QoS-aware Service-Oriented Middleware for Pervasive Environments

Nebil Ben Mabrouk

▶ **To cite this version:**

Nebil Ben Mabrouk. QoS-aware Service-Oriented Middleware for Pervasive Environments. Mobile Computing. Université Pierre et Marie Curie - Paris VI, 2012. English. NNT : . tel-00786466

HAL Id: tel-00786466

https://theses.hal.science/tel-00786466

Submitted on 8 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour l'obtention du grade de

DOCTEUR DE L'UNIVERSITÉ PARIS 6

École doctorale informatique, télécommunications, électronique

Mention: Informatique

Équipe d'accueil: INRIA - Projet ARLES

Présentée par

Nebil BEN MABROUK

QoS-aware Service-Oriented Middleware for Pervasive Environments

Soutenue le xx Mars 2012

Devant la Commission d'Examen

COMPOSITION DU JURY

Daniela GRIGORI	Université Paris Dauphine, France	Rapporteur
Farouk TOUMANI	Université Blaise Pascal, France	Rapporteur
Jacques MALENFANT	Université Pierre et Marie Curie, France	Examinateur
Nicolas RIVIERRE	Orange Labs, France	Examinateur
Valérie ISSARNY	INRIA Paris-Rocquencourt, France	Directeur de thèse
Nikolaos GEORGANTAS	INRIA Paris-Rocquencourt, France	Co-Directeur de thèse

Abstract

Pervasive computing is an intuitive evolution of computing paradigms driven by the wide adoption of mobile devices and wireless networks. It introduces a novel way to support users in their everyday life based on open and dynamic environments populated with unobtrusive services able to perform user tasks on the fly.

Nevertheless, supporting user tasks from a functional point of view is not enough to gain the user's satisfaction. Users instead require that their tasks meet a certain Quality of Service (QoS) level. QoS is indeed an inherent and primary requisite of users going along with their required tasks.

In the context of pervasive environments, fulfilling user tasks while delivering satisfactory QoS brings about several challenges that are mainly due to the openness, dynamics, and limited underlying resources of these environments. These challenges are mainly about (i) the lack of common QoS understanding among users and service providers, (ii) determining and integrating, on the fly, the services available in the environment and able to fulfill the functional and QoS requirements of users, and (iii) adapting the provided services at run-time to cope with QoS fluctuations and ensure meeting user requirements.

To cope with the aforementioned issues, we opt for a middleware-based solution. Middleware represents indeed the appropriate software system to deal with common concerns of user applications such as QoS. In particular, we opt for a specific kind of middleware, viz., Service Oriented Middleware (SOM). SOM can leverage middleware technologies and the Service Oriented Computing (SOC) paradigm to enable pervasive environments as dynamic service environments. Particularly, SOM can provide middleware services that allow for supporting QoS of user applications offered by pervasive environments.

This thesis presents a QoS-aware service-oriented middleware for pervasive environments.

The main contributions of this middleware are : (1) a semantic end-to-end QoS model that enables shared understanding of QoS in pervasive environments, (2) an efficient QoS-aware service composition approach allowing to build service compositions able to fulfill the user functional and QoS requirements, and (3) a QoS-driven adaptation approach to cope with QoS fluctuations during the execution of service compositions.

The proposed contributions are implemented within a middleware platform called QASOM and their efficiency is validated based on experimental results.

Table of Contents

List of Figures

List of Tables

Notations	Meaning
A_i	Abstract activity number i in the user task
$\mathcal{C}_v$	Concrete service composition number v fulfilling the user request
$cl_{j,r}$	Cluster of services of rank r for the QoS property p_j
$\mathcal{D}$	N-dimensional Euclidean distance
$\mathcal{F}_{\mathcal{C}_v}$	QoS utililty of the composition $\mathcal{C}_v$
$f_{s_{i,k}}$	QoS utililty of the service $s_{i,k}$
P	Set of QoS properties required by the user
p_j	QoS property number j
Q_j	Value of QoS property p_j of the composition $\mathcal{C}_v$
q_j	Value of QoS property p_j of the service $s_{i,k}$
$QC_{r,e}$	QoS Class with e QoS values in the QoS Level QL_r
QL_r	QoS Level of rank r
$QoS_{\mathcal{C}_v}$	QoS vector of the composition $\mathcal{C}_v$
$QoS_{s_{i,k}}$	QoS vector of the service $s_{i,k}$
$\mathcal{R}$	User request
S	Set of service candidates associated with the abstract activity A_i
$s_{i,k}$	Service candidate number k of the activity A_i
T	User required task
U	Set of global QoS constraints required by the user
u_i	Global QoS constraint number i
W	Set of weights defining user preferences for QoS properties
w_i	User preference for QoS property p_i
Δ	Computational complexity of K-means

Table 1 – Table of symbols sorted with respect to the alphabetic order

Chapter I

Introduction

1 Pervasive Computing

Pervasive computing [Bacon, 2002] is the intuitive evolution of computing paradigms driven by the wide adoption of mobile devices and wireless networks. It introduces a novel way to support users in their daily life based on open and dynamic environments populated with unobtrusive services able to fulfill user tasks on the fly.

Pervasive computing assumes the presence of networked software and hardware resources in the user surroundings offering various services to users, thus forming functionally-rich environments in which users are able to carry out their daily tasks on-the-fly in a dynamic way, i.e., without prior knowledge about available services.

Nevertheless, fulfilling the user's desired tasks from a functional point of view is not enough to gain the user's satisfaction. Indeed, along with the functional requirements associated with tasks, users have non-functional requirements which determine the degree of quality according to which tasks are fulfilled. For instance, users are generally interested in fulfilling their tasks using services with short response time and high availability. The response time and availability represent non-functional aspects of services.

In computer science, and in particular in pervasive computing, non-functional aspects of computer systems (including pervasive systems) are referred to as Quality of Service (QoS). QoS is a broad research topic, which is widely addressed in the literature. In the context of pervasive computing, several research efforts have been put forward to deal with emerging challenges related to QoS.

Indeed, pervasive computing presents three key features bringing about QoS challenges. First, pervasive computing represents a shift from static to dynamic environments in the sense that the resources and services available in pervasive environments vary continuously and there is no prior knowledge about their availability, whereas in static environments services offered

to users are known and fixed in advance. Second, pervasive computing shifts the emphasis from stationary environments with fixed infrastructures to *ad hoc* infrastructure-less environments formed of mobile devices connected *via* wireless networks. Third, pervasive computing puts a special emphasis on the use of resource-constrained devices (e.g., mobile phones) to fulfill user tasks, which is different from, traditional computing environments relying on resource-rich devices.

The shift from static, stationary and resource-rich environments to dynamic, *ad hoc* and resource-constrained environments raises several challenges about the level of QoS provided to users. To illustrate these challenges and explain the kind of situations that our research focuses on, we present the following scenarios:

"*This week, Bob's diary contains three main things to do. On Thursday, Bob will go to the hospital for a medical visit, on Friday, he will go to a commercial center for shopping, and for the weekend he will go to a holiday camp. Below, we use these activities to show how pervasive computing makes people's life easier while we emphasize at the same time research issues related to QoS.*

Pervasive Medical Visit *When he goes to the hospital, Bob has to move between different places in the hospital to fulfill the activities entailed by the medical visit such as registration, doctor diagnosis, getting medicines at the pharmacy and payment, which represents a long and hard process especially for patients.*

A second issue about the visit concerns the assignment of patients to doctors. Generally, Bob knows a priori the doctor who will examinee him. Nevertheless, the availability of this doctor can change with respect to many conditions. For instance, his doctor may be absent or too buzy with new visits (e.g., due to some emergency cases unforeseen during the scheduling of visits). In such cases, Bob has to ask for an assignment to another doctor.

To avoid such complicated situations and spare Bob from unnecessarily moving between different places, medical visits in hospitals should be managed in a novel way. Let us imagine the case where the hospital is considered as a pervasive environment offering various services (e.g., registration, diagnosis, providing medicines and payment) to patients who are generally holding mobile devices. Using his mobile phone, Bob can plan his medical visit while staying in the waiting room of the hospital.

To do so, he submits a medical visit request to the hospital information system via the wireless connectivity offered by the hospital. The information system takes in charge the registration of Bob given his medical information and returns the time and the cabinet of the visit. If the doc-

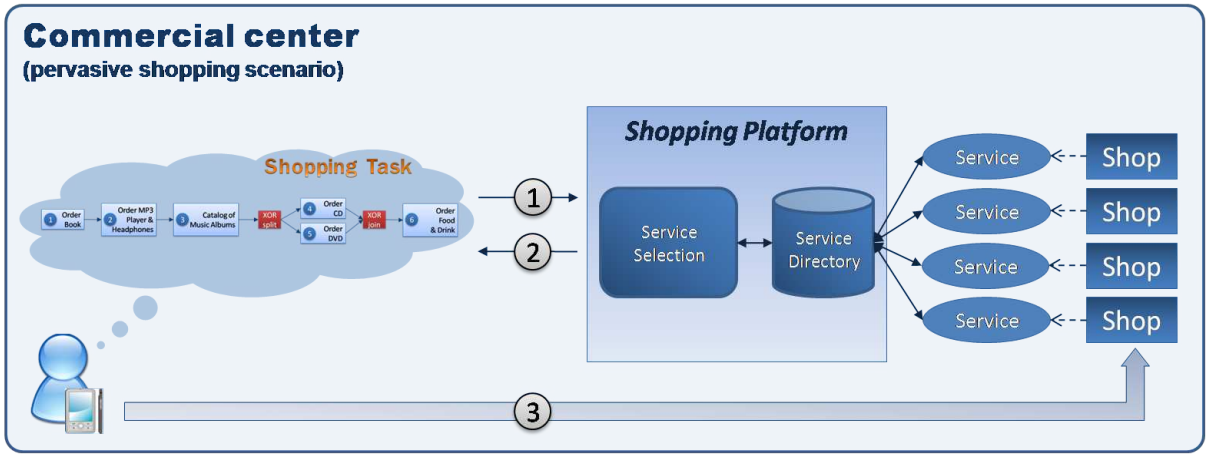

Figure I.1 – Pervasive shopping scenario

tor's availability changes in-between, the system assigns Bob dynamically to another available doctor having the same specialty and informs Bob about the changes.

When the doctor examination is finished, the system automatically orders the medicines subscribed by the doctor from the pharmacy of the hospital and informs Bob about the desk to get these medicines. Bob can also pay for the whole set of services using his mobile phone and credit card information.

Given the large number of patients, the hospital offers several services for each activity in the medical visit. For instance, it provides several registration services operating simultaneously. To offer a better medical visit to Bob, the information system attempts to select services with the highest QoS such as the registration service with the shortest response time and highest availability.

Pervasive Shopping *The next day, Bob goes to a commercial center to buy a set of items. Bob likes shopping since he considers it as an entertaining activity. However, Bob has to move between several shops in order to determine shops selling the desired items with respect to the total budget devoted to shopping. Visiting the plenty of shops in the commercial center represents a fastidious task for Bob.*

To cope with this issue, we propose a pervasive shopping scenario where commercial centers are managed as pervasive environments and customers equipped with mobile devices can fulfill their shopping in an easy way. According to our scenario, Bob can order his desired items using his mobile phone while staying in the lounge hall of the commercial center.

To do so, Bob submits a request to a shopping platform offered by the commercial center. The request comprehends two main elements (see Figure I.1) : (i) a composition of abstract

shopping tasks (e.g., buying a book, and a DVD or a CD,..) providing the set of items to buy and their descriptions, and (ii) Bob's QoS requirements such as the total price of the items and the availability of shopping services.

Based on Bob's request, the shopping platform proceeds to the selection of shopping services in the commercial center offering the required items and meeting Bob's QoS requirements. All services are published within a directory in the platform. Each service advertises the items to sell, their features, their prices as well as its QoS capabilities.

The shopping platform proposes to Bob several compositions of shopping services meeting his requirements. The proposed compositions are ranked according to their QoS. Bob has then to choose a composition of shopping services, to purchase and pay the items, and to go to the chosen shops in order to get these items.

To make our scenario even more dynamic, we further assume that Bob can fulfill his shopping task either in a commercial center or in an open-air market. The latter environment does not provide a shopping platform, it is rather completely formed of mobile and resource-constrained devices. In this environment vendors use their devices to advertise their services, whereas users fulfill their shopping using also their mobile devices. The fact of carrying out the shopping task in an ad hoc environment brings about further challenges related to QoS.

Pervasive Entertaining *For the weekend, Bob goes to a holiday camp. During his stay, he wants to listen to some recent music, which may be offered by people staying in the same camp. Related to this, we consider the holiday camp as a pervasive environment offering entertaining services provided by people in the camp using their mobile devices.*

Bob uses his mobile phone to submit a request, which comprehends the following services: First, he wants to get the 'Top 10' list of songs of the month, then he wants to look for an audio streaming or video streaming service for the first song in the list.

People staying in the camp can answer Bob's request if they provide such services. When several providers exist for each kind of services. Bob's device should select those services offering the highest QoS. For instance, it should select 'Top 10' services with the shortest response time and audio or video streaming services with the highest encoding quality.

Bob wants to listen to music or watch a video clip while moving in the camp, which may decline the quality of the audio or video stream. In this case, Bob's device should select another audio or video streaming service with a higher QoS.

The aforementioned scenarios highlight the importance of pervasive computing environments in people's life. Nevertheless, without providing a satisfactory QoS to users, pervasive computing looses much of its interest, hence the motivation of our research, which aims at es-

tablishing a QoS-aware middleware solution allowing to select, compose and adapt, on-the-fly, services available in pervasive environments and able to fulfill users tasks while meeting their QoS requirements. Next, we first introduce the concepts of QoS and QoS awareness. Then, based on the presented scenarios, we study QoS related issues in pervasive environments and delimit the scope of our research.

2 QoS and QoS-awareness

The concept of QoS emerged in the late 70' in the telecommunication and networking field to cope with the decline of quality in traditional networks that operate on a best effort basis, that is, networks do not give any guarantee about the quality of the communication. The concept of QoS has then been about ensuring a certain quality level for communication. Subsequently, QoS has increasingly gained interest and its use has been extended to cover several domains.

In computer science, in particular in pervasive computing, the term *Quality of Service* is used to refer to the quality of a computer system (including a pervasive system). Despite the broad literature about QoS, there is no agreed definition of this term. One of the earliest definitions of QoS that is also the commonly used one states that QoS is a *collective effect of service performance, which determine the degree of satisfaction of a user of the service* [ITU-T Rec., 1993]. Here, the reader can notice how broad this definition is. Indeed, it gives no tangible specification elements of the QoS concept. Similarly, most of QoS definitions proposed in the literature are less than tangible [Crawley et al., 1998; ISO, 1994].

This is due to a number of factors that are mainly about using QoS for a wide range of application domains and different kinds of systems. Indeed, each domain and system has its specific QoS vision and semantics, hence the heterogeneity of QoS definitions.

In this thesis, we deem that it is awkward to give a prescriptive and valuable definition of QoS due to the aforementioned reasons. However, to delimit the QoS concept, we deem that it is more important to establish a concrete model for QoS that enables the effective understanding of this concept. The required model should allow addressing QoS with respect to a specific domain (e.g., pervasive computing in our case) and not in a general manner.

Once a QoS model is established, it is important to define the notion of QoS awareness which deals with delivering satisfactory QoS to users using the established model as a basis for QoS understanding in pervasive environments.

The notion of QoS-awareness is seldom defined in the literature. Wac [Wac, 2005] states that QoS awareness is the fact of being aware of the user required QoS and of the QoS offered

by various resources at the user's location and time. Whereas Bellavista et al. [Bellavista et al., 2007] define QoS-awareness as the capability of having full visibility of QoS variations, thus enabling to perform adaptation countermeasures. A third definition is given by Wichadakul et al. [Wichadakul et al., 2001]. The authors claim that an application is QoS-aware if the application developer deploys QoS assurance services needed by the application to deliver the required QoS.

The above definitions are heterogeneous and they consider QoS awareness from different and complementary points of view. The first definition focuses on the fact of knowing and expressing the required and the offered QoS of a system. The second definition considers the dynamic aspect of QoS, which is about QoS fluctuations and the adaptation actions taken accordingly. Whereas the third definition concentrates on using QoS-supporting services and mechanisms to provide the intended QoS of the system. To intermingle the above aspects of QoS awareness, we provide the following definition:

QoS awareness is the fact of (i) being aware of the QoS required by actors (e.g., users, systems) operating within a computing environment (notably a pervasive environment), of the QoS offered by resources available in this environment, and of the variations of the required and offered QoS in time, as well as (ii) performing the set of actions leveraging the offered QoS to meet the required one and eventually, improve it.

This definition is quite comprehensive and makes it easier to delimit the notion of QoS awareness in pervasive computing environments. If a pervasive system has continuous knowledge about QoS requirements of the user and the QoS offered by services available in the environment, and if the system performs the set of actions allowing to meet the user's required QoS and eventually improve it, then this system is considered as QoS-aware.

3 QoS Issues in Pervasive Environments

In this section, we aim at determining key research issues related to QoS awareness in pervasive environments. Based on the application scenarios described in Section 1, we roughly distinguish four main QoS awareness issues in pervasive environments: (i) establishing QoS-enabling specifications (i.e., specifications enabling QoS awareness in pervasive environments), (ii) QoS-aware service discovery, (iii) QoS-aware service composition and (iv) QoS-driven composition adaptation.

3.1 QoS-enabling specifications

A first step towards QoS awareness in pervasive environments consists in establishing QoS-enabling specifications. The way QoS is specified has indeed a substantial impact on QoS provision in such environments. In order to address QoS specification, two major steps are required. First, defining a QoS model that enables the effective understanding and use of QoS. Second, using the specified model to describe QoS-aware services.

- *QoS modelling.* QoS modelling provides the appropriate ground for QoS provisioning in pervasive environments. It consists in identifying and formalizing QoS-related concepts (such as QoS properties, their classification and their associated metrics) as well as the relations among them. A key issue towards QoS modelling is the heterogeneity of QoS vocabulary and semantics in pervasive computing environments. Indeed, users and service providers operating in such environments use different QoS models. To cope with this issue and enable interoperability between these actors, QoS models should be specified in a way that enables a common understanding of QoS.
- *QoS-aware service specification.* Assuming a QoS model is established, a second issue to be addressed is how to use this model for specifying QoS-aware services. This concerns integrating QoS specifications within services descriptions. QoS-aware service specification should be addressed at two levels: first specifying the user task and its associated QoS requirements ; second, specifying concrete services available in pervasive environments and their QoS descriptions.

3.2 QoS-aware service discovery

As illustrated by the scenarios of Section 1, users do not have prior knowledge about the kind of services available in pervasive environments. To cope with this issue, a QoS-aware service discovery phase is required to determine the set of services able to fulfill the task and QoS requirements of the user. QoS-aware service discovery involves matching the functionalities and QoS required by users to the functionalities and QoS offered by services available in the environment. The way the matching is carried out deeply impacts the spectrum of services proposed to users, notably the number of provisioned services and the extent to which they fit the functionalities and QoS required by users.

3.3 QoS-aware service composition

In the scenarios presented in Section 1, we can clearly notice that users are generally demanding in the sense that they ask for complex tasks that cannot be fulfilled by a unique service, but rather by the composition of multiple services. Additionally, the devices operating in pervasive environments are generally resource-constrained. Thus, the services hosted by such devices are basic, in the sense that they offer simple functionalities and do not require a lot of resources. Therefore, it is hard to assume that a complex task required by users can be fulfilled by a unique service. Rather complex tasks are to be realized by the composition of several basic services offered by different providers in the environment.

Therefore, QoS-aware service composition is required to compose services able to fulfill the user task while considering their QoS aspects. Given the possibility to discover several services that are functionally equivalent, a primary issue towards QoS-aware service composition is to determine the set of services able to meet QoS requirements and provide the highest QoS to the user.

To address this issue, QoS-aware service selection algorithms are required. QoS-aware service selection can be carried out in two ways. One is the greedy way, which consists in choosing the service with the highest QoS for each functionality in the user task. This solution has a low computational cost but it does not guarantee providing the composition with the highest QoS. A solution with a higher QoS may be found by applying global QoS-aware selection algorithms (selection covering the scope of the whole composition), which is known to be NP-hard [Ben Mabrouk et al., 2009]. In the context of pervasive computing environments, global QoS-aware selection is even more challenging since it should be fulfilled on-the-fly, hence the time available for services' selection and composition is limited with regard to the computational complexity of the problem.

3.4 QoS-driven composition adaptation

QoS-aware service composition is generally carried out based on QoS advertised by service providers in pervasive environments. However, when executing service compositions, the effective QoS provided by their constituting services may fluctuate compared with the advertised one. This is due to many reasons, including services' failure and the dynamics of pervasive environments such as user mobility or the decline of wireless connectivity. To cope with these issues, service compositions have to be adapted dynamically at run-time with respect to QoS fluctuations, to which we refer to as QoS-driven composition adaptation.

QoS-driven composition adaptation requires a preliminary step of QoS monitoring in order

to determine the effective QoS provided by services at run-time. Based on the monitored QoS, QoS-driven adaptation can be carried out. It roughly consists in defining a set of adaptation actions and a set of rules controlling these actions. Adaptation actions induce changes on service compositions, which affect such aspects like their behaviours or their constituting services. The main issue towards QoS-driven adaptation is about applying these changes dynamically without interrupting the execution of running compositions. That is, adaptation actions should be applied in a flexible manner, in order to alter service compositions without inducing major impacts on their executions.

To cope with the above QoS issues in pervasive environments, next we introduce a QoS-aware service-oriented middleware solution for pervasive environments.

4 Towards QoS-aware Service-Oriented Middleware for Pervasive Environments

To address the aforementioned QoS research issues in pervasive environments, there is a number of questions that should be investigated. These questions are mainly about computing paradigms and methods that are appropriate for designing and building a QoS-aware software solution leveraging services available in pervasive environments to meet QoS requirements and to provide the highest QoS to users.

A first question towards this purpose concerns the software engineering paradigm to adopt in order to realize the required system. When we study the evolution of QoS-aware software systems, we can notice that the proposed solutions evolve along with the advancement of software engineering paradigms.

