Berlin, Heidelberg, 2007. Springer-Verlag. 41

[Project,] The Knopflerfish Project. Knopflerfish. http://www.knopflerfish.org. 21

[Przybilski, 2005] Michael Przybilski. Rest - representational state transfer, 2005. 16

[Ranganathan *et al.*, 2004] Anand Ranganathan, Jalal Al-Muhtadi, Shiva Chetan, Roy Campbell, and M. Dennis Mickunas. Middlewhere: a middleware for location awareness in ubiquitous computing applications. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 397–416, New York, NY, USA, 2004. Springer-Verlag New York, Inc. 6, 38

[Ratha *et al.*, 2001] Nalini K. Ratha, Jonathan H. Connell, and Ruud M. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Syst. J.*, 40:614–634, March 2001. 139

[Ratsimor *et al.*, 2002] Olga Ratsimor, Dipanjan Chakraborty, Anupam Joshi, and Timothy Finin. Allia: alliance-based service discovery for ad-hoc environments. In *Proceedings of the 2nd international workshop on Mobile commerce*, WMC '02, pages 1–9, New York, NY, USA, 2002. ACM. 43

[Razzaque *et al.*, 2006] M. A. Razzaque, Simon Dobson, and Paddy Nixon. Categorization and modeling of quality in context information. In *Proceedings of the IJCAI 2005 Workshop on AI and Autonomic Communications, 2005*, 2006. 8, 34

[Richardson and Ruby, 2007] Leonard Richardson and Sam Ruby. *Restful web services*. O'Reilly, 2007. 18

[Riva and Laitkorpi, 2009] Claudio Riva and Markku Laitkorpi. Designing web-based mobile services with rest. In Elisabetta Di Nitto and Matei Ripeanu, editors, *Service-Oriented Computing - ICSOC 2007 Workshops*, volume 4907 of *Lecture Notes in Computer Science*, pages 439–450. Springer Berlin / Heidelberg, 2009. 44

[Rodriguezi, 2006] Alex Rodriguezi. Restful web services: The basics. http://www.ibm.com/developerworks/webservices/library/ws-restful/, 2006. 18

[Román *et al.*, 2002a] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):65–67, 2002. 6

[Román *et al.*, 2002b] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, Oct–Dec 2002. 34, 36, 117

[Romero *et al.*, 2008] Daniel Romero, Carlos Parra, Lionel Seinturier, Laurence Duchien, and Rubby Casallas. An SCA-Based Middleware Platform for Mobile Devices. In *EDOC Conference EDOCW '08: Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops*, pages 393–396, Munich Allemagne, 2008. 11

[Romero *et al.*, 2010a] Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, and Frank Eliassen. RESTful Integration of Heterogeneous Devices in Pervasive Environments. In *Proceedings of the 10th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'10)*, volume 6115 of *LNCS*, pages 1–14. Springer, June 2010. 11, 66, 124

[Romero *et al.*, 2010b] Daniel Romero, Romain Rouvoy, Lionel Seinturier, and Pierre Carton. Service Discovery in Ubiquitous Feedback Control Loops. In *Proceedings of the 10th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'10)*, volume 6115 of *LNCS*, pages 113–126. Springer, june 2010. 11, 66, 81

[Romero *et al.*, 2010c] Daniel Romero, Romain Rouvoy, Lionel Seinturier, Sophie Chabridon, Denis Conan, and Nicolas Pessemier. Enabling Context-Aware Web Services: A Middleware Approach for Ubiquitous Environments. In Michael Sheng, Jian Yu, and Schahram Dustdar, editors, *Enabling Context-Aware Web Services: Methods, Architectures, and Technologies*. Chapman and Hall/CRC, 05 2010. 11, 66, 81, 85

[Romero *et al.*, 2010d] Daniel Romero, Romain Rouvoy, Lionel Seinturier, and Frédéric Loiret. Integration of Heterogeneous Context Resources in Ubiquitous Environments. In Michel Chaudron, editor, *Proceedings of the 36th EUROMICRO International Conference on Software Engineering and Advanced Applications (SEAA'10)*, pages 123–126, Lille France, 2010. ACM. 11, 66

[Romero *et al.*, 2011] Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, and Frank Eliassen. The digihome service-oriented platform. *Softw. Pract. Exper.*, 2011. To appear. 10, 124

[Romero, 2008] Daniel Romero. Context-Aware Middleware: An overview. *Paradigma*, 2, 12 2008. 11