Early solutions proposed in this context are based on the Object Oriented Programming (OOP) paradigm. They are designed and built as distributed object-oriented middleware platforms, which aim at providing QoS management support for distributed object-oriented applications. TAO [Schmidt et al., 1997], QuO [Vanegas et al., 1998], the Lancaster's Adapt project [Coulson et al., 1999] and the EPIQ project of the University of Illinois [Shankar et al., 1999] fall in this category.

Later, with the emergence of the component-based software engineering (CBSE), QoS-aware software systems shift to component-based technologies. Several QoS-aware platforms have been put forward in this context. $2K^{Q+}$ [Nahrstedt et al., 2001; Wichadakul et al., 2001], CIAO [Balasubramanian et al., 2003], QuAMobile [Amundsen and Eliassen, 2006] and the TAPAS project [Dept and Lodi, 2002] are some examples of research efforts proposed in this context.

Software engineering paradigms continue to evolve to cope with issues brought about in CBSE such as the heterogeneity of component technologies and the tight coupling between components, which led to the emergence of the Service Oriented Computing (SOC) paradigm [Koskela et al., 2003]. SOC introduces the concept of *service* as a fundamental building block of distributed applications. Services extend the characteristics of components with important features such as standard interface description independent from implementation technology, loose coupling, autonomy, openness and contracting ability between services.

SOC introduces the Service Oriented Architecture (SOA), which represents a concrete architectural view of SOC. SOA defines an interaction pattern based on three main entities: (i) service consumer, (ii) service provider, and (iii) service repository that provides facilities to discover services and acquire information needed for their usage. These entities interact with each other using a set of standard specifications and protocols ; in the Web Services, one of the dominant SOA technologies, these are WSDL, SOAP, and UDDI.

Based on the above interaction pattern, SOA advocates interaction between services without prior knowledge of services' specifications. In particular, SOA puts special emphasis on the composition of independently built services in order to form coarse-grained services enabling users to carry out more complex tasks [Koskela et al., 2003].

SOA is further important if we consider it from a QoS point of view. Indeed, a lot of research efforts have been devoted to QoS-related issues in the context of SOA [Hilari, 2009]. These efforts yield a set of QoS standard specifications and QoS-supporting methods and techniques that can be investigated to address QoS concerns in the context of pervasive computing [Issarny et al., 2011].

Based on the chronological evolution of software engineering paradigms, and given the aforementioned features of SOC (notably SOA), we believe that SOC is the appropriate software engineering paradigm to deal with QoS awareness issues in pervasive environments, namely QoS-aware service specification, discovery, composition and adaptation.

Dealing with QoS awareness issues can be carried out at different levels of networked software systems. Indeed, networked software systems can be divided into three broad classes : (i) Applications, (ii) Middleware, and (iii) Operating Systems (OS). Applications are those software systems realizing the functionalities required by the user. OS are software systems needed to interact with the hardware infrastructure underlying computer systems and networks. Middleware is the broad class of software lying in-between Applications and Operating Systems. Middleware represents a broad concept, which is used for multiple purposes such as (to name a few) hiding the complexity of OS implementations, coping with the heterogeneity of OS technologies, enabling interoperability between distributed user applications, and imple-

menting reusable services to support user applications. Focusing on the latter role, middleware offers unique reusability advantages through the provision of functionalities addressing common aspects of user applications such as QoS. Given the above reasons, we deem that middleware is the most appropriate software system to deal with QoS awareness issues in pervasive environments. As we opt for a service oriented solution, we employ Service Oriented Middleware (SOM) to address QoS awareness. SOM leverages middleware technologies and the Service Oriented Computing (SOC) paradigm to enable pervasive environments as dynamic service environments, and to deal with pervasive computing challenges, notably QoS awareness. Hence, our solution is implemented as a QoS-aware service-oriented middleware for pervasive environments.

Nevertheless, our middleware puts special emphasis on the fact that pervasive computing has unique characteristics in addition to typical service oriented computing due to the dynamics and limited resources of pervasive environments [Issarny et al., 2011] as discussed in detail in Section 1. These characteristics call for advanced QoS awareness solutions to be provided by SOM for pervasive computing.

With this in mind, we aim at establishing a QoS-aware service-oriented middleware that is specifically designed for pervasive computing environments. Our ambition is to provide novel QoS awareness solutions fitting the specific characteristics of pervasive environments. Next, we give an overview of our middleware while highlighting its contributions, then we detail the structure of this document.

5 Thesis Contribution and Document Structure

In this thesis, we introduce a QoS-aware service-oriented middleware for pervasive environments. As depicted in Figure I.2, our middleware addresses QoS-aware service specification, composition and adaptation in pervasive environments. However, we do not deal with QoS-aware service discovery for two main reasons. First, to limit the scope of our research. Second, because we deem that significant efforts have been put forward in this context (e.g., [Ben Mokhtar, 2007]).

Our solution starts from the assumption that users in pervasive environments submit requests to our QoS-aware service-oriented middleware (supposed to be installed on their mobile devices) in order to accomplish their desired tasks. User requests comprehend two main parts:

1. Functional requirements, which describe the task required by the user (e.g., the shopping task with respect to our scenario presented in Section 1). The user task is defined as a composition of abstract activities (e.g., purchase a book, purchase MP3 player and

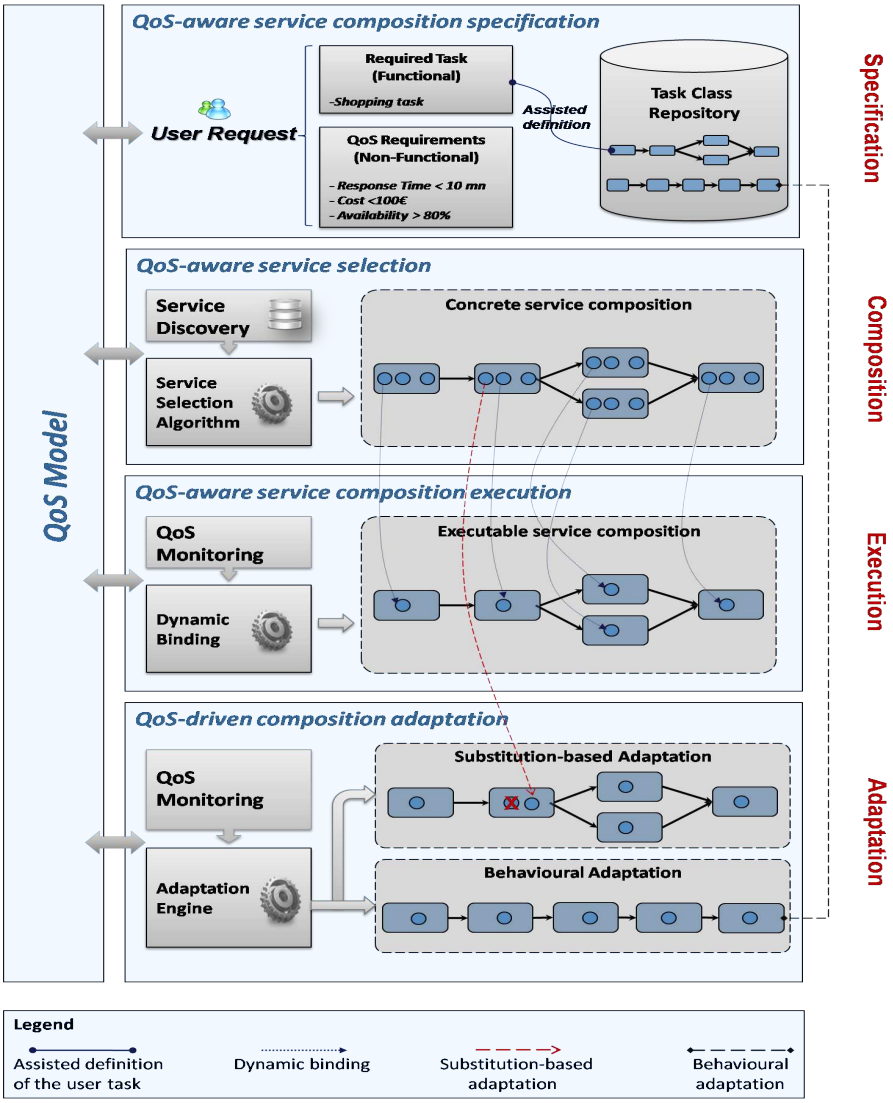

Figure I.2 – Thesis overview

headphones, etc.) structured with respect to given execution patterns (also known as composition patterns) such as sequential, parallel, and conditional execution.

A key assumption in our research is that the user task can be accomplished in several ways, i.e., using various compositions of abstract activities. For instance, the shopping task can be carried in different ways by changing the order in which shopping activities are executed, or by changing the granularity of shopping activities, i.e., splitting (respectively merging) shopping activities into simpler (respectively more complex) ones. To support more than one possible way of realizing the user task, we introduce the novel concept of *Task Class*, which defines several compositions of abstract activities that are functionally

equivalent, i.e., they allow to fulfill the same user task. A task class is stored within a *Task Class Repository* that makes part of our QoS-aware service-oriented middleware. The goal of the task class repository is to provide abstract descriptions of the tasks offered by the pervasive environment, and to assist users in expressing their desired tasks. Once the user task is defined, it is later transformed into a concrete service composition by binding a concrete service offered by the pervasive environment to each abstract activity in the user task.

2. QoS requirements, which describe the set of constraints imposed by the user on QoS properties associated with the required task (e.g., the total price and execution time of the shopping task). Given the heterogeneity of actors in pervasive environments, users and service providers may use different vocabularies to define the required and offered QoS. To cope with this issue, QoS properties required by the user are mapped to those offered by concrete services (available in pervasive environments) using a common QoS model.

Once the user request is defined, our QoS-aware service-oriented middleware processes the user request through the selection and composition of concrete services available in pervasive environments and able to meet the functional and QoS requirements of the user. To do so, we introduce a novel service selection algorithm designed with respect to major requirements of service selection in pervasive environments, notably timeliness, adaptation support and distributivity.

Our algorithm produces several concrete service compositions meeting the user requirements, but providing different levels of QoS. That is, our algorithm selects several concrete services to each abstract activity in the user task. When actually executing the composition, QoS of services is monitored and assessed at run-time (to which we refer to as run-time QoS of services), and just in time, only one service is bound to each activity in the user task, which is known as *Dynamic Binding* of services [Di Penta et al., 2006; Pautasso and Alonso, 2005]. Dynamic binding consists in binding services to abstract activities just before invoking these services, thus allowing to cope with the difference between the QoS advertised by service providers and the run-time QoS of services. In our thesis, we do not deal with dynamic binding, but rather adopt existing dynamic binding approaches e.g., [Châtel et al., 2010].

During the execution of the selected composition, the QoS provided by the services forming the composition may fluctuate due to the dynamically changing conditions of pervasive environments (e.g., services may join and leave the environment, user mobility, etc.). The running composition must then be adapted accordingly. Towards this purpose, our middleware sup-

ports QoS-driven composition adaptation at run-time through the provision of two adaptation strategies: (i) service substitution, which consists in replacing services providing unsatisfactory QoS with alternative services. If service substitution cannot be accomplished (i.e., there are no substituting services), our middleware proceeds to (ii) behavioural adaptation, which is based on the task class concept introduced above. It consists on fulfilling the user task using an alternative behaviour, i.e., an alternative composition of abstract activities defined in the task class repository. Based on the alternative behaviour, service selection and composition is performed again and a new service composition is built to accomplish the user task.

In accordance with the above highlighted contributions, we organize the remainder of this document as follows:

- **Chapter II** is devoted to the state of the art of QoS-aware service-oriented middleware. It is structured with respect to key functionalities provided by such middleware, notably QoS-enabling specification, QoS-aware service discovery, QoS-aware service composition and QoS-driven composition adaptation.
- **Chapter III** presents the first contribution of this thesis, which consists in a QoS model that provides the appropriate ground for QoS awareness in pervasive environments.
- **Chapter IV** presents our QoS-aware service composition approach for pervasive environments. Our approach aims at building service compositions able to fulfill the user task while meeting its associated QoS requirements. Particularly, our approach focuses on the specific problem of QoS-aware service selection under global QoS requirements (i.e., requirements imposed on the whole service composition), which is known to be NP-hard [Yu et al., 2007]. To solve this issue, we propose a novel efficient heuristic algorithm based on clustering techniques. Our algorithm represents a key contribution of this thesis.
- **Chapter V** presents our QoS-driven composition adaptation approach and its underlying adaptation strategies (i.e., service substitution and behavioural adaptation). In this chapter, we specifically focus on the behavioural adaptation strategy, which consists in finding an alternative composition of abstract activities having a different structure (i.e., coordinated with respect to different composition patterns) or a different activities granularity. To do so, we first introduce the task class concept. Then, we show how to reduce behavioural adaptation to a graph comparison problem, more specifically *graph homeomorphism determination* problem. Using graph homeomorphism determination introduces a novel idea towards behavioural adaptation and represents another important contribution of this thesis.
- **Chapter VI** presents QASOM, a prototype implementation of our QoS-aware service-

oriented middleware. Based on this prototype, we present experimental evaluations to assess the efficiency of our middleware.

- **Chapter VII** summarizes the contributions of this thesis and discusses future research issues related to the studied topic.

Chapter II

QoS-aware Service-Oriented Middleware: State of the Art

As introduced in the previous chapter, we opt for a QoS-aware service-oriented middleware solution to address QoS awareness in pervasive computing environments. A first step towards this purpose consists in studying the state of the art of QoS-aware service-oriented middleware and determine their *pros* and *cons*.

To make our survey comprehensive, we study QoS-aware service-oriented middleware in the context of Service Oriented Computing (SOC) in general, while focusing on those middleware solutions addressing service-oriented pervasive environments. Briefly stated, QoS-aware service-oriented middleware supports QoS awareness in service environments through the provision of a set of key functionalities dealing with QoS specification, QoS-aware service discovery, composition and adaptation.

In accordance with the above, our survey is structured with respect to the key functionalities provided by QoS-aware service-oriented middleware. We start by giving an overview of QoS-aware service-oriented middleware in the service oriented community and service-oriented pervasive computing community. Then, we focus on key functionalities provided by QoS-aware service-oriented middleware, viz., (i) QoS-enabling specifications (i.e., specifications enabling QoS awareness), (ii) QoS-aware service discovery, (iii) QoS-aware service composition, and (iv) QoS-driven composition adaptation.

1 QoS-aware Service-Oriented Middleware

Service Oriented Middleware (SOM) is a key enabler of service-oriented computing as it supports the service-oriented interaction pattern through the provision of proper functionalities for deploying, publishing, discovering, accessing, composing and adapting services at run-time

[Issarny et al., 2011].

The above functionalities provided by SOM are challenged by several cross-cutting requirements, notably QoS awareness [Maia et al., 2009]. Indeed, performing user tasks while delivering satisfactory QoS requires bringing QoS awareness to all the stages (i.e., specification, discovery, composition and adaptation of services) needed to achieve the user task. That is, service specification, discovery, composition and adaptation must be "QoS-aware".

Towards this purpose, significant research efforts have been devoted to QoS-aware service-oriented middleware in the service oriented community. A first class of middleware present fragmented solutions to QoS awareness as they address only a specific functionality of QoS-aware service-oriented middleware such as service composition (e.g., middleware platforms presented in [Kuehne et al., 2010; Van Hoecke et al., 2005; Zeng et al., 2004]) or service adaptation (e.g., SIROCCO [Fredj, 2009] and middleware platforms presented in [Erradi et al., 2006; González and Ruggia, 2010; Zhai et al., 2009]). A second class of middleware (e.g., METEOR-S [Verma et al., 2005], DySOA [Siljee et al., 2005], A-WSCE [Chafle et al., 2006], SCENE [Colombo et al., 2006], PAWS [Ardagna et al., 2007], VRESCo [Rosenberg, 2009] and the middleware platform presented in [Cavallaro, 2010]) present more comprehensive solutions to QoS awareness in service-oriented environments as they deal with major functionalities of QoS-aware service-oriented middleware. A particular category of QoS-aware service-oriented middleware concentrate on a specific aspect of QoS such as reliability and fault tolerance [Erradi and Maheshwari, 2005; Kareliotis et al., 2009; Zheng and Lyu, 2008].

Along with the emergence of pervasive computing, QoS-aware service-oriented middleware has been increasingly adopted to deal with QoS awareness concerns in pervasive environments. Related to this, several middleware platforms have been proposed in the literature, e.g., SpiderNet [Gu, 2004], Amigo [Ben Mokhtar et al., 2005], Aura [Sousa et al., 2006], PICO [Kalasapur et al., 2007], MUSIC [Rouvoy et al., 2009], MySIM [Ibrahim et al., 2009], PERSE [Ben Mokhtar, 2007] and DAMS-SS [Dutra and Junior, 2010].

QoS-aware service-oriented middleware dealing with pervasive environments focuses on providing advanced solutions to QoS-aware service specification, discovery, composition and adaptation that are customized to the specifics of pervasive environments, notably the dynamics and resource limitations of these environments. Next, we detail the state of the art of each functionality provided by QoS-aware service-oriented middleware, while highlighting the solutions that are more suitable for pervasive environments.

2 QoS-enabling specifications

Our study of QoS-aware service-oriented middleware starts from the analysis of their QoS-enabling specifications. Related to this, two specifications should be addressed: (i) QoS models and (ii) Quality-Based Service Description (QSD) [Benbernou et al., 2010], i.e., the way QoS models are used to specify QoS-aware services.

2.1 Taxonomy of QoS models

As introduced in the previous chapter, QoS is a broad concept and there is no agreed definition neither a standard model for it. Accordingly, QoS-aware service-oriented middleware rely on heterogeneous QoS models. These models can be classified with respect to their underlying QoS properties and their specification language (see Figure II.1).

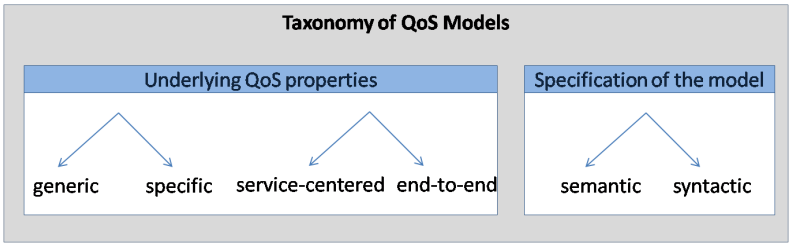

Figure II.1 – Taxonomy of QoS models

- *Generic vs. Specific QoS models.* QoS models can be divided into specific and generic models with respect to their underlying QoS properties. Specific QoS models define a limited number of QoS properties that are generally about commonly used QoS properties such as response time, cost, availability and reliability. ASOB [Kareliotis et al., 2009], DAMS-SS [Dutra and Junior, 2010] and the middleware presented in [Zhai et al., 2009] use specific QoS models.
  Generic QoS models are comprehensive models in the sense that define an extensive categorization of QoS properties. VRESCo [Rosenberg, 2009], Amigo [Ben Mokhtar et al., 2005] and PERSE [Ben Mokhtar, 2007] are based on generic QoS models. For instance, Amigo and PERSE define a QoS model with five categories: *Performance, Reliability, Cost, Security* and *Transaction*. Each category comprehends one or more QoS properties. Regarding the openness of pervasive environments, generic QoS models are deemed more suitable to address QoS awareness as they cover more QoS properties that are required by users and service providers. However, generic QoS models are more complex to handle

(compared with specific QoS models) as their expressiveness attempts to cover all QoS properties with their specific semantics.

- *End-to-end vs. Service-centred QoS models.* QoS models proposed by QoS-aware service-oriented middleware can be also divided into: (i) service-centred models, which focus only on the QoS properties of application services (e.g., models proposed in[Kareliotis et al., 2009; Zhai et al., 2009]) and (ii) end-to-end QoS models which consider all the factors having impact on the QoS delivered to users. These factors include application services as well as their underlying software, network and hardware infrastructure. In pervasive environments, the infrastructure underlying application services may considerably affect QoS provisioning, due to the usage of wireless network and resource-constrained devices. Therefore, considering QoS on an end-to-end basis represents a key requisite of QoS modelling in pervasive computing. Related to this, several middleware platforms define QoS on an end-to-end basis. Yang et al. [Yang et al., 2009] propose the QoPS model (quality of pervasive services) that considers both user-perceived QoS properties (i.e., availability, reliability, price and delay) as well as network QoS properties (i.e., node availability and network delay and reliability). QoPS further formulates the relationship between the user-perceived and network QoS properties.
  Another interesting model is proposed by Chang and Lee [Chang and Lee, 2009]. The authors define a quality model with three dimensions: (i) QoS (Quality of Service) that considers quality properties measurable during the execution of services, (ii) QoC (Quality of Content) that deals with the precision and correctness of the information processed by services and (iii) QoD (Quality of Device) that addresses the processor speed, memory capacity and power of devices underpinning the provision of services.
  Another end-to-end QoS model called DSS (Degree of Service Satisfaction) is proposed by ASPF (Adaptive Service Provision Framework) [Zhang et al., 2006]. DSS is a hierarchical model that considers the network QoS properties (e.g., bandwidth, latency) as well as the device capabilities (e.g., battery life, screen size). In the same way, Zhang et al. [Zhang et al., 2007] present a layered end-to-end QoS model including user, application, environment and resources QoS properties.
- *Syntactic vs. Semantic QoS models.* QoS models can be divided into syntactic and semantic models with respect to their ability to express the semantics of QoS concepts. Syntactic QoS models specify QoS concepts using a predefined syntax with implicit semantics. Thus, users and service providers must use the same syntax to define the required and offered QoS.
  Semantic QoS models (e.g., QoS ontologies) explicitly define the semantics of QoS concepts

and the relationship between these concepts. Such QoS models enable reasoning on QoS concepts and inferring matches between them. Hence, users and service providers can use different QoS syntax to define the required and offered QoS.
Employing syntactic QoS models requires establishing an agreement between users and service providers about a common QoS syntax, which is hard to achieve in open pervasive environments. For this reason, semantic QoS models represent a more suitable solution to QoS description in pervasive environments. However, the usage of semantic QoS models implies reasoning on QoS descriptions on-the-fly at run-time in order to infer matches between the required and offered QoS, which calls for efficient semantic reasoning solutions. In the literature, VRESCo [Rosenberg, 2009] and ASOB [Kareliotis et al., 2009] use syntactic QoS models, whereas METEROR-S [Verma et al., 2005], Amigo [Ben Mokhtar et al., 2005], PERSE [Ben Mokhtar, 2007] and DAMS-SS [Dutra and Junior, 2010] propose semantic QoS models based on the OWL semantic language [1].

2.2 Quality-Based Service Description (QSD)

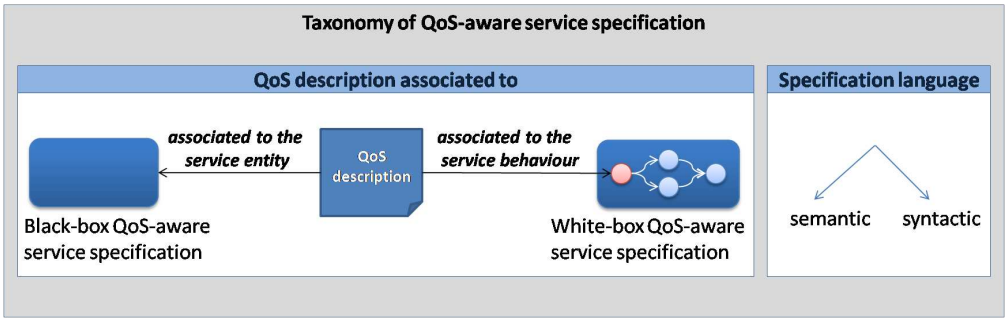

Figure II.2 – Taxonomy of QoS-aware service specifications

QSDs are defined by embedding QoS specifications (i.e., established based on QoS models) into the description of services. This can be carried out according to two approaches (see Figure II.2). In the *black-box approach* (e.g., ASOB [Kareliotis et al., 2009], QuAMobile [Amundsen and Eliassen, 2006]), QoS specifications are associated to services wrapped as black boxes. That is, the specified QoS concerns the whole service entity. In the *white-box approach*, QoS specifications are associated to the functional behaviour of services. More specifically, QoS specifications are associated with elementary parts (i.e., known in the literature as operations [Ben Mokhtar et al., 2007]) forming the service's functional behaviour. For instance, PERSE [Ben Mokhtar, 2007] represents the functional behaviour of services as conversations. A conversation is a set

1. OWL:http://www.w3.org/TR/owl-features/

of operations coordinated by control constructs (e.g., sequence, parallel, loop). In PERSE, QSDs are defined by associating QoS descriptions to elementary operations forming the service conversation.

The black-box approach is generally used when addressing atomic services with simple behaviours (i.e., a single operation). However, the white-box approach is more suitable for long running services with complex behaviours (i.e., various operations, each characterized with a different QoS). The main issue of the white-box approach is that it requires the existence of services' behavioural specifications, which is not always possible, i.e., depending on service information offered by service providers. A particular QSD is presented by VRESCo [Rosenberg, 2009]. The authors define QoS-aware service compositions using three layers: (i) service layer, (ii) choreography layer, and (iii) orchestration layer. The first layer defines QoS attributes associated with atomic services forming the composition. The second layer gives a peer-to-peer view of QoS in the composition. It specifies QoS agreements (viz., SLA) established between atomic services of the service layer. The third layer gives a centralized view of QoS. It focuses on aggregating QoS of atomic services in order to determine the overall QoS of the composition.

Finally, QSDs can be further divided into syntactic and semantic specifications depending on their underlying QoS models (i.e., syntactic or semantic QoS models). A hybrid approach is presented by PICO [Kalasapur et al., 2007]. The authors define QoS-aware services based on two-layers. The first layer describes the semantic concepts associated with services' functions, inputs and outputs, whereas the second layer describes the syntactic types of services' functions, inputs and outputs. For example, a service has an input that is defined using the semantic concept 'name'and the syntactic type 'text'.

3 QoS-aware Service Discovery

QoS-aware service discovery is a fundamental functionality of QoS-aware service-oriented middleware. It allows for determining service candidates that may fulfill the user's required task from the functional and QoS point view. QoS-aware service discovery involves matching the required task and its associated QoS requirements to services available in the environment. The way the matching is carried out impacts the spectrum of discovered services, notably the number of discovered services and the extent to which they fit the user's required task and QoS. Related to this, we distinguish three main criteria allowing to compare QoS-aware service discovery approaches:

- *Syntactic vs. Semantic QoS-aware service discovery.* QoS-aware service discovery ap-

proaches can be syntactic or semantic depending on their underlying QoS models (i.e., syntactic or semantic QoS models). Syntactic discovery approaches (e.g., VRESCo [Rosenberg, 2009]) impose using the same syntax to describe the required and offered QoS. Hence, they provide services that fully correspond to user requirements. However, such discovery approaches constrain the number of discovered services as they disregard services that fit the user requirements but use a different QoS syntax.
Semantic QoS-aware service discovery approaches (e.g., METEROR-S [Verma et al., 2005], Amigo [Ben Mokhtar et al., 2005], PAWS [Ardagna et al., 2007], PERSE [Ben Mokhtar, 2007], DAMS-SS [Dutra and Junior, 2010]) infer matches between heterogeneous QoS terms used to describe the required and offered QoS, hence determining all services in the environment that are able to fulfill the required QoS. Semantic discovery enables taking advantage of the full potential of pervasive environments (in terms of available services), however it requires performing semantic reasoning on services' descriptions on-the-fly.

- *Black-box vs. White-box QoS-aware service discovery.* QoS-aware service discovery approaches can be divided into black-box and white-box approaches with respect to the type of QoS-aware services' specifications. Black-box discovery (e.g., Amigo [Ben Mokhtar et al., 2005], PAWS [Ardagna et al., 2007]) performs the matching between the required task and services available in the environment based on their *profiles*. The service profile presents a high-level description of the service comprising its provided functionalities as well as its Inputs, Outputs, Preconditions and Effects (collectively denoted as IOPEs [Issarny et al., 2011]).
White-box discovery performs a more refined matching, which is based on services' profiles, but also their behaviour. The service behaviour specifies the observable supported execution patterns (often called conversations) of the service in coordination with its environment that allow the service to produce meaningful results [Issarny et al., 2011]. White-box discovery guarantees a full compatibility between the required task and the offered services since it addresses not only what is required by users, but also the way it is fulfilled. However, it may limit the number of discovered services since it discards the services able to meet the functional and QoS requirements of users, but having a different behaviour (than the one required by users). Related to this, white-box discovery is generally used when a specific behaviour is needed ; this is particularly true when the user task comprehends long-running services with complex behaviours.
METEROR-S [Verma et al., 2005] performs QoS-aware service discovery based on services' profiles, but also based on their interaction protocols. The authors represent interac-

tion protocols as activity diagrams and use a protocol mediation module to ensure that interaction protocols required by users are satisfied by service candidates.

4 QoS-aware Service Composition

Given the potential existence of functionally equivalent services in pervasive environments, the user task can be fulfilled by one or more service compositions, which are functionally equivalent but providing different QoS levels. The aim of QoS-aware service composition is to select and assemble service compositions that: (i) meet the QoS requirements of the user, and (ii) maximize the offered QoS. To do so, QoS-aware service selection algorithms should be established. These algorithms represent indeed a key feature of QoS-aware service composition.

Related to this, we structure the state of the art of QoS-aware service composition with respect to five criteria describing the way of assembling compositions and selecting their underlying services (see Figure II.3): (i) the service assembly approach, (ii) the scope of QoS constraints imposed on the composition (i.e., whether QoS constraints cover the whole service composition or an abstract activity of the composition), (iii) the way the selection problem is modelled, (iv) the adopted strategy to resolve the problem, and (v) the technique used to select service compositions.

4.1 Service assembly approach

A first criterion to evaluate QoS-aware service composition concerns the service assembly approach. This criterion is generally associated with service composition from a functional point of view. In our survey, we are interested in the service assembly approach as it determines the way of exploring and selecting service compositions form a QoS point of view also.

Related to this, we distinguish the (i) template-based approach, (ii) graph-based approach and (iii) AI planning. The template-based approach (e.g., PAWS [Ardagna et al., 2007]) assumes that the user task is defined as template formed of a set of abstract activities coordinated using composition patterns (sequence, parallel, choice, loop). To realize the user task, QoS-aware service-oriented middleware proceeds through the discovery and selection of services based on the considered template ; one or more services are then bound to each abstract activity in the user task.

The graph-based approach assumes that the user task is defined as a set of abstract activities (i.e., disregarding the way these activities are coordinated). Each activity is described with its function, inputs and outputs. The graph-based approach (e.g., PICO [Kalasapur et al.,

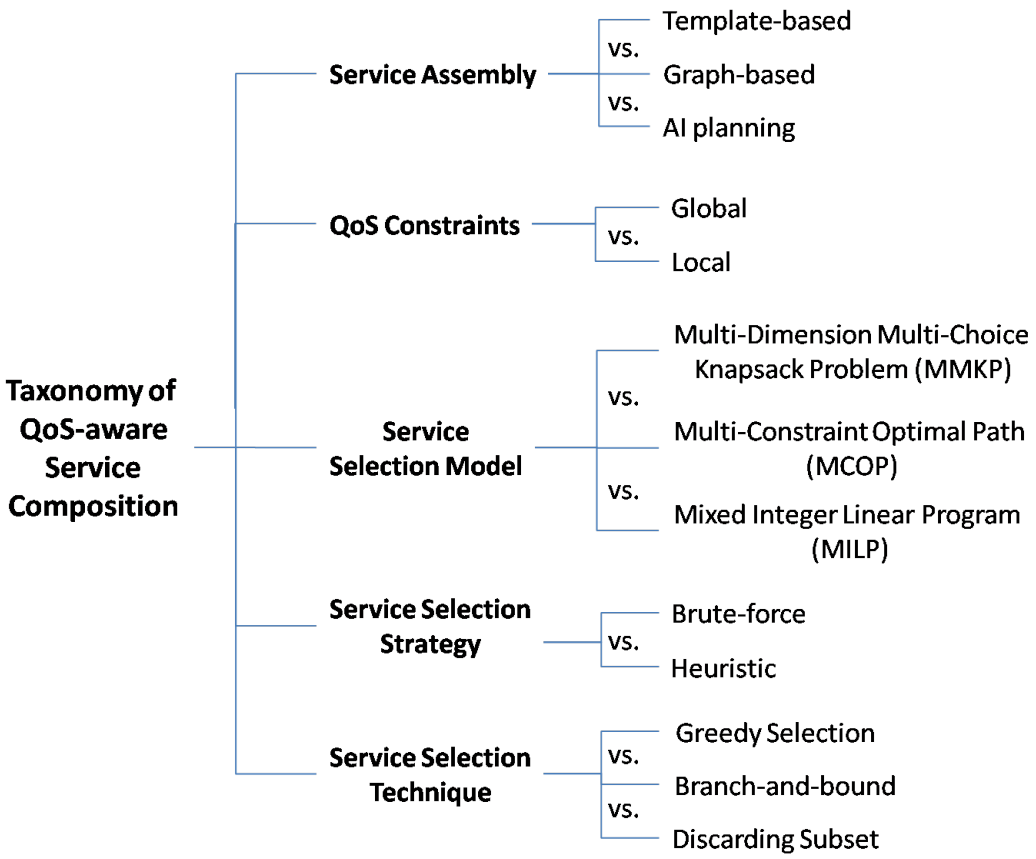

Figure II.3 – Taxonomy of QoS-aware service composition

2007]) proceeds through the construction of a global graph representing all possible service compositions in the environment. The global graph is built by matching services' inputs to services' outputs. Then, the user task is built by exploring the global graph in order to find a path (of services) that satisfies the functionalities, inputs and outputs given by the user.

A third service assembly approach is AI planning (e.g., A-WSCE [Chafle et al., 2006]). This approach requires only the inputs and outputs respectively consumed and produced by the user task. AI planning performs forward and/or backward chaining of services available in the environment in order to find a sequence of services that consumes and produces the considered inputs and outputs, respectively.

The graph-based and AI planning approaches do not support service composition structured with respect to composition patterns (sequence, parallel, choice, loop), they only produce simple sequences of services able to fulfill the user task. On the other hand, the template-based approach requires defining the structure of the user task (in terms of abstract activities and composition patterns), which puts more responsibility on the user side. Therefore, the adoption

of a service assembly approach depends on the trade-off between supporting simple service composition (i.e., sequences of services) and imposing to users to define their required tasks in detail.

4.2 Scope of QoS constraints

QoS-aware selection algorithms aim at selecting service compositions under a number of QoS constraints. We distinguish two types of QoS constraints: global and local constraints, which respectively denote QoS constraints imposed on the whole user task, and constraints imposed on individual activities forming the user task. The type of QoS constraints has substantial impact on the complexity of selection algorithms since QoS-aware selection under global QoS constraints is NP-hard [Yu et al., 2007], whereas QoS-aware selection under local QoS constraints is of linear complexity [Yang et al., 2009].

Most of the algorithms proposed in the literature (e.g., Amigo [Ben Mokhtar et al., 2005], Aura [Sousa et al., 2006], PICO [Kalasapur et al., 2007], SCENE [Colombo et al., 2006], PAWS [Ardagna et al., 2007], Daidalos [Funk et al., 2007], PERSE [Ben Mokhtar, 2007]) focus on QoS-aware service selection under global QoS constraints since it is a challenging task that should be taken up, whereas selection under local QoS constraints (e.g., [Yang et al., 2009]) is seldom addressed in the literature.

4.3 QoS-aware service selection models

QoS-aware service selection models allow the formal specification of the selection problem and lay a foundation for its resolution. Jaeger et al. [Jaeger et al., 2005] present a comprehensive taxonomy of QoS-aware service selection models. The authors map QoS-aware service selection to one of the following models:

- *Multi-dimension Multi-choice Knapsack Problem (MMKP).* Given a set of items with an associated value and a number of resource requirements for each item, MMKP aims at selecting a sub-set of items to put into a knapsack with limited resource capacity, so as to maximize the sum of the values of the included items, while the sum of required resources is less or equal to the knapsack capacity. According to this definition, QoS-aware service selection can be formulated as MMKP by mapping the composition to a Knapsack with limited capacity (i.e., QoS constraints) and service candidates to items with resource requirements (i.e., QoS of services). The Aura project [Sousa et al., 2006] presents a QoS-aware service composition approach that models the selection problem as

a variant of MMKP called 0-1 MMKP. In 0-1 MMKP, the number of selected services for each abstract activity in the user task is either 0 or 1.

- *MultiConstraint Optimal Path (MCOP).* MCOP is an optimization problem which considers a graph formed of nodes and edges. Each node is associated with a benefit and a set of attributes. The goal of MCOP is to determine the path with the highest benefit among all feasible paths from a root node to a sink node under multiple constraints imposed on the attributes. By abstracting service candidates and the control/data flow between them to nodes and edges respectively, QoS-aware service selection can be expressed as an MCOP problem that aims at traversing the graph of service candidates in order to find a path having the highest QoS utility and satisfying global QoS constraints. PICO [Kalasapur et al., 2007] models QoS-aware service selection as a variant of MCOP called multi-constrained path selection (MCPS) that aims at selecting a path in a multi-constrained graph of services.
- *Mixed Integer Linear Program (MILP).* The goal of a Mixed Integer Linear Program (MILP) is to to determine a vector of variables that maximizes or minimizes an objective function, given a set of constraints imposed on variables and represented as linear inequations. QoS-aware selection can be expressed as a mixed integer linear program by defining the global QoS of the whole service composition as an objective function and global QoS constraints as linear constraints. The main limitation of modelling QoS-aware selection using MILP is that the considered QoS properties must be aggregated in an additive way (i.e., using the operator '+' in order to express QoS constraints as linear inequations), which is not supported by all QoS properties. A-WSCE [Chafle et al., 2006], SCENE [Colombo et al., 2006], PAWS [Ardagna et al., 2007], Daidalos [Funk et al., 2007] and VRESCo [Rosenberg, 2009] are QoS-aware service-oriented middleware modelling the service selection problem using MILP.

4.4 QoS-aware service selection strategies

Based on the models presented above, QoS-aware service-oriented middleware approaches proceed to the selection of services using different strategies. Like all the algorithms dealing with combinatorial problems, QoS-aware selection algorithms can be divided into two broad classes: (i) brute-force-like algorithms and (ii) heuristic algorithms. The first class of algorithms aim at determining the optimal service composition (i.e., with the highest QoS) by exploring all possible compositions of services. Amigo [Ben Mokhtar et al., 2005], COCOA [Ben Mokhtar et al., 2007], Daidalos [Funk et al., 2007] and the approach proposed by [Lee and Lee, 2006] fall

under brute-force algorithms as they evaluate QoS of all possible compositions of services, and then select the optimal one. These algorithms have high computational cost (NP-hard [Yang et al., 2009]), thus they can not be executed in a timely manner with respect to spontaneous interactions with the user aimed at by pervasive computing. To cope with this issue, a second class of solutions propose heuristic algorithms that find near-optimal compositions, i.e., compositions that meet global QoS constraints and maximize the QoS delivered to the user. This class of algorithms do not explore all possible compositions of services ; they rather use different heuristics to explore the set of service compositions that most likely can lead to a satisfactory solution. For instance, the selection algorithm proposed by the Aura project [Sousa et al., 2006] uses the QoS utility to required resources ratio as a heuristic to determine near-optimal compositions, whereas PICO [Kalasapur et al., 2007] applies two heuristic algorithms (viz., the extended Dijkstra algorithm and Bellman Ford algorithm [Xiao et al., 2004]) to resolve QoS-aware service selection formulated as an MCPS problem.

4.5 QoS-aware service selection techniques

QoS-aware service selection algorithms use various techniques to explore and select service compositions. The goal of the selection technique is to reduce the number of service compositions to be investigated, thus enabling to enhance the timeliness of the algorithm. Several selection techniques have been proposed in the literature [Jaeger et al., 2005]. In our study, we identify three main selection techniques proposed by QoS-aware service-oriented middleware: (i) greedy selection, (ii) branch and bound and (iii) discarding subsets.

Greedy selection is a technique that selects, for each abstract activity in the user task, the service candidate with the highest QoS. The selection is performed for each abstract activity individually and it is generally used for QoS-aware selection under local QoS constraints [Jaeger et al., 2005].

Yang et al. [Yang et al., 2009] present a greedy selection algorithm called LOSSA (Local Optimal Service Selection Algorithm). LOSSA proceeds to the selection of services for each abstract functionality in the user required task through two steps: *I-level* selection that filters out service candidates that fail individual QoS constraints, and *A-level* selection that selects services resulting from the I-level selection based on a comprehensive score aggregating all QoS values of services. A second greedy selection approach is presented by Chang and Lee [Chang and Lee, 2009]. The authors use PROMOTHEE [Brans and Vincke, 1985], a multi-criteria decision making (MCDM) technique to select the best service for each abstract activity in the user task. Another interesting greedy selection approach is presented by the DAMS-SS middleware [Dutra

and Junior, 2010]. This approach consists in three steps. First, the middleware clusters services based on their QoS properties using the Self-Organizing Map (SOM) algorithm. Second, it uses ADAPTREE which is an adaptive decision algorithm to generate a hierarchy of service clusters with respect to the importance of QoS properties. Finally, it decides about the best cluster of services in terms of QoS using the ANFIS (Adaptive Network-based Fuzzy Inference System) algorithm.

The main issue of greedy selection techniques in general, is that they cannot ensure meeting global QoS constraints. To cope with this issue, other global selection techniques have been put forward. The selection algorithm proposed by the Aura middleware [Sousa et al., 2006] applies a *branch-and-bound* technique to resolve QoS-aware selection formulated as an 0-1 MMKP problem (as explained above). PICO [Kalasapur et al., 2007] uses the *discarding subsets* technique to resolve the MCPS problem. Discarding subsets [Jaeger et al., 2005] consists in exploring service compositions over several steps. In each step, only relevant compositions (i.e., determined with respect to certain criterion) are considered for the remainder of the selection. Accordingly, the extended Dijkstra and Bellman Ford algorithms used by PICO traverse the graph of services over several steps; in each step only non-dominated paths [Xiao et al., 2004] are kept for the remainder of the selection.

Overall, selection techniques vary from one QoS-aware service-oriented middleware to another. The choice of a selection technique depends on the kind of QoS constraints to deal with (i.e., local or global QoS constraints), the way QoS-aware service selection is modelled, and the adopted strategy to solve it.

5 QoS-driven Composition Adaptation

Service composition adaptation is a key functionality of QoS-aware service-oriented middleware. It enables service compositions to evolve in dynamic pervasive environments and adequately react to various changes in these environments. As we focus on QoS concerns in pervasive environments, in our work we concentrate on adaptation driven by QoS changes (e.g., QoS fluctuation), known as QoS-driven composition adaptation.

The goal of QoS-driven composition adaptation is to adjust running QoS-aware service compositions in order to ensure meeting QoS requirements, and/or to optimize the QoS delivered to users [Kazhamiakin et al., 2010]. Towards this purpose, various approaches have been proposed by QoS-aware service-oriented middleware. To evaluate these approaches, we recall the survey proposed by Kazhamiakin et al.. The authors present a conceptual model and a detailed

taxonomy of service adaptation in general (i.e., including QoS-driven composition adaptation). The authors introduce a set of criteria for the comparison of adaptation approaches (the survey is interesting for the reader who would like to probe further). Based on this survey, we identify our main criteria allowing to assess QoS-driven composition adaptation approaches (see Figure II.4): (i) adaptation model, (ii) adaptation timing, (iii) adaptation subject, (iv) adaptation mechanism, and (v) the scope of adaptation effect [Kazhamiakin et al., 2010].