[Rouvoy *et al.*, 2008] Romain Rouvoy, Denis Conan, and Lionel Seinturier. Software Architecture Patterns for a Context-Processing Middleware Framework. *IEEE Distributed Systems Online (DSO)*, 9(6), June 2008. 70, 104, 112

[Rouvoy *et al.*, 2009] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein O. Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software Engineering for Self-Adaptive Systems*, pages 164–182, 2009. 56

[Sailhan and Issarny, 2005] Francoise Sailhan and Valerie Issarny. Scalable service discovery for manet. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 235–244, Washington, DC, USA, 2005. IEEE Computer Society. 43

[Schilit *et al.*, 1994] Bill N. Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994. 3, 34

[SCOrWare Project, 2007] SCOrWare Project. SCA Platform Specifications - Version 1.0. `http://www.scorware.org/projects/en/deliverables`, 2007. 9, 20

[Seinturier *et al.*, 2009] Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni, and Jean-Bernard Stefani. Reconfigurable SCA Applications with the FraSCAti Platform. In *6th IEEE International Conference on Service Computing (SCC'09)*, pages 268–275, Bangalore Inde, 2009. IEEE. CORE A. Acceptance rate: 1823, 81, 101

[Seinturier *et al.*, 2011] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Softw. Pract. Exper.*, 2011. To appear. 10, 23

[Shaw and Garlan, 1996] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996. 24

[Sheikh *et al.*, 2008] Kamran Sheikh, Maarten Wegdam, and Marten van Sinderen. Quality-of-context and its use for protecting privacy in context aware systems. *Journal of Software*, 3(3):83–93, 2008. 8, 34

[SIG, 2001] Bluetooth SIG. Specification of the bluetooth system, core v1.1. http://www.bluetooth.com/dev/specifications.as, 2001. 29, 37

[Sim *et al.*, 2006] N. Sim, R. Turnbull, and M. D. Walker. Open devices – their role in supporting converged services. *BT Technology Journal*, 24(2):200–204, 2006. 129

[Sivavakeesar *et al.*, 2006] S. Sivavakeesar, O.F. Gonzalez, and G. Pavlou. Service discovery strategies in the ubiquitous communication environments. In *IEEE Communications Magazine*, volume 44(9), pages 106 – 113, Antwerp Belgium, 09 2006. 29

[Smith, 1982] Brian Cantwell Smith. *Procedural Reflection in Programming Languages*. PhD thesis, Massachusetts Institute of Technology, 1982. 138

[Smith, 1984] Brian Smith. Reflection and Semantics in Lisp. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL'84)*, pages 23–35, 1984. 24

[SOA, 2007] Open SOA. Power combination: Sca, osgi and spring, March 2007. 21

[Soldatos *et al.*, 2007] John Soldatos, Ippokratis Pandis, Kostas Stamatis, Lazaros Polymenakos, and James L. Crowley. Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services. *Comput. Commun.*, 30:577–591, February 2007. 53

[Sorensen *et al.*, 2004] Carl-Fredrik Sorensen, Maomao Wu, Thirunavukkarasu Sivaharan, Gordon S. Blair, Paul Okanda, Adrian Friday, and Hector Duran-Limon. A context-aware middleware for applications in mobile ad hoc environments. In *MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 107–110, New York, NY, USA, 2004. ACM. 37

[Sousa and Garlan, 2002] João Pedro Sousa and David Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *WICSA 3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 29–43, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V. 34, 37

[Srirama *et al.*, 2006] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mobile Web Service Provisioning. In *International Conference on Advanced International Conference on Telecommunications / Internet and Web Applications and Services*, page 120. IEEE, 2006. 115

[Stefan Sidahmed, 2010] Stefan Sidahmed. iPhone Market Share: The Rest of the Story. http://seekingalpha.com/article/187212-iphone-market-share-the-rest-of-the-story?source=email, February 2010. 3

[Stirbu, 2010] Vlad Stirbu. A restful architecture for adaptive and multi-device application sharing. In *WS-REST '10: Proceedings of the First International Workshop on RESTful Design*, pages 62–66, New York, NY, USA, 2010. ACM. 45

[Systems, 2010] Metaform Systems. Fabric3. http://www.fabric3.org/, 2010. 22

[Szyperski, 2002] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002. 4

[Taherkordi *et al.*, 2011] Amirhosein Taherkordi, Daniel Romero, Romain Rouvoy, and Frank Eliassen. Restful service development for resource-constrained environments. In *REST: From Research to Practice*. Springer, 2011. To appear. 11

[Taylor *et al.*, 2009] R. N. Taylor, Nenad Medvidovic, and Irvine E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, January 2009. 70, 73, 117

[TelecomSpace,] TelecomSpace. General Packet Radio Service. http://www.telecomspace.com/datatech-gprs.html. 37

[The Apache Software Foundation,] The Apache Software Foundation. HTTP Server Project. http://httpd.apache.org. 115

[The OSGi Alliance, 2009] The OSGi Alliance. OSGi service platform core specification, release 4.2. http://www.osgi.org/Specifications, 2009. 4, 9, 20

[Topley, 2002] Kim Topley. *J2ME in a nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002. 36

[TopoGraphix,] TopoGraphix. GPX: the GPS Exchange Format - Version 1.1. http://www.topografix.com/gpx.asp. 118

[Tyagi, 2006] Sameer Tyagi. Restful web services. http://www.oracle.com/technetwork/articles/javase/index-137171.html, 2006. 18

[Ulmer *et al.*, 2009] Cedric Ulmer, Gabriel Serme, and Yohann Bonillo. Enabling web object orientation with mobile devices. In *Mobility '09: Proceedings of the 6th International Conference on Mobile Technology, Application & Systems*, pages 1–4, New York, NY, USA, 2009. ACM. 44

[UNS *et al.*,] UNS, INRIA ADAM, LIP6/MoVe, Paris 8, EBM WebSourcing, Deveryware, MAAT, and Thales. The SALTY Project. https://salty.unice.fr/. 139

[UPnP Forum, 2008] UPnP Forum. UPnP Device Architecture 1.0. http://www.upnp.org/resources/documents.asp, april 2008. 7, 29, 30

[W3C, 2007] W3C. Soap version 1.2 part 0: Primer (second edition). online, April 2007. W3C Recommendation. 9, 16

[Waldo, 2000] Jim Waldo. *The Jini Specifications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000. 37

[Weiser, 1999] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999. 3, 34

[Weiss and Craiger, 2002] R. Jason Weiss and J. Philip Craiger. Ubiquitous computing. *Leading Edge*, 39(4), april 2002. 3, 34

[White, 2004] S.A. White. Introduction to bpmn. Technical report, Object Management Group, 2004. 91

[Wikman and Dosa, 2006] Johan Wikman and Ferenc Dosa. Providing HTTP Access to Web Servers Running on Mobile Phones, 2006. 115

[Xiao, 2008] XiPeng Xiao. *Technical, Commercial and Regulatory Challenges of QoS: An Internet Service Model Perspective*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. 95

[Yasar *et al.*, 2008] Ansar-Ul-Haque Yasar, Davy Preuveneers, and Yolande Berbers. Adaptive context mediation in dynamic and large scale vehicular networks using relevance backpropagation. In *Mobility '08: Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, pages 81:1–81:8, New York, NY, USA, 2008. ACM. 41

[Yasar *et al.*, 2010] Ansar-Ul-Haque Yasar, Yves Vanrompay, Davy Preuveneers, and Yolande Berbers. Optimizing information dissemination in large scale mobile peer-to-peer networks using context-based grouping. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1065–1071. IEEE, September 2010. 41

[Yau *et al.*, 2002] Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K. S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, 2002. 40, 117

[Yau *et al.*, 2004] Stephen S. Yau, Dazhi Huang, Haishan Gong, and Siddharth Seth. Development and runtime support for situation-aware application software in ubiquitous computing environments. In *COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference*, pages 452–457, Washington, DC, USA, 2004. IEEE Computer Society. 40, 117

[Zhao and Rosson, 2009] Dejin Zhao and Mary Beth Rosson. How and why people twitter: the role that micro-blogging plays in informal communication at work. In *GROUP '09: Proceedings of the ACM 2009 international conference on Supporting group work*, pages 243–252, New York, NY, USA, 2009. ACM. 125

[Zhu *et al.*, 2005] Fen Zhu, Matt W. Mutka, and Lionel M. Ni. Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, 4(4):81–90, 2005. 4, 29, 76

[Zigbee Alliance, 2007] Zigbee Alliance. ZigBee and Wireless Radio Frequency Coexistence. `http://www.zigbee.org/imwp/download.asp?ContentID=11745`, June 2007. 127