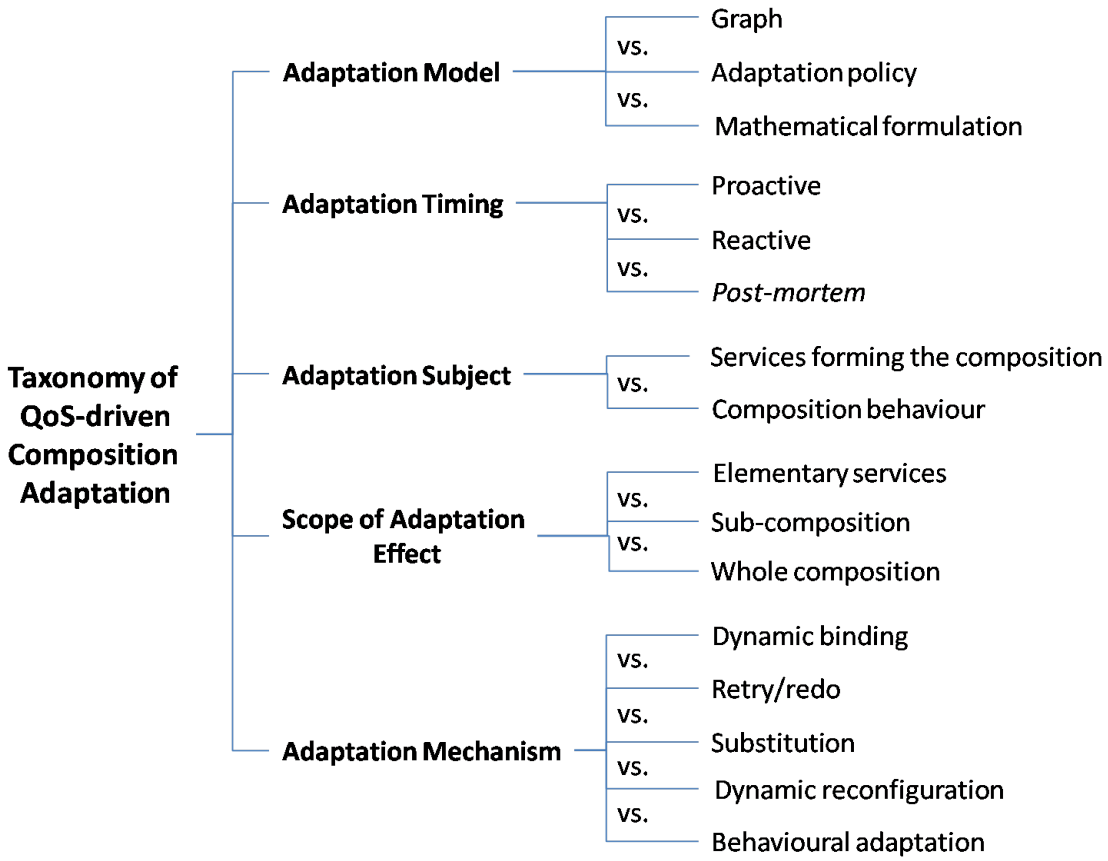

Figure II.4 – Taxonomy of QoS-driven composition adaptation

5.1 Adaptation model

Adaptation model concerns the way the adaptation problem is formulated. We identify (1) the graph-based adaptation model, (2) adaptation policy, and (3) mathematical formulation.

PICO [Kalasapur et al., 2007] presents a graph model to enable adaptation. The authors define services available in the environment as graphs and combine these services into a global graph representing all possible composition of services in the environment. If one or more

services making part of a running composition fail, PICO traverses the global service graph in order to find alternative services able to replace the failed ones.

Zhang et al. [Zhang et al., 2007] define an adaptation policy which allows for expressing the behaviour of users, user tasks, the environment and services available in the environment according to the *event-condition-action* paradigm. Based on the proposed policy, the authors introduce an adaptation approach that selects and assembles actions (i.e., associated with users, user tasks, the environment and services) in order to adapt the running service composition. This is carried out through two steps: (i) assessing the effect of each policy action and choosing those actions complying with the current execution context of the running composition, then (ii) selecting the set of optimal adaptation actions (i.e., actions allowing to achieve the highest QoS) and their associated services (i.e., services allowing to achieve these actions).

The Aura project [Sousa et al., 2006] presents a mathematical formulation that expresses adaptation as an optimization problem. This problem consists in selecting the optimal reconfiguration of a running QoS-aware service composition among several possible reconfigurations (where each reconfiguration is defined as a set of service substitutions). The authors apply an optimization algorithm, which selects the optimal reconfiguration, i.e., a set of substituting services able to fulfill the user required task while providing a higher QoS than the currently running services.

5.2 Adaptation timing

Adaptation timing defines the moment of time when the adaptation is performed [Kazhamiakin et al., 2010]. Related to this, we distinguish: (i) *Proactive*, (ii) *Reactive*, and (iii) *Post-mortem* adaptation.

Proactive adaptation (e.g., PERSE [Ben Mokhtar, 2007], Aura [Sousa et al., 2006], [Zhang et al., 2007]) anticipates future needs of adaptation and determine possible adaptation actions of QoS-aware service compositions. These actions are applied later when adaptation is effectively triggered. Proactive adaptation allows then for enhancing the agility and timeliness of adaptation approaches. However, it does not consider the variations occurring between the time of computing adaptation scenarios and the time these actions are effectively applied (e.g., services that join the pervasive environment after the computation of the adaptation scenario).

Reactive adaptation (e.g., PICO [Kalasapur et al., 2007]) computes and applies adaptation actions on-the-fly just after adaptation is triggered, therefore it considers the actual execution context of QoS-aware service compositions. However, the delay of reactive adaptation may be significant (compared with proactive adaptation), since adaptation actions are computed

on-the-fly.

Post-mortem adaptation ((e.g., A-WSCE [Chafle et al., 2006])) represents a third class of adaptation timing. It is characterized by a big gap between adaptation triggering and adaptation execution [Kazhamiakin et al., 2010]. Generally, the *post-mortem* adaptation is accomplished by bringing about deep changes on the running service composition such as re-designing the composition and altering its behaviour.

5.3 Adaptation subject

Adaptation subject refers to the entity that should be modified by the adaptation approach [Kazhamiakin et al., 2010]. We distinguish the *composition behaviour* and the *services forming the composition*.

By behaviour we mean the observable supported execution patterns (often called conversations) of the service in coordination with its environment that allow the service to produce meaningful results [Issarny et al., 2011]. Related to this, PERSE [Ben Mokhtar, 2007] supports adaptive user task specification by providing various behaviours allowing to achieve the same user task. If the execution of the task according to one behaviour fails, an alternative behaviour is then carried out.

Other QoS-aware service-oriented middleware (e.g., Aura [Sousa et al., 2006], PICO [Kalasapur et al., 2007], [Zhang et al., 2007]) focus on adapting the services forming the running composition. That is, these middleware substitute failed services in the composition with functionally equivalent ones, without altering the behaviour of the composition.

5.4 Scope of adaptation effect

The scope of adaptation effect says whether the adaptation is local, i.e., applied locally to a service in the composition ; partial, i.e., concerns a part of the composition (sub-composition) ; or global, i.e., concerns the whole service composition. The Aura middleware [Sousa et al., 2006] opts for a local adaptation approach that affects elementary services in the composition, whereas PICO [Kalasapur et al., 2007] applies QoS-driven composition adaptation locally, partially or globally depending on adaptation requirements.

5.5 Adaptation mechanism

Adaptation mechanisms refer to the techniques and facilities used to adapt service-oriented systems [Kazhamiakin et al., 2010]. Adaptation mechanisms represent a broad research topic. In

this survey, we focus on those mechanisms used by QoS-aware service-oriented middleware namely, *dynamic binding, retry/redo, substitution, dynamic reconfiguration* and *behavioural adaptation*.

In dynamic binding (known also as late binding) [Di Penta et al., 2006], the user task is defined as a set of abstract activities (template), while multiple services are selected as potential candidates to fulfill each activity. At run-time, the selected services are bind or re-bind to abstract activities with respect to QoS changes [Kazhamiakin et al., 2010]. Service discovery is particularly relevant in this context, since it allows providing several alternative services for each abstract activity in the user task. SCENE [Colombo et al., 2006] provides a flexible platform that supports selecting alternative services and dynamically binding one service at run-time.

Retry consists in invoking the failed service (or the service causing QoS decline) once again, whereas redo consists in retrying the service using different input parameters. Both mechanisms are implemented by the PAWS middleware [Ardagna et al., 2007]. The same middleware implements the substitution adaptation mechanism, which consists in replacing the failed service (or the service causing QoS decline) with a functionally equivalent service that provides a satisfactory QoS. The main difference between dynamic binding and substitution is that the latter adaptation mechanism selects substituting services once the failure of a service or QoS decline occurs.

Dynamic reconfiguration represents a broad class of adaptation mechanisms. It includes resource reconfiguration, parameter reconfiguration and service composition reconfiguration [Amundsen and Eliassen, 2006]. Most of QoS-aware service-oriented middleware solutions (e.g., DySOA [Siljee et al., 2005]) focus on service composition reconfiguration, which consists in adding, deleting, replacing services forming service compositions [Amundsen and Eliassen, 2006].

Last, behavioural adaptation consists in executing the service composition with respect to an alternative behaviour [Kazhamiakin et al., 2010]. Alternative behaviours are generally obtained by changing composition patterns structuring service compositions or/and by changing services' granularity (i.e., merging fine-grained services into coarse-grained services or the opposite). A-WSCE framework [Chafle et al., 2006] presents an adaptation approach that proceeds through two steps: (i) generating multiple abstract compositions (i.e., templates) allowing to perform the user task, but behaving differently, and (ii) at run-time, if the execution of the task according to one behaviour fails, an alternative behaviour is then carried out.

Overall, the choice of adaptation mechanisms depends on various parameters including the adaptation timing, adaptation subject and adaptation effect. That is, given the requisites of the pervasive environment, the middleware designer has to choose the proper adaptation mechanism

with respect to the timing, subject and effect of adaptation.

6 Summary and Research Challenges

In this section, we introduce two tables (Tables II.1 and II.2) providing an overview of state-of-the-art QoS-aware service-oriented middleware approaches, respectively, for traditional service-oriented environments and pervasive environments. These tables are structured with respect to the surveyed functionalities of QoS-aware service-oriented middleware and their associated taxonomies.

Based on our survey, we argue that QoS-aware service-oriented middleware approaches addressing pervasive environments present QoS awareness solutions that are closely related to those solutions proposed for service-oriented environments in general. As shown in our survey, the same taxonomy can be applied for both kinds of middleware, they tend to apply similar models, methods and technologies. We deem that this is not sufficient to cope with QoS awareness issues in pervasive environments given the specifics of these environments.

For this reason, research efforts should be devoted to the design of innovative QoS-aware service-oriented middleware that takes into account the challenges brought about by pervasive computing. Below, we identify major challenges imposed by pervasive computing on key functionalities of QoS-aware service-oriented middleware, notably QoS modelling, QoS-aware service composition, and QoS-driven composition adaptation. These challenges are respectively addressed in chapters III, IV and V of the thesis.

1. ***QoS modelling research challenges***. Addressing QoS awareness issues requires establishing a standard and rich QoS model that includes an extensive categorization of QoS properties [Benbernou et al., 2010]. This model should define QoS on an end-to-end basis by considering all the factors impacting QoS delivered to users, notably application services and their underlying system and network infrastructure. In the context of pervasive environments, considering QoS of the infrastructure is of significant importance, because it allows assessing the impact of the dynamics and limited resources of such infrastructure on the QoS provided to users.

   Additionally, the QoS model should define QoS concepts with explicit semantics in order to be both human-understandable and machine-interpretable [Benbernou et al., 2010]. This model should be further extensible so as to allow the addition of new QoS properties on an as-needed basis (e.g., domain-specific QoS properties).

2. ***QoS-aware service composition research challenges***. QoS-aware service selection

is a key requisite of QoS-aware service composition in pervasive environments. Despite the wide literature about selection algorithms, to the best of our knowledge there are no algorithms dealing with major challenges imposed by pervasive environments on QoS-aware service selection. These challenges are mainly about:

- Timeliness: QoS-aware service composition typically involves devices with limited resources and computational capabilities. Thus, the algorithms for selecting and composing services have to be conceived to be efficient. This is particularly true when we deal with QoS-aware service selection under global QoS requirements, which is NP-hard [Yu et al., 2007]. Indeed, pervasive computing envisions fulfilling user tasks on the fly, hence the time available for services' selection and composition is limited compared to the computational complexity of the problem. Existing algorithms used for QoS-aware service selection should be revisited and further investigated in this context.
- Considering end-to-end QoS requirements: Most of existing QoS-aware service selection algorithms consider only QoS of application services. However, in pervasive environments the software, hardware and network infrastructure underlying applications services has substantial impact on QoS delivered to users. Therefore, QoS-aware service selection algorithms should consider user QoS requirements on an end-to-end basis. QoS-aware service selection under end-to-end QoS requirements is already addressed in the literature [Yu et al., 2007]. However, further investigations are required to fit the specifics of pervasive environments (e.g., resource limitations and mobility of devices, and wireless network connectivity).
- Considering run-time QoS: Most of the existing service selection algorithms are carried out based only on QoS information advertised by services providers. Nevertheless, at run-time, the effective QoS provided by services may fluctuate with respect to the advertised one because of the changes that may occur in the pervasive environment (e.g., user mobility, service overloading). To cope with this issue, service selection algorithms should additionally consider QoS of services measured at run-time, referred to as *run-time QoS*. This requires monitoring QoS of all service candidates just before enacting the services, which is difficult to achieve given the high number of services that may be investigated to fulfill the user task. An optimized solution for service selection is needed to this regard. The required solution should fulfill service selection based on the run-time QoS of services while reducing the cost of QoS monitoring at run-time.
- Adaptation support: Service compositions should be adapted at run-time with respect to QoS fluctuations. One approach towards this purpose consists in substituting services delivering unsatisfactory QoS by selecting other services with better QoS. Nevertheless,

performing service selection during the execution of service compositions may delay the execution and even interrupt it. To cope with this issue, alternative service compositions should be selected simultaneously while constructing the composition needed to fulfill the user task. Existing QoS-aware service selection algorithms focus on selecting only one service composition. In this thesis, we aim at selecting several alternative service compositions (i.e., several services should be assigned to each activity in the user task), thus enabling dynamic re-binding of services at run-time.

- Distributivity: Most of the surveyed QoS-aware service-oriented middleware assume a centralized and stationary infrastructure. However, pervasive computing advocates *ad hoc* infrastructure-less environments populated with resource-constrained devices. For this reason, we argue that QoS-aware service selection algorithms should be designed in a distributed fashion enabling them to execute in infrastructure-less pervasive environments. Given that QoS-aware service selection under global QoS requirements is of high computational complexity, selection algorithms in pervasive environments cannot be carried out using only one device, but rather through the collaboration of several devices available in the environment, which calls for a distributed QoS-aware service selection algorithm.

3. ***QoS-driven composition adaptation research challenges***. QoS-aware service compositions have to be adapted at run-time (during their executions) to cope with QoS fluctuations. One approach towards this purpose consists in replacing services forming the composition with alternative services providing better QoS. However, it is not always possible to find alternative services that can specifically substitute the former services. Thus, adaptation approaches should investigate further solutions to fulfill the user task.

   As already explained in Section 5.3, one possible solution is to alter the behaviour of QoS-aware service compositions. Starting from the observation that the user task can be fulfilled in several ways (i.e., with respect to several abstract compositions (templates) which are functionally equivalent but behaving differently), QoS-driven composition adaptation can be potentially accomplished by executing the user task with respect to an alternative behaviour.

   Compared to existing adaptation solutions (e.g., [Benatallah and Motahari-Nezhad, 2006]) where alternative behaviours are determined based on an exact matching of behavioural specifications (notably automata), in this thesis we focus on a flexible matching between alternative behaviours where an abstract activity in the user task can be replaced by a set of fine-grained activities, and inversely (i.e., fine-grained abstract activities are merged

into a coarse-grained activity). Nevertheless, a number of research questions should be resolved towards this purpose. A first question is about how to enable such a flexible matching. A second question deals with the efficiency of this adaptation method. More specifically, the challenge is how to change the behaviour of QoS-aware service compositions in a timely and transparent manner without interrupting their execution, and while respecting QoS requirements.

			DySOA	METEOR-S	SCENE	PAWS	A-WSCE	VRESCo	SIROCCO
QoS-enabling specifications	QoS model	(a) generic (b) specific	-	b	-	b	-	a	-
		(a) service-centered (b) end-to-end	-	-	-	a	-	b	-
		(a) semantic (b) syntactic	-	-	b	a	-	b	-
	QoS-aware composition specification	(a) black-box (b) white-box	-	a	a	a	-	a	ab
		(a) semantic (b) syntactic	-	a	b	a	-	b	ab
QoS-aware service discovery		(a) black-box (b) white-box	-	a	a	a	-	a	ab
		(a) semantic (b) syntactic	-	a	b	a	-	b	ab
QoS-aware service composition	Service assembly	(a) template-based (b) graph-based (c) AI planning	-	a	a	a	c	-	-
	QoS constraints	(a) local (b) global	-	ab	a	ab	b	-	-
	Selection model	(a) MMKP (b) MCOP (c) MILP	-	c	-	c	c	-	-
	Selection strategy	(a) brute-force-like (b) heuristic	-	-	-	b	-	-	-
	Selection technique	(a) greedy selection (b) branch-and-bound (c) discarding subsets	-	-	a	-	-	-	-
QoS-driven composition adaptation	Adaptation model	(a) graph (b) adaptation policy (c) mathematical formulation	b	b	b	c	c	-	a
	Adaptation timing	(a) proactive (b) reactive (c) *post-mortem*	b	ab	a	b	b	ab	b
	Adaptation subject	(a) services (b) composition behaviour	a	a	a	a	ab	a	ab
	Scope of adaptation effect	(a) elementary part (b) sub-composition (c) whole composition	a	ab	ab	a	abc	a	a
	Adaptation mechanism	(a) dynamic binding (b) retry/redo (c) substitution (d) dynamic reconfiguration (e) behavioural adaptation	d	ad	a	bc	ce	a	cd

Table II.1 – Summary of relevant QoS-aware service-oriented middleware for service-oriented environments

			AMIGO	DAMS-SS	PERSE	AURA	PICO	WSAMI	DAIDALOS	MySIM	MUSIC	SpiderNet
QoS-enabling specifications	QoS model	(a) generic (b) specific	a	b	a	b	a	b	a	a	a	a
		(a) service-centered (b) end-to-end	a	b	a	a	a	a	a	a	b	b
		(a) semantic (b) syntactic	a	a	a	b	b	b	ab	b	a	b
	QoS-aware composition specification	(a) black-box (b) white-box	a	a	b	a	a	a	a	a	a	a
		(a) semantic (b) syntactic	a	a	a	b	ab	-	ab	ab	a	b
QoS-aware service discovery		(a) black-box (b) white-box	a	a	b	a	a	-	a	a	a	a
		(a) semantic (b) syntactic	a	a	a	b	ab	-	ab	ab	a	b
QoS-aware service composition	Service assembly	(a) template-based (b) graph-based (c) AI planning	b	-	b	b	b	-	a	-	-	b
	QoS constraints	(a) local (b) global	b	a	b	b	b	-	a	-	-	ab
	Selection model	(a) MMKP (b) MCOP (c) MILP	-	-	-	a	b	-	-	-	-	ab
	Selection strategy	(a) brute-force-like (b) heuristic	a	-	a	b	b	-	-	-	-	b
	Selection technique	(a) greedy selection (b) branch-and-bound (c) discarding subsets	-	a	-	b	c	-	a	a	-	-
QoS-driven composition adaptation	Adaptation model	(a) graph (b) adaptation policy (c) mathematical formulation	-	-	a	c	a	-	b	-	-	b
	Adaptation timing	(a) proactive (b) reactive (c) *post-mortem*	-	b	a	a	b	-	b	-	b	b
	Adaptation subject	(a) services (b) composition behaviour	-	a	b	b	a	-	a	-	b	a
	Scope of adaptation effect	(a) elementary part (b) sub-composition (c) whole composition	-	a	c	c	abc	-	a	-	c	abc
	Adaptation mechanism	(a) dynamic binding (b) retry/redo (c) substitution (d) dynamic reconfiguration (e) behavioural adaptation	-	c	e	cd	cd	-	cd	-	d	c

Table II.2 – Summary of relevant QoS-aware service-oriented middleware for pervasive computing environments

Chapter III

QoS Modelling in Pervasive Environments

Addressing QoS-aware middleware for pervasive environments requires establishing QoS models that provide the appropriate ground for QoS awareness in pervasive environments. A promising approach towards this purpose relies on Semantic Web technologies, notably ontologies. Ontologies provide a formal, syntactic and semantic description model of QoS concepts and relationships between these concepts. Ontologies define QoS concepts with explicit meanings, so they are human-understandable, machine-interpretable, and provide the means for interoperability. Moreover, ontologies are extensible so that new concepts and relationships can be easily added [Benbernou et al., 2010]. In addition, Semantic Web techniques can be used for reasoning about QoS concepts, thus enabling the semantic matching of heterogeneous QoS terms used in different QoS descriptions. Therefore, ontologies cater for sophisticated matching between the required and offered QoS. For these reasons, many ontologies have been proposed for specifying QoS.

Nevertheless, recent surveys about QoS modelling (e.g., [Benbernou et al., 2010]) have showed the lack of well established and standard QoS ontologies. Indeed, existing QoS ontologies do not offer a rich description of QoS that includes an extensive categorization of QoS properties [Benbernou et al., 2010]. Moreover, most of existing QoS ontologies focus only on QoS provided by application services. They do not consider QoS properties associated with other service technology layers, notably the infrastructure layer, which impacts the QoS offered to users [Benbernou et al., 2010].

Hence, in our research [Ben Mabrouk et al., 2009] we focus on the development of a standard-based and rich QoS ontology that provides an extensive categorization of QoS properties in all service technology layers. As we address QoS modelling in the context of pervasive computing, the required QoS ontology should consider features related to pervasive environments. For instance, user mobility and context awareness of application services should be considered as quality features affecting the QoS provided to users.

Given the wide literature about QoS modelling, the required QoS ontology should not be established from scratch, but rather from existing QoS models. Towards this purpose, we investigated existing standards or standardization efforts for QoS modelling. Indeed, standard QoS models represent the consensus of the QoS modelling community, as well as comprehend rich QoS information including various visions of QoS. In this thesis, we investigate the Web Service Quality Model (WSQM) [Min et al., 2007], which is a standardization effort proposed by the OASIS WSQM technical committee for the specification of Web Services' QoS. WSQM is a comprehensive model that defines a well-founded taxonomy of QoS and provides a wide range of QoS properties [Benbernou et al., 2010].

Based on WSQM, we present a QoS model formed of a set of QoS ontologies. Our model considers QoS on an end-to-end basis, in the sense that it defines QoS properties related to application services, but also the system and network infrastructure underlying these services. Our model puts special emphasis on emerging QoS features related to the dynamics of pervasive environments. Next, we first give an overview of WSQM and we assess it with respect to QoS modelling requirements in pervasive environments (Section 1). Then, we detail our QoS model and its underlying ontologies (Section 2). Finally, we discuss the proposed model with respect to existing QoS ontologies (Section 3).

1 WSQM Overview

WSQM (Web Service Quality Model) [1] is a conceptual model for Web Services' QoS which defines three basic concepts: quality associates, quality activities and quality factors. A *quality associate* refers to the person or the organization that is directly or indirectly managing QoS. The set of actions performed by quality associates throughout the whole services' lifecycle are called *quality activities*, whereas the *quality factors* are the attributes used to represent and assess QoS. Our primary focus is on the quality factors, which represent the core entities of QoS. Indeed, QoS models are usually centred on the definition and the classification of quality factors. WSQM formally specifies the quality factors and their associated concepts using the Web Service Quality Description Language (WSQDL), which provides an XML-based description method for standardizing the expression of QoS. WSQDL defines the constructs needed to specify the quality factors, their taxonomy, the way they are assessed and the relationships between them.

The main construct within WSQDL is *QualityFactor*. It is represented with a global struc-

1. WSQM: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsqm

ture that expands into four levels reflecting the complex details of quality factors. These levels are: *SubFactor, Property, SubProperty* and *Function*. For instance, the Security quality factor is divided into three properties: confidentiality, integrity and non-repudiation. Confidentiality may include other sub-properties such as message level confidentiality and user confidentiality. In turn, the message-level confidentiality can be defined, e.g., using the XML-encryption function.

Additionally, WSQDL divides *QualityFactor* into two main groups: *System* quality group and *Business* quality group. The former group describes quality factors related to the implementation of services, whereas the latter group deals with quality factors defining the business aspects of services. The Business quality group includes the *Business Value* quality factor, which comprehends in turn quality sub-factors such as price, penalty and service provider reputation. The System quality group is composed of two parts: varying quality factors and non-varying quality factors. Varying quality factors are determined with metric values that can change at run-time while a service is being used. The *Service Level Measurement* quality factor falls under this category. It includes quality sub-factors such as response time, maximum throughput and availability. On the other hand, non-varying quality factors are determined and fixed before using the service. It includes *Interoperability*,*Workflow Processing*, *Manageability* and *Security* quality factors.

Interoperability is a quality factor evaluating whether a service implementation conforms to standard specifications. Workflow Processing is a quality factor evaluating the ability of a service to participate in a workflow in order to fulfill a common goal ; it includes the reliability, transaction integrity and collaborability sub-factors. Manageability is a quality factor concerning the facilities offered by services to be easily managed. This includes sub-factors such informability, observability and controllability. Finally, the Security quality factor includes the sub-factors: encryption, authentication, authorization, integrity, non-repudiation, availability, audit and privacy.

Complementary to the quality factors, which are centred on describing the individual aspects of QoS, the *Quality Chain* concept introduced by WSQDL gives a global view of QoS. It states that the quality factors are not totally independent and they may influence each other, and hence impact the overall QoS. Indeed, Quality Chain defines a configuration of dependencies describing the correlation between quality factors. This can be used, for instance, to reflect the relationship between BusinessValue factors (e.g., service price) and other quality factors (e.g., response time), which allows for specifying trade-offs that service providers can make in the quality levels they offer.

Although it provides a well-founded specification of QoS at the service level, WSQM presents three main shortcomings regarding QoS modelling in pervasive environments: First, it neglects

QoS associated to other resources and actors participating in service provisioning (e.g., network, devices and end-users). Second, it does not consider emerging QoS factors related to the dynamics of pervasive environments (e.g., adaptability, context-awareness). Third, as already explained, WSQM adopts an XML-based approach to specify QoS, which may bring about the syntactic heterogeneity problem (since XML is a syntactic language). In the following section, we define a semantic QoS model that copes with the aforementioned shortcomings. The proposed model is defined in terms of a set of ontologies expressed using the Web Ontology Language (OWL), which is a W3C recommendation designed for publishing and sharing ontologies. The next section details the proposed ontologies and outlines their usage in the context of pervasive environments.

2 A Semantic End-to-End QoS Model for Pervasive Environments

In our work, we concentrate on QoS knowledge representation rather than a language to specify QoS. To this extent, our approach is to provide a set of QoS ontologies that can be referenced by any appropriate QoS specification language, notably XML-based QoS languages supporting OWL semantic annotations Oldham et al. [2006]. This approach yields semantically enriched QoS descriptions that combine the accuracy of QoS description languages with the rich semantics of QoS ontologies, hence broadening QoS understanding among users and service providers in pervasive environments.

In this section, we present a QoS model formed of a set of QoS ontologies addressing QoS on an end-to-end basis. Our model covers quality factors associated with the main elements impacting QoS in pervasive environments. These elements are mainly about: (i) the environment and its underlying network and system resources, (ii) application services, and (iii) users. Additionally, our model puts special emphasis on quality features related to the dynamics of the environment, application services and users (e.g., user mobility, adaptability and context awareness of application services).

Our model is designed according to a layered approach, thus aiming to provide distinct and easily manageable ontologies. As depicted in (Figure III.1), it comprehends four ontologies:

1. The *QoS Core ontology* incorporates general concepts needed for QoS description (e.g., quality group and quality factor). Most of the conceptual elements of this ontology are derived from WSQDL.
2. The *Infrastructure QoS ontology* specifies quality factors related to the environment and

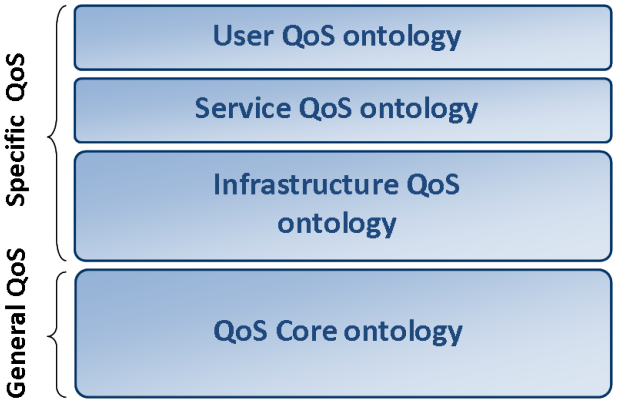

Figure III.1 – QoS model overview

its underlying network and system infrastructure. More specifically, it defines inherent characteristics of the environment where users and services act. These characteristics are mainly about the service density and the number of active users that can be supported, in addition to the ability of the environment to offer functionally equivalent services that can replace each other. The Infrastructure QoS ontology defines also quality factors related to the capabilities of mobile devices and the connectivity of wireless networks.

3. The *Service QoS ontology* specifies quality factors of application services. It recalls the quality factors already defined by WSQM and extend them with other factors (e.g., adaptation and context awareness) supporting the dynamics of the application services. The added factors are carefully selected by examining the dynamicity of pervasive environments. Additionally, this ontology is extendable in that new quality factors (e.g., domain-specific quality factors) can be easily added.
4. The *User QoS ontology* addresses user concerns about QoS. The role of this ontology is twofold: First, it provides the concepts needed to specify user QoS requirements. Second, it specifies quality factors associated with the user such as user mobility.

Ontologies 2), 3) and 4) specialize the general concepts defined in the QoS Core ontology ; they are layered (Figure III.1) from lower (i.e., infrastructure) to higher level of abstraction (i.e., end-user).

2.1 QoS Core ontology

Figure III.2 gives an overview of the QoS Core ontology, which defines basic concepts required for QoS description. In this figure, the boxes denote ontological concepts, and the ellipses denote properties relating these concepts.

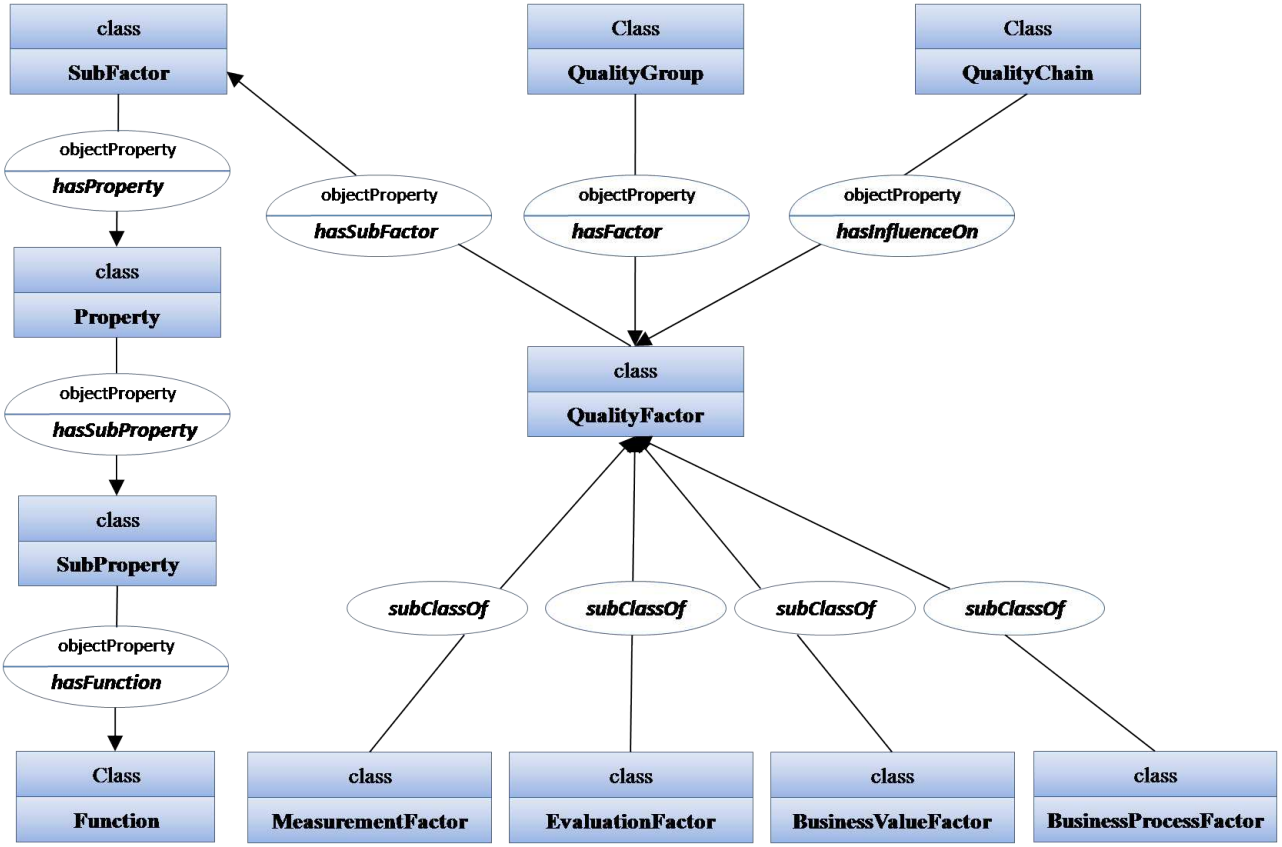

Figure III.2 – Overview the QoS Core ontology

The QoS Core Ontology reproduces the constructs used in WSQDL to define the QoS taxonomy of WSQM. The main concepts of the QoS Core ontology are *QualityGroup, QualityFactor* and *QualityChain*. QualityGroup consists of one or more QualityFactor, which are represented from two points of view: (i) their structure (i.e., SubFactor, Property, SubProperty, Function), and (ii) their type (i.e., MeasurementFactor, EvaluationFactor, BusinessProcessFactor, BusinessValueFactor). The QualityChain concept is defined using the property *hasInfluenceOn* indicating that two or more quality factors are in a quality chain relationship.

For visibility reasons, Figure III.2 does not show further concepts of the QoS core ontology such as the concepts needed to assess quality factors like *MetricType* and *Conformity*. Related to this, it is worth mentioning that the concept *Unit* within MetricType references other external ontologies to define its semantics. Currently, Unit references the OWL time ontology Hobbs and Pan [2006].

2.2 Infrastructure QoS ontology

As already explained, QoS is difficult to evaluate in an accurate manner without considering the infrastructure supporting the provision of services, namely the network and the devices enabling services and users to interact, as well as the characteristics of the environment where services and users act. For this reason, we developed the Infrastructure QoS ontology, which consists of three quality groups: *Network, Device* and *Environment* as depicted in Figure III.3. For visibility reasons, in this figure we only represent quality groups and their underlying quality factors and quality subfactors using different boxes. The relation between theses entities are already explained in the QoS Core ontology (see Figure III.2).

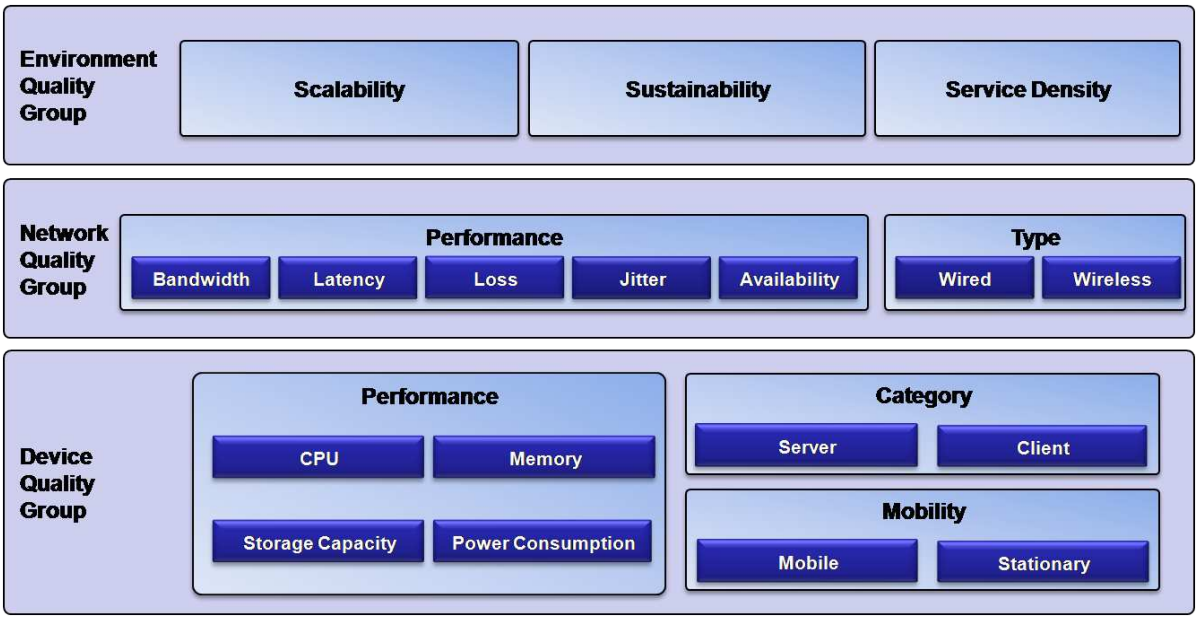

Figure III.3 – Overview the Infrastructure QoS ontology

The Network quality group addresses the communication infrastructure in pervasive environments. It consists of two quality factors: *Type* and *Performance*. The first factor indicates the network category, which can be either *Wired* or *Wireless*. The second factor comprehends quality items specifying the performance of a network. Common performance metrics include *Bandwidth, Latency, Loss, Jitter* Marchetti et al. [2004] and *Availability* Yang et al. [2009]. Bandwidth refers to the rate of data transfer ; Latency refers to the total time needed to deliver a message ; Loss represents the rate of message units lost during the delivery of a message ; Jitter expresses the variation in Latency. Finally, Availability refers to the presence of a node in the network in the transmission range of the user device Yang et al. [2009].

The Device quality group addresses devices hosting application services and supporting end-

users (e.g., PDA, SmartPhone, PC). It defines three quality factors outlining the capacity of these devices: *Category*, *Mobility* and *Performance*. The first factor refers to the role of devices in service provision, which divides into two types: *Client* and *Server* Papaioannou et al. [2006]. The second factor describes the mobility of devices which can be either *Stationary* or *Mobile*. The third factor comprehends quality items specifying common device capabilities, i.e., *CPU, Memory, Storage Capacity* and *Power Consumption*.

The Environment quality group specifies quality factors intrinsically related to the networking environment where users and services exist. Such environments are populated with services supporting user applications. The quality of these environments can be assessed by evaluating their degree of support to user applications [Kalasapur et al., 2006]. This can be specified using three quality factors: Service Density, Sustainability and Scalability [Kalasapur et al., 2006]. Service Density refers to the number of services available in service environments. It indicates in part the ability of an environment to satisfy user requests, i.e., the higher the service density is, the higher is the probability to fulfill the users' tasks. Sustainability measures the environment's ability to sustain employed services if they fail. Sustainability is generally enabled by fault tolerance mechanisms, like identifying alternative services which are able to replace the failed ones. Finally, scalability refers to the ability of the environment to support a large number of active users and to handle their requests in a satisfying manner.

2.3 Service QoS ontology

The Service QoS ontology defines QoS features associated with application services. This ontology reproduces WSQM standard quality factors to define common QoS properties of application services. The Service QoS ontology further specifies new quality factors addressing, on the one hand, the dynamicity of services in pervasive environments and, on the other hand, domain-specific quality factors. The main concepts of the Service QoS ontology are depicted in Figure III.4. In this figure, the coloured boxes represent the quality factors already defined by WSQM, whereas the white boxes represent the added quality factors. Concerning the relations between quality groups, quality factors and quality subfactors, they are already explained in the QoS Core ontology (see Figure III.2).

As already introduced, WSQM divides quality factors into two main groups : *Business* quality group and *System* quality group. The latter group is in turn composed of two parts: varying quality factors and non-varying quality factors. In this section, we aim at extending these groups in order to support dynamic features of services. The responsibility of services with respect to the dynamics of pervasive environments mainly includes supporting adaptability and

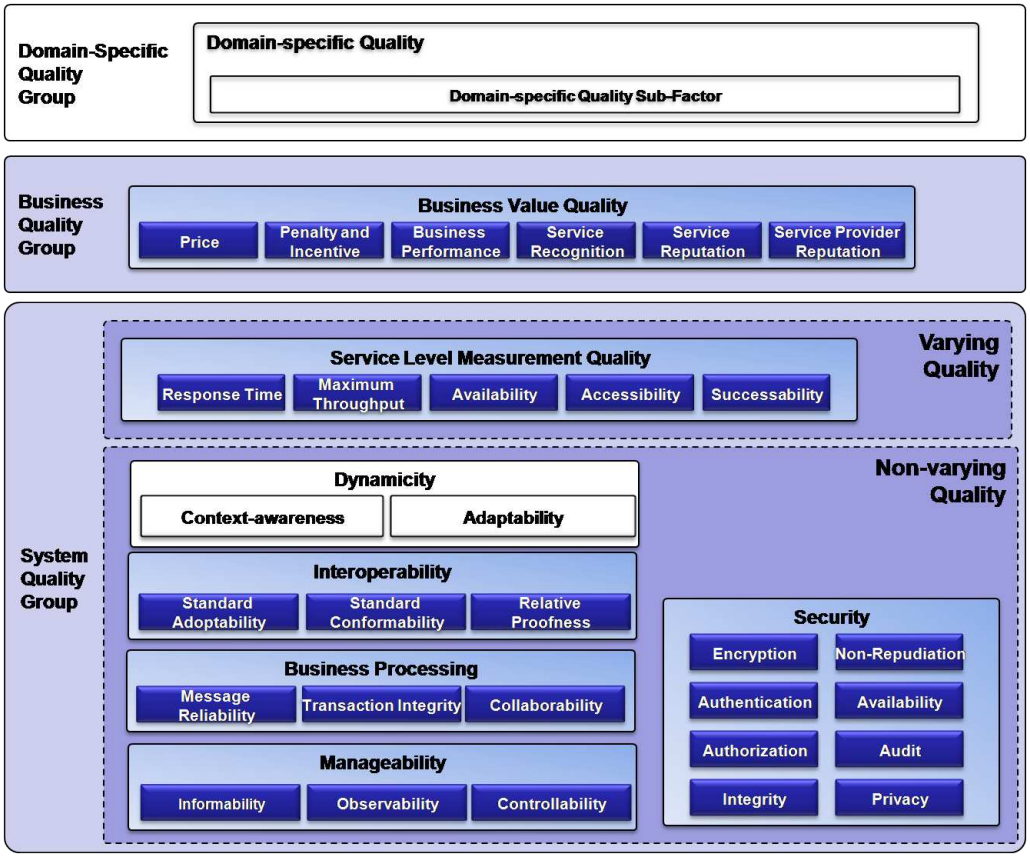

Figure III.4 – Overview the Service QoS ontology

context awareness [Maia et al., 2009]:

- **Adaptability**: Services operating in pervasive environments need to continuously adapt themselves in order to react to changing conditions such as QoS fluctuations and user requirements [Nitto et al., 2008]. Related to this, we define the *Adaptability* quality factor as the ability of a service to adapt itself to changing conditions, and to reconfigure itself accordingly.
- **Context-awareness**: Context awareness is the ability to provide and use relevant information about the networking, and in particular the pervasive environment [Maia et al., 2009]. Context awareness allows for enhancing the QoS provided to users by exploiting context information, such as users' location and their proximity to other devices and services. It further allows for reducing input required from users and replacing it with knowledge of context information, thus enabling fulfilling user tasks in a transparent way that places fewer demands on user attention. Related to this, we introduce the *Context Awareness* quality factor, which describes the ability of a service to gather, manage, use and disseminate context information.

The aforementioned quality factors (i.e., Adaptability and Context-awareness) are non-varying quality factors which have been incorporated into of the System quality group.

Furthermore, in addition to the Business and System quality groups, we define the *Domain-Specific* quality group to support quality factors related to services' application domains. Indeed, pervasive environments offer a wide variety of services dealing with various domains. Thus, users often need to specify quality factors related to particular domains. For instance, if we consider the application scenario (introduced in Chapter I, Section 1) where Bob asks for audio or video streaming service with high encoding quality, the *encoding quality* represents a domain-specific quality factor related to the audio and video streaming application domain.

2.4 User QoS ontology

The User QoS ontology addresses two main issues (Figure III.5): (i) user requirements modelling and (ii) user profile modelling. On the one hand, requirements modelling deals with the concepts needed to express the user's QoS requirements. Related to this, we define the concept *Requirement*, which corresponds to the description of a QoS constraint imposed by the user on the offered QoS. A QoS constraint targets the concept *QualityFactor* defined in the QoS Core ontology, and it is given in terms of *Operator*, *Value* and *Unit*. The latter constructs change according to the type of quality factors. Requirements concerning *EvaluationFactor, BusinessValueFactor* and *BusinessProcessFactor* (see Figure III.2) are expressed using boolean operators (i.e., *is-a, is-not-a*) and string values, whereas requirements regarding *Measurement-Factor* are given in terms of comparison operators (i.e., *equal, not-equal, more-than, less-than, max-value-of, min-value-of*), numerical values and measurement units. Users can further define composite requirements using *and* and *or* operators. Moreover, they are allowed to express their relative preferences about QoS requirements by assigning a certain *Weight* to every requirement.

On the other hand, user profile modelling aims at defining the user's inherent quality factors (i.e., quality factors intrinsically related to the user) such as user mobility and user generated traffic. Related to this, we introduce the concept of *User Profile*, which comprehends two quality factors: *Mobility* and *Traffic*. These factors have been defined based on the model proposed in [Resta and Santi, 2008]. The authors divide User Mobility into three patterns: Stationary users, QoS-driven users, and Mobile users. The first pattern concerns really stationary users or users with constrained movement that does not affect service provisioning. QoS-driven users are mostly stationary, but they move when their perceived QoS level drops below an acceptable threshold. Finally, the mobile users are characterized by continuously moving positions.

Concerning User Traffic, users are divided into three classes of load according to their

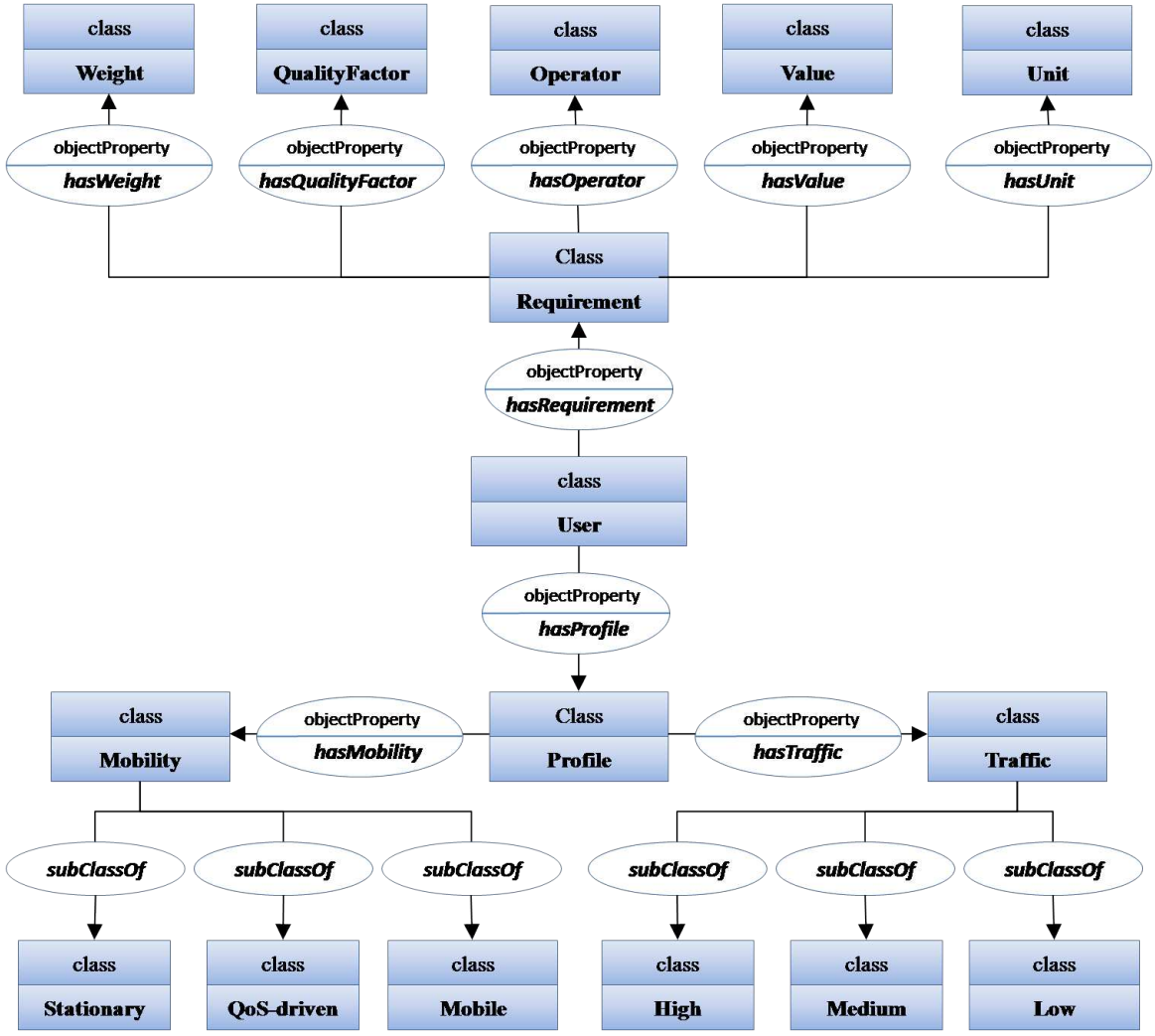

Figure III.5 – Overview the User QoS ontology

generated traffic: low, medium, and high load. The lowest class of traffic accounts for users who are using the network for lightweight services (i.e., in terms of resource consumption) such as e-mailing and Web browsing. The medium class of traffic accounts for users who are using services such as file downloading and audio streaming. Finally, the highest class of traffic accounts for users who make an intensive use of the network, such as video streaming. User Mobility and Traffic are extensible in that new specific patterns of mobility and traffic can be easily defined and added.

3 Discussion

In this section, we study QoS models proposed in the literature in general, and not only in the context of QoS-aware service-oriented middleware (as already presented in Chapter II). Thus, we address a broader field of study with many interesting works.

QoS modelling has been the topic of wide research crossing distinct communities. For the purpose of this work, it is worth mentioning the service-oriented community (e.g., [Dobson et al., 2005; Dobson and Sanchez-Macian, 2006; Kim et al., 2005; Maximilien and Singh, 2004; Papaioannou et al., 2006]), the end-to-end QoS community (e.g., [Marchetti et al., 2004]), as well as the pervasive computing and mobile computing communities (e.g., [Mcnamara et al., 2006; Resta and Santi, 2008]). Even if different in purpose, these efforts have aimed at least to identify relevant quality factors affecting QoS and being associated to a specific resource (e.g., an application service, a network, a device).

For instance, focusing on the service-oriented community, several proposed approaches [Dobson et al., 2005; Maximilien and Singh, 2004] identify quality factors deemed useful for characterizing services' QoS. In [Kim et al., 2005], the authors propose a methodology enabling to express user QoS requirements and to link them to quality factors of services using the notion of traceability. More recently, other approaches [Dobson and Sanchez-Macian, 2006; Papaioannou et al., 2006] have derived semantic languages of QoS ontologies that represent generic models for QoS. These models identify a set of general and abstract concepts needed to define QoS factors and the way they are assessed. Nevertheless, the aforementioned models focus on QoS of application services and do not consider QoS of the infrastructure underpinning these services.

To define more comprehensive QoS models, other works consider QoS on an end-to-end basis by defining QoS of application services and their underlying system and network infrastructure. Marchetti et al. [Marchetti et al., 2004] present a quality model for capturing and reasoning about quality aspects of multichannel services (a channel being the abstraction of a device and a network). This model enables a clear separation of quality aspects of services, networks, devices and users. Furthermore, it embeds rules enabling the evaluation of end-to-end QoS. Nevertheless, this model does not consider the dynamic features of these elements (i.e., networks, devices, services, users) such as user mobility and context awareness of application services.

Another related area of research focuses on representing dynamic features of users, application services and their underlying devices and networks. Capra et al. [Capra et al., 2005] define a context-aware semantic QoS model, which considers three types of resources: *services, sensors* and *components*. Based on this model, the authors address QoS issues related to the mobility of these resources. The authors combine mobility patterns with QoS information in order to

give an accurate evaluation of QoS. This work presents two main drawbacks: first it considers specific patterns of user mobility (i.e., "seasonal" mobility, in the sense that it is related to the daily routines of users). Second it does not take into account QoS at the end-user level (i.e., user QoS requirements).

In the same context, Resta and Santi propose WiQoSM, a Wireless QoS-aware Mobility model [Resta and Santi, 2008]. WiQoSM considers QoS, user mobility, user behaviour and wireless connectivity. This model defines generic mobility patterns and takes QoS at the end-user level into account. The main drawback of WiQoSM is that it totally separates QoS and the other aforementioned aspects, as it considers them in four disjoint models: 1) a QoS model, 2) a user mobility model, 3) a user traffic model, and 4) a wireless technology model. This structure enables WiQoSM to express the above factors, but it does not allow to consider them jointly and establish relations among them, which represents a major shortcoming for having a comprehensive QoS description.

Our model considers collectively the above aspects. Using the quality chain concept, we can describe relationships between QoS factors associated to networks, devices, application services and end-users, thus enabling a comprehensive description of QoS. Moreover, our model focuses on representing QoS knowledge with rich semantic information rather than specifying a language for QoS. Coupled to XML-based or other QoS language specifications, our proposed ontologies can formulate a robust QoS description framework that combines the rich semantics of QoS ontologies with the accuracy of QoS specification languages. Additionally, our model is extendable in that domain-specific QoS factors can be easily added.

The presented model underpins our QoS-aware middleware solution for pervasive environments and its constituent functions, notably QoS-aware service composition, which is the topic of the next chapter.

Chapter IV

QoS-aware Service Composition in Pervasive Environments

Pervasive computing environments enable integrating and composing, on the fly, services that are available in the environment in order to fulfill complex tasks required by users. By user task, we refer to a composition of abstract activities (i.e., functionalities) structured with respect to certain execution patterns such as sequential, parallel, and conditional execution (also known as composition patterns).

Nevertheless, fulfilling the user's tasks only from the functional point of view is not enough to gain user satisfaction. Users further require a certain Quality of Service (QoS) when executing their tasks. Related to this, a lot of research efforts in pervasive computing have been devoted to the composition of services under the user's QoS requirements, which is known as QoS-aware service composition. QoS-aware service composition in a broad topic. At the core of QoS-aware service composition is the issue of QoS-aware service selection, which allows determining services available in pervasive environments and able to fulfill the user's QoS requirements. The problem arises when each abstract activity in the user's task can be fulfilled by several services which are functionally equivalent but providing different QoS levels. The question to be asked is then: "*what is the service that should be selected for each functionality in the user's task in order to meet the user's QoS requirements and produce the highest QoS ?*"

To answer this question, several QoS-aware service selection algorithms have been put forward. However, as already discussed in Chapter II (Section 6), the proposed algorithms do not address major challenges of QoS-aware service selection in pervasive environments. These challenges are mainly about: (i) timeliness, (ii) considering end-to-end QoS requirements, (iii) considering run-time QoS, (iv) adaptation support, and (v) distributivity.

In this chapter, we present QASCO, a QoS-Aware Service Composition approach fulfilling the above requirements. We first give an overview of QASCO (Section 1), then we formally

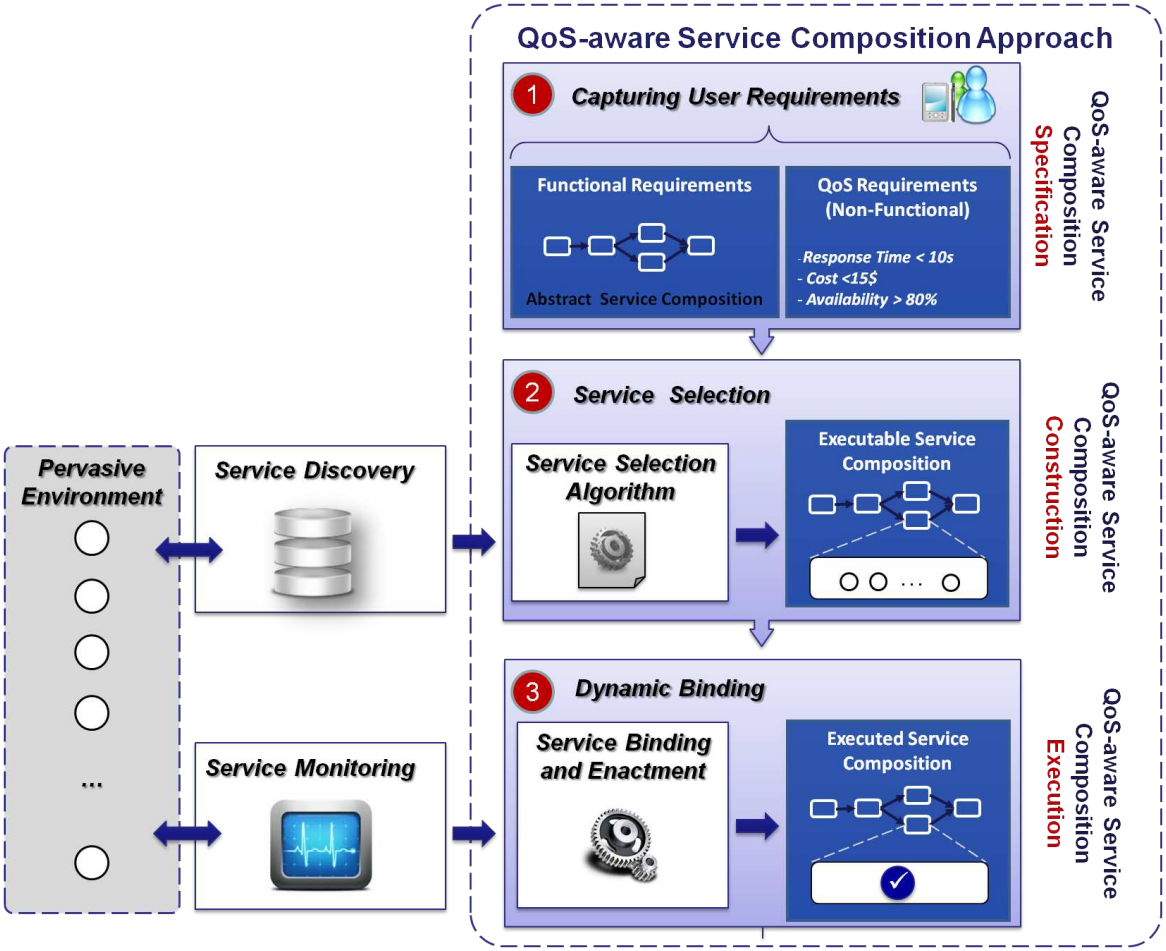

Figure IV.1 – QoS-aware Service Composition Approach

define its underlying QoS-aware service composition model (Section 2). After that, we present an efficient service selection algorithm called QASSA (Qos-Aware Service Selection Algorithm) which represents the key contribution of QASCO (Section 3). Later, we present a distributed version of QASSA, which is able to execute in infrastructure-less pervasive environments (Section 4). Finally, we discuss the novelty of our algorithm with respect to related work in the service-oriented community in general (Section 5).

1 QASCO Overview

The main idea of QASCO is to select and compose services able to fulfill the user task while considering their QoS measured at run-time. Indeed, most of the existing service selection algorithms are carried out based only on QoS information advertised by services available in

pervasive environments. Nevertheless, at run-time the QoS provided by these services may fluctuate with respect to the advertised one because of the changes that may occur in pervasive environments (e.g., user mobility). For this reason, service selection approaches should consider QoS of services measured at run-time (referred to as run-time QoS).

One approach towards this purpose is to use the *Dynamic Binding* technique [Di Penta et al., 2006; Pautasso and Alonso, 2005] (also known as *Late Binding*), which enables binding concrete services to abstract activities of the user task at run-time just before invoking services. However, this technique requires monitoring QoS of all service candidates just before binding them to abstract activities, which is difficult to achieve especially when a high number of services are investigated for fulfilling the user task.

To cope with this issue, QASCO combines both solutions, i.e., (i) initial selection based on advertised QoS, and (ii) final selection based on run-time QoS and dynamic binding of services. The principle of QASCO is to perform a preliminary phase of selection at the global level (i.e., for the whole composition) which is based on QoS information advertised by service providers. This phase is based on QASSA, which selects several alternative service compositions meeting global QoS requirements imposed by users. That is, it selects several services for each abstract activity in the user task. At the dynamic binding stage, one service (among those previously selected by QASSA) is bound to each activity in the user's task. The binding is based on the QoS of services monitored at run-time.

The contribution of QASCO is twofold: First, it enables selecting services based on their run-time QoS while avoiding to monitor all service candidates associated with the user task. Only services advertising high QoS are selected, monitored and evaluated for dynamic binding. Second, it supports adapting service compositions on the fly at run-time by providing several alternative services for each activity in the user task. As depicted in Figure IV.1, our approach comprehends three main steps: (1) capturing user requirements, (2) global service selection and (3) dynamic binding of services.

1. **Capturing user requirements**. QASCO starts from the assumption that users submit their service composition requests *via* a GUI (Graphical User Interface) provided by our middleware platform (i.e., supposed to be installed on their devices). Each request comprehends the functional and QoS requirements of the user.

   As already explained, the user task is formulated as a composition of abstract activities. Each activity is described with its function, inputs and outputs, and it can be fulfilled by one or several concrete services available in pervasive environments. Concerning user's QoS requirements, they are formulated as global QoS constraints associated with a number

of QoS properties and imposed on the whole composition.

2. **Global service selection**. Once user requirements are captured, we proceed to the global selection phase, which aims at selecting, on-the-fly, services able to fulfill the user task. This phase requires a prior step of service discovery which allows for discovering services available in pervasive environments and potentially able to fulfill the required task from the functional and QoS points of view. In this thesis, we do not deal with services' discovery since it is not in the scope of our research. We rather adopt a semantic-based approach for QoS-aware service discovery introduced in [Ben Mokhtar et al., 2006, 2008]. This approach uses domain-specific and QoS ontologies to match the functional and QoS requirements of users to services available in the environment. The matching uses services' advertised information and performs efficient semantic reasoning at run-time. The discovery approach yields as output the set of service candidates potentially able to fulfill the composition from both the functional and QoS points of view. Focusing on the QoS point of view, the discovery phase represents an initial filtering of services, which allows for: (i) discarding services that do not advertise QoS properties required by the user, and (ii) ensuring that the discovered services respect global QoS requirements individually (i.e., each service apart). Services resulting from the discovery phase are further filtered by the global selection phase in order to ensure meeting global QoS requirements jointly when composed inside the user task. To achieve this, we propose an efficient service selection algorithm for pervasive environments called QASSA (Sections 3 and 4).

3. **Dynamic binding**. Complementary to the global selection phase, which selects several alternative services per user task activity based on their advertised QoS, dynamic binding aims at enacting one service per activity among those previously selected based on its run-time QoS, thus ensuring that the composition will indeed provide the QoS level required by the user. To do so, dynamic binding requires a prior phase of QoS monitoring, which allows for determining the QoS level delivered by services at run-time. Based on the monitored QoS, dynamic binding checks whether a service provides a run-time QoS that is equal or higher than the advertised one. If many services meet this condition, the one with the highest run-time QoS is enacted. In the remaining part, we do not focus on dynamic binding since it is not in the scope of our research. We rather recall existing works such as [Châtel et al., 2010].

2 QoS-aware Service Composition Model

The way service composition is specified has a substantial impact on QoS-aware service composition approaches in pervasive environments. In order to address this issue, two major steps are required. First, establishing a QoS model that enables the effective understanding and use of QoS. Second, using the specified model to describe QoS associated to compositions and their constituent services.

2.1 QoS model

We recall our semantic QoS model presented in the previous chapter. Our model is expressed as a set of QoS ontologies defining a detailed taxonomy of QoS. These ontologies are extendible so that they can support the myriad of QoS properties needed in pervasive environments. For the purpose of our QoS-aware service selection algorithm, we distinguish two classes of QoS properties: (i) negative QoS properties, which a have negative effect on QoS, i.e., the higher their values, the lower is QoS, thus they should be minimized ; and (ii) positive QoS properties (e.g., availability), which should be maximized since they have positive effect on QoS, i.e., the higher their values, the higher QoS, is. In the remainder of this chapter, we consider only positive QoS properties in order to make our approach easier to understand. Negative QoS properties can be easily transformed into positive properties using simple arithmetic operations.

Additionally, in our approach we consider user QoS requirements on an end-to-end basis. This is based on our semantic end-to-end QoS model. For instance, in our approach the response time required by users is computed using the expression $rt = t + d$, meaning that the total response time perceived by users when executing a service is the sum of the service response time t and the network delay d. Likewise, the availability perceived by users is computed by the expression $av = a_s \times a_n$, where a_s is the service availability and a_n is network availability. Similarly, the reliability perceived by users is defined by $re = r \times l$, where r represents the reliability of the service and l is the network loss rate.

2.2 Composition model

Our QoS-aware service composition approach is initialized by taking as input a user request $\mathcal{R}$ which is defined as a quadruple $\mathcal{R} = (T, U, P, W)$, where T refers to the required task and U refers to global QoS constraints $U = \langle u_1, .., u_n \rangle$ imposed by the user on a set of QoS properties $P = \langle p_1, .., p_n \rangle$. For each constraint, the user has to specify the relative importance of its associated QoS property by giving a set of weights $W = \langle w_1, .., w_n \rangle$, where w_i is the weight

of QoS property p_i. It is worth noting that the sum of all the weights must be equal to 1, i.e., $\sum_{i=1}^{n} w_i = 1$.

For a user task T, its structure is specified as a set of abstract activities $T = \langle A_1, ..A_z \rangle$ coordinated by composition patterns. In our approach, we consider four types of composition patterns, which are commonly used to construct service compositions [Yu et al., 2007; Zeng et al., 2004]: *Sequence, AND, XOR* and *Loop*.

To each abstract activity A_i in the user task is associated a set of concrete service candidates $S = \{s_{i,1}, s_{i,2}, .., s_{i,m_i}\}$ that are able to realize A_i. Each service $s_{i,k}$ $(1 \leq k \leq m_i)$ is represented by its QoS vector $QoS_{s_{i,k}} = \langle q_1, .., q_n \rangle$, where q_j is the advertised value of QoS property p_j $(1 \leq j \leq n)$.

Only one concrete service s_{i,k_i} is enacted for each abstract activity A_i, thus forming a concrete service composition $\mathcal{C}_i = \langle s_{1,k_1}, .., s_{z,k_z} \rangle$ realizing the user's task.

The question to be asked is then which service should be enacted for each activity in the user task so that the overall composition meets the user's QoS requirements $U = \langle u_1, .., u_n \rangle$. The problem becomes even more complicated when we aim at selecting several alternative service compositions $\mathcal{C}_1, .., \mathcal{C}_v$ in order to support dynamic binding of services (i.e., allowing the choice among several alternative services per user task activity).

The first step to address this problem is to determine how to evaluate the QoS of a service composition $QoS_{\mathcal{C}_i} = \langle Q_1, .., Q_n \rangle$ based on the structure of the composition (i.e., its composition patterns) and the QoS of its constituent services, as discussed in the next section.

2.3 QoS aggregation

Global service selection requires evaluating QoS of service compositions prior to their execution. As the way service compositions will be executed is unforeseen during service selection, the overall QoS of a composition cannot be assessed in an accurate manner, it is rather estimated with respect to possible execution scenarios of the composition. For instance, the response time of two services composed in exclusive choice (XOR) can be estimated either to the longest, or the shortest or the mean response time of the two services. Related to this, we consider three QoS aggregation approaches: (1) optimistic approach (i.e., considering the best QoS value), (2) pessimistic approach (i.e., considering the worst QoS value), and (3) mean-value approach (i.e., considering the average of services' QoS values).

The choice of the proper aggregation approach is performed with respect to the requirements of the pervasive environment, notably the degree to which the user QoS requirements are strict. More specifically, if the user requirements are too strict, the middleware designer shall opt

for the pessimistic aggregation approach, which ensures meeting the user QoS requirements, even when the composition is executed with respect to the worst scenario (i.e., from a QoS point of view). However, it is worth noting that the pessimistic aggregation approach may lead to discarding more solutions (i.e., service compositions) compared with the mean-value or optimistic aggregation approaches.

Table IV.1 gives examples of QoS aggregation formulae of commonly used QoS properties according to the optimistic, pessimistic and mean-value aggregation approaches [Alrifai et al., 2008]. In this table, rt_i, pr_i, av_i, re_i and th_i denote respectively the response time, price, availability, reliability and throughput of services $s_{i,k}$ forming the composition pattern, and k is the estimated number of loops in a repeated activity.

The QoS of a service composition $QoS_{\mathcal{C}_i} = \langle Q_1, .., Q_n \rangle$ is determined by aggregating QoS values of its constituent services $QoS_{s_{i,k}} = \langle q_1, .., q_n \rangle$. QoS aggregation is a complicated task which depends on both the considered QoS properties and the composition patterns underlying the composition. We consider three types of QoS properties with respect to their inherent aggregation semantics: i) additive properties which can be aggregated using the operator $+$ such as response time, ii) multiplicative properties which can be aggregated using the operator $\times$ such as availability and reliability, and iii) properties which can be aggregated using *min* and *max* operators. Each type of QoS properties can be aggregated differently with respect to composition patterns (Seq, AND, XOR, Loop). Concerning the particular case of iterative execution of services (i.e., loop execution), where we need to estimate the number of loops, we adopt a history-based estimation that considers the number of loops previously executed for the considered service.

3 The QASSA Algorithm

3.1 Design Rationale

As introduced in Chapter II, QoS-aware service selection algorithms fall under two broad classes with respect to their selection techniques. We distinguish *local selection* (i.e., greedy selection), which proceeds by selecting the best service in terms of QoS for each abstract activity in the user task separately. This technique has a low computational cost but it can not guarantee meeting global QoS requirements. Second, *global selection* covers the scope of the whole composition and ensures meeting global QoS requirements. However, it is has a high computational complexity.

Based on the aforementioned classification of QoS-aware service selection algorithms, we

QoS aggregation of service compositions						
		Additive properties		Multiplicative properties		Min/Max properties
		response time	price	availability	reliability	throughput
Seq	optimistic	$\sum_{i=1}^{n} rt_i$	$\sum_{i=1}^{n} pr_i$	$\prod_{i=1}^{n} av_i$	$\prod_{i=1}^{n} re_i$	max (th_i)
	pessimistic	$\sum_{i=1}^{n} rt_i$	$\sum_{i=1}^{n} pr_i$	$\prod_{i=1}^{n} av_i$	$\prod_{i=1}^{n} re_i$	min (th_i)
	mean	$\sum_{i=1}^{n} rt_i$	$\sum_{i=1}^{n} pr_i$	$\prod_{i=1}^{n} av_i$	$\prod_{i=1}^{n} re_i$	$(\sum_{i=1}^{n} th_i)/n$
AND	optimistic	max (rt_i)	$\sum_{i=1}^{n} pr_i$	$\prod_{i=1}^{n} av_i$	$\prod_{i=1}^{n} re_i$	max (th_i)
	pessimistic	max (rt_i)	$\sum_{i=1}^{n} pr_i$	$\prod_{i=1}^{n} av_i$	$\prod_{i=1}^{n} re_i$	min (th_i)
	mean	max (rt_i)	$\sum_{i=1}^{n} pr_i$	$\prod_{i=1}^{n} av_i$	$\prod_{i=1}^{n} re_i$	$(\sum_{i=1}^{n} th_i)/n$
XOR	optimistic	min (rt_i)	min (pr_i)	max (av_i)	max (re_i)	max (th_i)
	pessimistic	max (rt_i)	max (pr_i)	min (av_i)	min (re_i)	min (th_i)
	mean	$(\sum_{i=1}^{n} rt_i)/n$	$(\sum_{i=1}^{n} pr_i)/n$	$(\sum_{i=1}^{n} av_i)/$n	$(\sum_{i=1}^{n} re_i)/$n	$(\sum_{i=1}^{n} th_i)/n$
Loop	optimistic	$rt_i \times k$	$pr_i \times k$	$(av_i)^k$	$(re_i)^k$	th_i
	pessimistic	$rt_i \times k$	$pr_i \times k$	$(av_i)^k$	$(re_i)^k$	th_i
	mean	$(\sum_{i=1}^{n} rt_i)/n$	$(\sum_{i=1}^{n} pr_i)/n$	$(av_i)^k$	$(re_i)^k$	th_i

Table IV.1 – Examples of QoS aggregation formulas

propose a heuristic algorithm that combines local and global selection techniques in order to handle the complexity of service selection under global QoS requirements. The objective of our algorithm is to select *near-optimal* service compositions, i.e., service compositions meeting user QoS requirements and providing a level of QoS as high as possible. More specifically, our algorithm aims at selecting several alternative near-optimal compositions, thus enabling to support dynamic binding of services.

Our algorithm proceeds through two main steps: (1) *local selection*, which aims at selecting services with the highest QoS for each abstract activity in the user task and (2) *global selection*, which aims at composing services resulting from local selection and selecting near-optimal service compositions. The selected compositions are ranked according to their QoS.

3.2 Local Selection Phase

The local selection phase represents a preliminary filtering of services aiming to determine services with the highest QoS for each activity in the user task. The selected services are further investigated by the global selection phase in order to select near-optimal service compositions.

To address local selection, we propose investigating clustering techniques, notably the K-means algorithm. Clustering techniques allow for grouping a set of data into several clusters with respect to given criteria. If we apply the same principle to our purpose (i.e., QoS-aware

service selection), we can group service candidates associated with an abstract activity into several clusters according to their QoS values. Each cluster includes services having roughly the same QoS.

The importance of this idea is that it allows for determining the set of services providing high QoS, dynamically, with respect to the number of service candidates. Indeed, given that the number of service candidates varies from an activity to another and from a pervasive environment to another, determining the number of services providing high QoS for each activity in the user task is not obvious. Traditionally, local selection approaches either select only one service (i.e., the one with the highest QoS) or they fix a static number α of services to be selected. Depending on the density of services and the level of QoS they provide, the number α may be excessive or insufficient. On the one hand, if the number of services providing a high level of QoS is superior to α, selecting only α services leads to discarding services with high QoS that can potentially fulfill the user task. On the other hand, if the number of services providing a high level of QoS is inferior to α, selecting α services leads to investigating services with low QoS. Using K-means allows coping with this issue, since K-means dynamically determines the number of services with high QoS depending on the density of services and the level of QoS they offer.

Moreover, K-means allows for establishing several clusters of services representing different QoS levels. Thus, if we consider only services belonging to the highest QoS level, we can considerably reduce the number of service compositions to be investigated, hence improving the timeliness of the algorithm without much loss in the optimality of the resulting composition (i.e., since we consider services with the highest QoS).

Finally, it is worth noting that K-means is a simple algorithm with low computational cost, thus using it as a preliminary selection phase does not increase the complexity of the problem. Indeed, K-means reduces the overall computational cost of QASSA since it prunes the size of service compositions to be investigated at the global selection phase, especially since our method of using K-means is highly selective (see details in Section 3.2.2).

3.2.1 Preliminary Investigation

In this section, we present our preliminary investigations concerning the usage of K-means for local selection, thus sharing our experience with the reader. K-means is an iterative algorithm which takes as input (i) the set of data points defined by their coordinates $d_i = \langle x_1, .., x_n \rangle$ (where n is the number of coordinates), and (ii) a number of centroids defined by their coordinates $c_i = \langle y_1, .., y_n \rangle$. Every centroid represents a cluster of data points.

In each iteration, K-means performs two steps. First it associates data points to their nearest centroid (i.e., cluster) by computing the n-dimensional Euclidean distance between each data point and each centroid:

$$\mathcal{D} = \sqrt[n]{\sum_{j=1}^{n}(x_i - y_i)^2} \tag{IV.1}$$

Second, it updates the centroids' coordinates by computing the average of their associated data points. The clustering further iterates by alternating these two steps until reaching a fixpoint (i.e., data points associated to each cluster do not change any more).

In our work [Ben Mabrouk et al., 2009], we applied the aforementioned method of K-means to fulfill the local selection phase of our algorithm. To do so, we replaced data points and their coordinates by service candidates and their associated QoS vectors. Additionally, we fix the number of centroids to three and we determine their initial coordinates as follows :

- $c_1 = \langle x_1^{min}, .., x_n^{min} \rangle$;
- $c_2 = \langle \frac{x_1^{max} - x_1^{min}}{2}, .., \frac{x_n^{max} - x_n^{min}}{2} \rangle$;
- $c_3 = \langle x_1^{max}, .., x_n^{max} \rangle$.

The above formulae mean that the three centroids take as initial coordinates, respectively, the minimal, middle and maximal value of each QoS dimension (among all service candidates).

Although, our approach yields interesting results in terms of timeliness and optimality [Ben Mabrouk et al., 2009], the way of using K-means based on n-dimensional Euclidean distance brings about a specific problem, which represents the main weakness of the existing algorithms. This problem is about compensation and counterbalance between QoS properties. Indeed, most of existing algorithms [Yu et al., 2007] use QoS utility functions aggregating jointly all QoS properties of services. Such functions may hide the deficiency of services with respect to one or more QoS properties. More specifically, if a service has a low value for a given QoS property (e.g., availability) and high values for other properties (e.g., reliability and response time), when we aggregate all these values using a utility function, the low QoS property can be compensated by the high ones, thus yielding a balanced QoS utility and hence the service may be selected by the local selection phase. However, when checking global QoS constraints during global selection, the service will be most likely rejected as it may lead to the violation of one or more QoS constraints related to the deficient QoS properties (e.g., availability). Therefore, such services should be discarded during local selection in order to improve the timeliness and the optimality of the algorithm.

In our case, the n-dimensional Euclidean distance represents a QoS utility function aggregating all QoS properties of services jointly. Hence, it causes the same problem of compensation

and counterbalance between QoS properties. To cope with this issue, in QASSA we propose a novel way of using K-means to fulfill local selection of services.

3.2.2 Local Selection in QASSA

As explained above, the preliminary approach using K-means for local selection (that we proposed in [Ben Mabrouk et al., 2009]) is subject to the problem of compensation between QoS properties. In QASSA, we apply K-means in a way that guarantees selecting services with high value for each QoS property individually. We first explain this idea, then we give the detailed algorithm of local selection in QASSA.

Idea. To select services providing high values for all QoS properties, we shift from clustering services based on n-dimensional Euclidean distance (i.e., considering n QoS properties jointly) to one-dimensional clustering applied n times (once per QoS property). That is, we cluster service candidates (i.e., associated with each abstract activity in the composition) for each QoS property separately (Figure IV.2). Thus, we obtain for each QoS property p_j several clusters, going from the cluster of services with the highest values of p_j to the cluster of services with the lowest values for the same property (respecting the definitions of positive and negative QoS properties). Further, by considering the intersection of the clusters with the highest values associated with each QoS property, we obtain the set of services providing high values for all QoS properties jointly. To explain this idea in detail, let us consider the scenario introduced in Chapter I where Bob asks for an audio or video streaming service with high availability and high encoding quality. To fulfill the local selection for this activity and determine what are the most interesting audio and video streaming services from the availability and encoding quality points of view, we first cluster audio and video streaming services with respect to their availability into three groups (low, medium and high), then we cluster them again with respect to their encoding quality again into three groups (low, medium and high), and after that we perform the intersection of the group of services with high availability and the group with high encoding quality, thus obtaining audio and video streaming services with high availability and encoding quality at once.

To implement the local selection in QASSA, we use a variant of K-means called K-means++ [Arthur and Vassilvitskii, 2007], which takes as input only the number of clusters (in opposition to K-means which takes as input the number of centroids and the initial values of their coordinates). The number of clusters can be determined using various techniques proposed in the literature, e.g., [Bezdek and Pal, 1998; Davies and Bouldin, 1979; Milligan and Cooper,

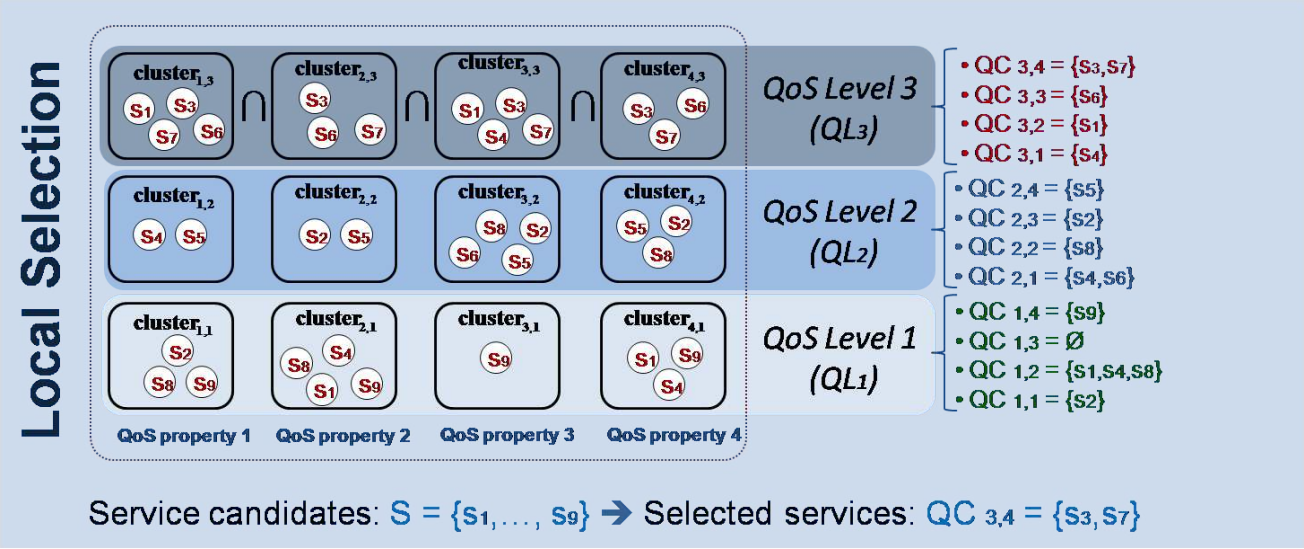

Figure IV.2 – The local selection phase

1985]. In our approach, we opt for the Davies–Bouldin index [Davies and Bouldin, 1979], which is specifically designed for the K-means clustering algorithm.

Algorithm. We present the Algorithm 1 of local selection in QASSA, which comprehends two main steps. The first step (lines 2 to 5) concerns clustering service candidates associated with an abstract activity for each QoS property p_j separately, thus yielding a set of clusters $CL = \langle cl_{j,1}, .., cl_{j,g} \rangle$, where $cl_{j,g}$ is the best cluster (i.e., with the highest values of p_j) and $cl_{1,j}$ is the lowest cluster (i.e., with the lowest values of p_j). Once the clustering is fulfilled, we perform the intersection of the best clusters (i.e., clusters with the highest values) associated with each QoS property, thus obtaining services with high values for all QoS properties. To define the local selection in QASSA in a formal way, we introduce two concepts: *QoS Level* and *QoS Class*.

Definition 1 *Given a set of QoS properties* $P = \langle p_1, .., p_n \rangle$ *and a set of services* $S = \{s_{i,1}, s_{i,2}, .., s_{i,m_i}\}$ *grouped into* g *clusters* $\langle cl_{j,1}, .., cl_{j,g} \rangle$ *for each QoS property* p_j *(where* $cl_{j,1}$ *is the cluster of services with the lowest values of* p_j *and* $cl_{j,g}$ *is the cluster of services with the highest values of* p_j*), we define a QoS level* $QL_r = \langle cl_{1,r}, .., cl_{n,r} \rangle$ *as the set of clusters associated with each QoS property* p_j *and having the same rank* r $(1 \leq r \leq g)$.

The concept of QoS Level is used to group clusters with the same rank r together, thus we can perform their intersection and determine services with QoS values in the rank r. In particular, we are interested in the best QoS level QL_g which groups clusters with the highest

input : a set of activities $T = \{A_1, .., A_z\}$ of size z,
a set of services $S_i = \{s_{i,1}, .., s_{i,m_i}\}$ of size m_i for each activity A_i $(i \in [1, z])$,
a set of QoS properties $P = \{p_1, .., p_n\}$ of size n,
a set of weights on QoS properties $W = \{w_1, .., w_n\}$ of size n,
a QoS vector $QoS_{s_{i,k}} = \langle q_1, .., q_n \rangle$ for each service $s_{i,k}$ $(k \in [1, m_i])$.

output: a set of services S'_i for each activity A_i $(S'_i \subseteq S_i \quad i \in [1, z])$.

foreach $A_i \in T$ **do**
  (Step 1) Clustering services;
  foreach $p_j \in P$ **do**
    K-means++$(S_i, w) \rightarrow \langle cl_{j,1}, .., cl_{j,g} \rangle$;
    // g is the number of clusters
    // apply K-means++ to obtain g clusters for the QoS property p_j
  end
  (Step 2) Services' selection;
  for $r \leftarrow g$ ***downto*** 1 **do**
    for $e \leftarrow n$ ***downto*** 1 **do**
      build QoS Level r;
      $QL_r \leftarrow \langle cl_{r,1}, cl_{r,2}, .., cl_{r,n} \rangle$;
      build QoS class e *of the level* r *by*;
      $QC_{r,e} \leftarrow \{s_{i,k} \setminus s_{i,k} \in \bigcap C_e^{QL_r}\}$;
      // $C_e^{QL_r}$ is the combination of e clusters among QL_r
      intialize the selected QoS class and its score;
      $S' \leftarrow \emptyset$;
      highest-score $\leftarrow 0$;
      select the QoS class with the highest score;
      if $QC_{r,e}$ *is not empty* **then**
        $score_{QC_{r,e}} \leftarrow r \times e \times \sum_{j=1}^{y} w_j$;
        $(w_j \in W \setminus cl_{r,j} \in QC_{r,e})$
        if $score_{QC_{r,e}} >$ *highest-score* **then**
          highest-score $\leftarrow score_{QC_{r,e}}$;
          $S'_i \leftarrow QC_{r,e}$;
        end
      end
    end
  end
  return $(S'_i, score_{QC_{r,e}})$;
end

Algorithm 1: The local selection algorithm

QoS values ($\langle cl_{1,g}, .., cl_{n,g} \rangle$). The intersection of these clusters yields services with the highest QoS values for all QoS properties. To do so, we define the concept of QoS Class.

Definition 2 *Given a QoS level $QL_r = \langle cl_{1,r}, .., cl_{n,r} \rangle$, we define a QoS class $QC_{r,e}$ $(1 \leq e \leq n)$ as the intersection of e clusters among QL_r. Subsequently, a QoS level QL_r comprehends n QoS classes ($QL_r = \{QC_{r,1}, .., QC_{r,n}\}$).*

Literally, a QoS class $QC_{r,e}$ represents the set of services having exactly e QoS properties out of n at the QoS level QL_r. According to this, the QoS class $QC_{g,n}$ groups the best set of services in terms of QoS, since they have all their n QoS properties in the highest QoS level QL_g. Nevertheless, if $QC_{g,n}$ is an empty set (i.e., there are no services with high values for all QoS properties), we try to find the next best QoS class in terms of QoS (e.g., $QC_{g,n-1}$). However, we may obtain several QoS classes having the same level and the same rank (e.g., selecting $n-1$ QoS properties out of n). To determine the best QoS class, we introduce the following score:

$$Score_{QC_{r,e}} = r.e. \sum_{j=1}^{e} w_j \qquad where \quad w_j \in \{W / cl_{j,r} \in QC_{r,e}\} \tag{IV.2}$$

This score means that a QoS class is important (i.e., it includes services with high QoS) when (i) it is associated with a high QoS level QL_r, (ii) it comprehends a high number of QoS properties e in that level, and (iii) the weights w_j associated with these QoS properties are important for the user.

3.3 Global Selection Phase

Once the local selection is fulfilled, we proceed to global selection, which allows for composing locally selected services and determining near-optimal service compositions. Two challenging issues should be addressed concerning global selection. First, the computational complexity of the problem is still NP-hard, even though our local selection phase is highly selective and reduces considerably the number of services to be investigated. Therefore, we need a heuristic algorithm to address the global selection phase. Second, the purpose of our global selection phase is not to select only one composition fulfilling user requirements, but rather several alternative compositions. More specifically, we aim at selecting several services for each abstract activity in the user task, such that, no matter what is the service executed for each activity, the resulting composition meets user QoS requirements. Nevertheless, selecting several alternative service compositions makes the problem even more complicated.

In QASSA, we propose a heuristic algorithm for global selection (Algorithm 2), which selects several alternative service compositions. Our algorithm comprehends five steps:

1. Figure IV.3 depicts an example of running our global selection algorithm for a sequence of 3 abstract activities with 8 services candidates (resulting from the local selection phase). For each activity, we sort its associated services with respect to their overall QoS (lines 2 to 8 in 2). To do so, we introduce a QoS utility function, f, which is defined as follows:

$$f_{s_{i,k}} = \begin{cases} \sum_{x=1}^{z}(w_x \cdot \frac{q_x - q_{min}}{q_{max} - q_{min}}) + \sum_{x=z}^{n}(w_x \cdot \frac{q_{max} - q_x}{q_{max} - q_{min}}) & \text{if } q_{max} - q_{min} \neq 0 \\ 1 & \text{if } q_{max} - q_{min} = 0 \end{cases}$$

where z is the number of positive QoS properties (respectively $n - z$ is the number of negative QoS properties), q_x is the value of the QoS property p_x for the service $s_{i,k}$, and q_{min}, q_{max} denote respectively the minimal and maximal values of p_x among all service candidates associated with the activity A_i. Yet, it is worth noting that we use the QoS utility function f only to sort services (i.e., to establish an order according to which the services are processed), and not to select these services based on their overall QoS utility. Therefore, we avoid the problem of compensation between QoS properties.

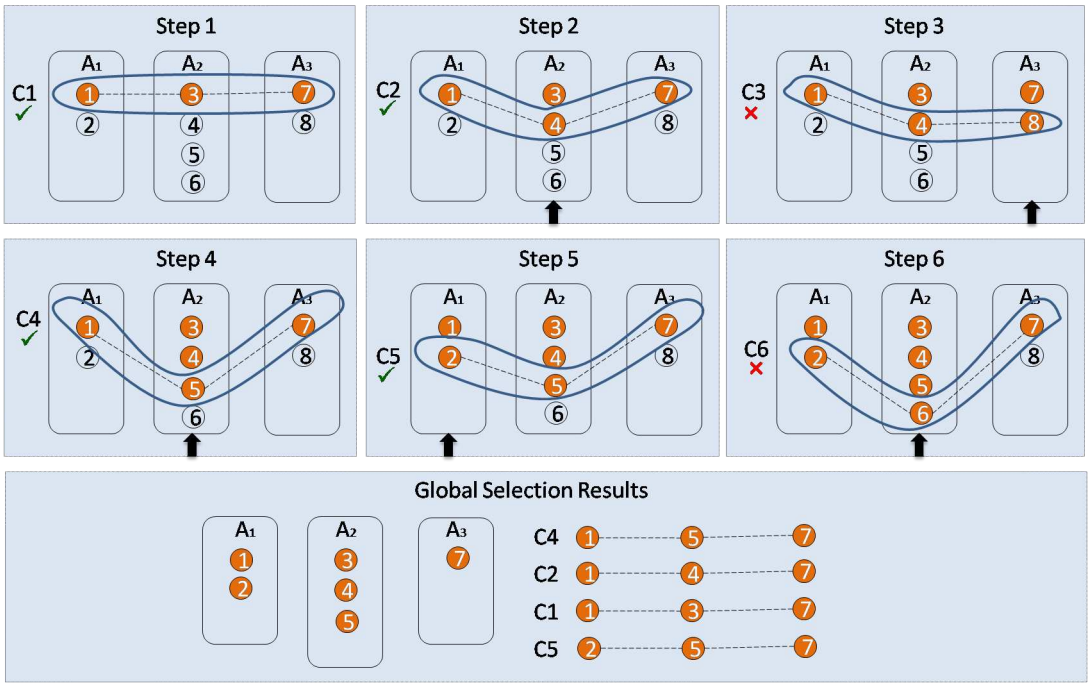

Figure IV.3 – An example of running the global selection algorithm in QASSA

2. As depicted in Figure IV.3, our heuristic algorithm explores service compositions by picking a service from each activity and composing them together. Services with the

input : a set of activities $T = \{A_1, .., A_z\}$ of size z,
a set of services $S'_i = \{s_{i,1}, .., s_{i,y_i}\}$ of size y_i for each activity A_i $(i \in [1, z])$,
a set of QoS properties $P = \{p_1, .., p_n\}$ of size n,
a set of QoS constraints $U = \{u_1, .., u_n\}$ imposed on each property,
a set of weights $W = \{w_1, .., w_n\}$ of size n associated with each QoS property,
a QoS vector $QoS_{s_{i,k}} = \langle q_1, .., q_n \rangle$ for each service $s_{i,k}$ $(k \in [1, y_i])$.
output: a set of service compositions ranked according to their QoS utilities.

1 **begin**
2 *Sort services of with respect to their QoS utilities*;
3 **foreach** $A_i \in T \quad (i \in [1, z])$ **do**
4 **foreach** $s_{i,k} \in S'_i \quad (k \in [1, y_i])$ **do**
5 $f_{s_{i,k}} = \sum_{x=1}^{j} (w_x \cdot \frac{q_x - q_{min}}{q_{max} - q_{min}}) + \sum_{x=j}^{n} (w_x \cdot \frac{q_{max} - q_x}{q_{max} - q_{min}})$
6 **end**
7 $S'_i \leftarrow \text{sort}(S'_i, f)$;
// sort is a function which yields an oredered set given a set of elements and a sorting criteria
8 **end**
9 *Initialize the service composition, its status and the set of solutions*;
10 $\mathcal{C} \leftarrow \langle s_{1,1}, .., s_{z,1} \rangle$;
11 near-optimal $\leftarrow$ true;
12 solutions $\leftarrow \emptyset$;
13 *Initialize the minimal QoS values*;
14 **foreach** $A_i \in T \quad (i \in [1, z])$ **do**
15 **foreach** $p_j \in P$ **do**
16 $q_{j,min} \leftarrow q_j \qquad (q_j \in QoS_{s_{i,1}})$;
17 **end**
18 $QoS_{s_{i,min}} = \langle q_{1,min}, .., q_{n,min} \rangle$;
19 **end**
20 **while** $s_{i,k} \neq s_{i,v_i}$ **do**
21 *Checking whether $\mathcal{C}$ meets user requirements*;
22 **foreach** $p_j \in P$ **do**
23 **foreach** $A_i \in T \quad (i \in [1, z])$ **do**
24 *aggregating the minimal QoS value for the property p_j*;
25 $Q_{j,min} = Q_{j,min} \oplus q_{j,min} \qquad (q_{j,min} \in QoS_{s_{i,min}})$;
// $\oplus$ is a function which aggragates the QoS of a composition for a QoS property p_j with respect to formulas in Table IV.1
26 *checking whether the minimal QoS value $Q_{j,min}$ meets the user requirement u_j*;
27 **if** $Q_{j,min}$ *violates* u_j **then**
28 near-optimal $\leftarrow$ false;
29 break;
30 **end**
31 **end**
32 **end**
33 *Updating the set of solutions and the minimal QoS values*;
34 **if** *near-optimal = true* **then**
35 *add $\mathcal{C}$ to the list of near-optimal service compositions*;
36 solutions $\leftarrow$ solutions $\cup \mathcal{C}$;
37 *updating the minimal QoS values*;
38 **foreach** $A_i \in T \quad (i \in [1, z])$ **do**
39 $QoS_{s_{i,min}} = min(QoS_{s_{i,min}}, QoS_{s_{i,k}})$;
40 **end**
41 **end**
42 *Constructing a new composition by updating one service of $\mathcal{C}$*;
43 *intializing the highest heuristic*;
44 $h_{max} = 0$;
45 **foreach** $A_i \in T \quad (i \in [1, z])$ **do**
46 *compute the heuristic $h_{s_{i,k+1}}$ of service $s_{i,k+1}$ for each activity A_i*;
47 $h_{s_{i,k+1}} = Score_{QC_{r,e}} \times \frac{1}{f - f'}$;
48 *determine the highest heuristic*;
49 **if** $h_{s_{i,k+1}} \geq h_{max}$ **then**
50 $h_{max} \leftarrow h_{s_{i,k+1}}$;
51 **end**
52 **end**
53 *update $\mathcal{C}$ by switching to the service $s_{i,k+1}$ with the highest heuristic*;
54 $\mathcal{C} \ni s_{i,k} \leftarrow s_{i,k+1}$;
55 *reinitialize the composition status*;
56 near-optimal $\leftarrow$ true;
57 **end**
58 *Sort accepted compositions with respect to their QoS utilities*;
59 **foreach** $\mathcal{C} \in$ *solutions* **do**
60 *compute the utility of each composition*;
61 $\mathcal{F}_{\mathcal{C}} = \sum_{x=1}^{j} (w_x \cdot \frac{Q_x - Q_{min}}{Q_{max} - Q_{min}}) + \sum_{x=j}^{n} (w_x \cdot \frac{Q_{max} - Q_x}{Q_{max} - Q_{min}})$
62 **end**
63 solutions $\leftarrow$ sort(solutions, $\mathcal{F}$);
64 **end**

Algorithm 2: The global selection algorithm

highest utilities are explored first. Our algorithm checks whether a service composition $\mathcal{C}_v$ meets global QoS requirements $U = \langle u_1, .., u_n \rangle$ (lines 26 to 30 in 2). If so, $\mathcal{C}_v$ is considered as a solution (i.e., a near-optimal composition). Otherwise, $\mathcal{C}_v$ is rejected. After that, our algorithm proceeds to checking another service composition $\mathcal{C}_w$ obtained by performing a simple update to ($\mathcal{C}_v$). We define a simple update as changing only one service in $\mathcal{C}_v$ by switching from a service $s_{i,k}$ to its following service $s_{i,k+1}$ (i.e., the service having the next best utility f), where both services are associated with the same activity A_i. Changing only one service allows to browse service compositions in an ordered manner and also to speed up the time needed for computing the QoS of the resulting composition (i.e., $\mathcal{C}_w$). This is achieved by simply updating the QoS of the current composition (i.e., $\mathcal{C}_v$) with respect to the difference of QoS between services $s_{i,k}$ and $s_{i,k+1}$ (i.e., $QoS_{s_{i,k}}$ and $QoS_{s_{i,k+1}}$).

3. When a service composition $\mathcal{C}_v$ meets user QoS requirements $U = \langle u_1, .., u_n \rangle$ (for example *Step 1* in Figure IV.3), our algorithm does not switch from $\mathcal{C}_v$ to a new composition $\mathcal{C}_w$ randomly, but it rather attempts to find a service $s_{i,k+1}$ replacing $s_{i,k}$ such that the resulting composition $\mathcal{C}_w$ yields nearly the same QoS utility $\mathcal{F}$ as $\mathcal{C}_v$, hence it, too, can be most likely a near-optimal composition. To do so, we introduce a heuristic h of services $s_{i,k+1}$ associated with each activity A_i (lines 45 to 52 in 2). This heuristic is defined as follows:

$$h_{s_{i,k+1}} = \frac{r.e}{f - f'} \tag{IV.3}$$

where r and e denote the coordinates of $QC_{r,e}$, the QoS class to which $s_{i,k+1}$ belongs, and f and f' denote respectively the QoS utilities of services $s_{i,k}$ and $s_{i,k+1}$.

The above formula means that our algorithm switches to the service $s_{i,k+1}$ (i) belonging to the most important QoS class (i.e., with the highest coordinates r and e), and (ii) having a QoS utility f' which is the closest to the utility f of its preceding service $s_{i,k}$ in the activity A_i. In Figure IV.3, the black arrow indicates the activity in which we switch from the service $s_{i,k}$ to $s_{i,k+1}$ according to the heuristic h.

4. Our algorithm must guarantee that, at run-time, if the running composition fails, we can switch to any accepted service composition (i.e., while always respecting global QoS requirements of the user). For this reason, when checking whether the current service composition $\mathcal{C}_w$ meets user QoS requirements, we must guarantee that $\mathcal{C}_w$ can replace any previously accepted composition $\mathcal{C}_v$ without violating QoS requirements.

   To achieve this purpose, for each user task activity, we compute the minimal QoS value $q_{j,min}$ (of each QoS property p_j) among the service belonging to $\mathcal{C}_w$ and the previously

accepted compositions. Thus, we obtain a QoS vector $\langle q_{1,min}, .., q_{z,min} \rangle$ for each activity A_i. For instance, in Figure IV.3, Step 5, we compute the minimal QoS values of services 1 and 2 for activity A_1, services 3,4 and 5 for activity A_2, and service 7 for activity A_3. After that, we aggregate the minimal QoS values of all the activities and we obtain a QoS vector $QoS_{\mathcal{C}_{min}} = \langle Q_{1,min}, .., Q_{z,min} \rangle$ representing the composition with the minimal QoS values among $\mathcal{C}_w$ and the previously accepted compositions (lines 22 to 32 in 2). If $QoS_{\mathcal{C}_{min}}$ meets global QoS requirements, this means that any combination of services belonging to $\mathcal{C}_w$ or the previously accepted compositions $\mathcal{C}_v$ can also meet these requirements.

5. Our algorithm iterates checking service compositions until investigating all the services associated with each user task activity (as depicted in Figure IV.3). It yields as a result several alternative service compositions ranked with respect to their overall QoS utility $\mathcal{F}$ defined as follows (lines 59 to 62 in 2).

$$\mathcal{F}_{\mathcal{C}_w} = \begin{cases} \sum_{x=1}^{z}(w_x \cdot \frac{Q_x - Q_{min}}{Q_{max} - Q_{min}}) + \sum_{x=z}^{n}(w_x \cdot \frac{Q_{max} - Q_x}{Q_{max} - Q_{min}}) & \text{if } Q_{max} - Q_{min} \neq 0 \\ 1 & \text{if } Q_{max} - Q_{min} = 0 \end{cases}$$

where z is the number of positive QoS properties (respectively n-z is the number of negative QoS properties), Q_x is the value of the QoS property p_x of $\mathcal{C}_w$, and Q_{min}, Q_{max} denote respectively the minimal and maximal values of p_x among the accepted service compositions.

During dynamic binding, we first enact services forming the composition $\mathcal{C}_{max}$ (having the highest QoS utility $\mathcal{F}_{max}$). If the run-time QoS of one service in $\mathcal{C}_{max}$ declines, we enact another service belonging to another composition. Obviously, the QoS utility of the resulting composition is different from $\mathcal{F}_{max}$, but it always respects global QoS requirements.

The importance of this flexibility in choosing and enacting services is twofold. On the one hand, it avoids managing complicated dependencies between services. On the other hand, it avoids rolling back services already executed when a service composition fails. For instance, in the example of Figure IV.3, the global selection yields four service compositions $\langle \mathcal{C}_4, \mathcal{C}_2, \mathcal{C}_1, \mathcal{C}_5 \rangle$ ranked according to their utilities $\mathcal{F}$. To fulfill the user task, we attempt to execute at first the composition $\mathcal{C}_4$. To do so, we start by enacting Service 1, then Service 5. However, if the latter service is not available at run-time, we execute the composition $\mathcal{C}_2$ by enacting Service 4, then Service 7.

We note that if we consider our global selection approach apart (without the local selection phase), it may lead to discarding a lot of possible solutions, because it considers the least QoS

values when selecting service compositions. However, our global selection approach, strongly relies on the quality of the local selection phase. More specifically, there are two facts which make our local and global selection phases complementary. First, the services yielded by local selection are of high QoS, thus at the global selection phase, service compositions will be most likely near-optimal compositions. Second, the services yielded by local selection (for each activity in the user task) belong to the same QoS class, thus they have roughly the same QoS, and hence at the global selection phase it is very probable to find alternative service compositions.

Overall, our selection approach presents two main advantages. First, as explained above, it allows high flexibility when choosing services at the dynamic binding stage. Second, it considerably reduces the complexity of the problem. In the next section, we study in detail the computational complexity of our algorithm.

3.4 Computational Complexity Analysis

In this section, we give a detailed analysis of the computational complexity of our QoS-aware service selection algorithm. The analysis concerns both phases of our algorithm (i.e., local selection and global selection). It will be based on the following parameters:

Z = the number of activities in the composition,
P = the number of QoS properties,
K = the number of clusters.

Due to the fact that the local selection algorithm makes extensive use of K-means, its performance is strongly dependant on the performance of this algorithm. As already explained, in the local selection phase we cluster services for each QoS property and for all the activities in the composition. For this reason, the computational complexity of our local selection algorithm is of $O\ (Z.P.\Delta)$ where Δ is the complexity of K-means. Δ is defined as $O\ (N^{P.K+1}\ \log\ N)$ where N is the number of services to cluster [Lloyd, 1957].

Concerning the global selection phase, its computational complexity is related to two main facts. First, we explore service compositions while changing only one service when switching from a service composition to another. Second, each service we switch to is investigated once. That is, it is not investigated again with another combination of services. According to this, our global selection phase comprehends $N - Z + 1$ steps, each dealing with a service composition, where N is the total number of services associated with all the activities in the user task. Then, the computational complexity of our global selection algorithm is of $O\ (N{-}Z{+}1)$. Accordingly,

our approach reduces considerably the computational complexity of service selection under global QoS requirements, which is NP-hard. Even more, our approach has a lower computational complexity compared to existing heuristic algorithms for QoS-aware service selection. Further details about this subject are given in Section 5.

4 Distributing QASSA

The version of QASSA presented in Section 3 assumes the presence of a centralized and stationary infrastructure supporting QoS-aware service composition. Nevertheless, within pervasive environments, it is not always possible to assume the support of such infrastructure. QoS-aware service composition in pervasive environments (i.e., more specifically selection algorithms) rather relies on *ad hoc* environments with no infrastructure support.

For this reason, we aim at making the QASSA algorithm capable of operating on top of *ad hoc* infrastructures. We present a distributed version of QASSA, which can be executed in *ad hoc* environments populated by mobile resource-constrained devices.

As depicted in Figure IV.4, our distributed algorithm starts form the assumption that pervasive environments are populated by mobile and resource-constrained devices, which are able to communicate in a peer-to-peer fashion. Devices are mobile, so that the environment changes dynamically according to the emergence or departure of devices. Thus, it is hard to assume a stationary infrastructure in *ad hoc* pervasive environments. Our distributed selection algorithm is implemented within a QoS-aware service-oriented middleware that is supposed to be installed on each device operating in the environment. We also assume that our middleware deals with preliminary phases required for QoS-aware composition such as service discovery in *ad hoc* pervasive environments [Raverdy et al., 2006].

Our distributed service selection algorithm allows for fulfilling service selection as a synergetic interaction between the user device (referred to as requester) and other devices available in the environment (referred to as helpers). The main idea of our distributed algorithm is to fulfill local selection for each abstract activity in the user task using a helper, thus enabling to execute the whole local selection phase using several helpers simultaneously. After that, the requester collects the local selection results and performs the global selection phase on the user device.

When a user submits a QoS-aware composition request, the middleware platform installed on his/her device processes the request through the following steps (Algorithm 3):

1. The middleware analyzes the composition request and splits it into several elementary

input : a set of activities $T = \{A_1, .., A_z\}$ of size z,
a set of services $S_i = \{s_{i,1}, .., s_{i,v_i}\}$ of size v_i for each activity A_i $(i \in [1, z])$,
a set of QoS properties $P = \{p_1, .., p_n\}$ of size n,
a set of QoS constraints $U = \{u_1, .., u_n\}$ imposed on each property,
a set of weights $W = \{w_1, .., w_n\}$ of size n associated with each QoS property,
a QoS vector $QoS_{s_{i,k}} = \langle q_1, .., q_n \rangle$ for each service $s_{i,k}$ $(k \in [1, v_i])$.

output: a set of service compositions ranked according to their QoS utilities.

1 **begin**
2 *(Step 1) Splitting the user request into a set $\mathcal{E}$ of elementary requests*;
3 **foreach** $A_i \in T \quad (i \in [1, z])$ **do**
4 $e_i \leftarrow (A_i, S_i, P, W)$;
5 $\mathcal{E} \leftarrow \mathcal{E} \cup e_i$;
6 **end**
7 *(Step 2) Broadcasting help message and getting helpers*;
8 broadcast (help message);
9 **foreach** *device d favorably replying to the help message* **do**
10 helpers $\leftarrow$ helpers $\cup$ d;
11 **end**
12 **while** $\mathcal{E} \neq \emptyset$ **do**
13 *(Step 3) Scheduling elementary requests to helpers*;
14 **foreach** $e_i \in \mathcal{E}$ **do**
15 $e_i \leftarrow d \qquad (d \in$ helpers$)$;
16 **end**
17 *(Step 4) Fulfill the local selection phase*;
18 **while** *Timeout Session* **do**
19 **Helper Algorithm** : execute local selection (Algorithm 1) given e_i as input;
20 **end**
21 *(Step 5) Getting the results of elementary requests*;
22 **foreach** $e_i \in \mathcal{E}$ **do**
23 **if** *result* $(e_i) \neq$ *null* **then**
24 $S'_i \leftarrow$ result (e_i);
25 $\mathcal{E} \leftarrow \mathcal{E} \setminus \{e_i\}$;
26 **end**
27 **end**
28 **end**
29 *(Step 6) Fulfill the global selection phase*;
30 execute global selection (Algorithm 2);
31 **end**

Algorithm 3: Overview of the distributed service selection algorithm from the *requester* point of view. The coloured box concerns the part executed on the *helper* side.

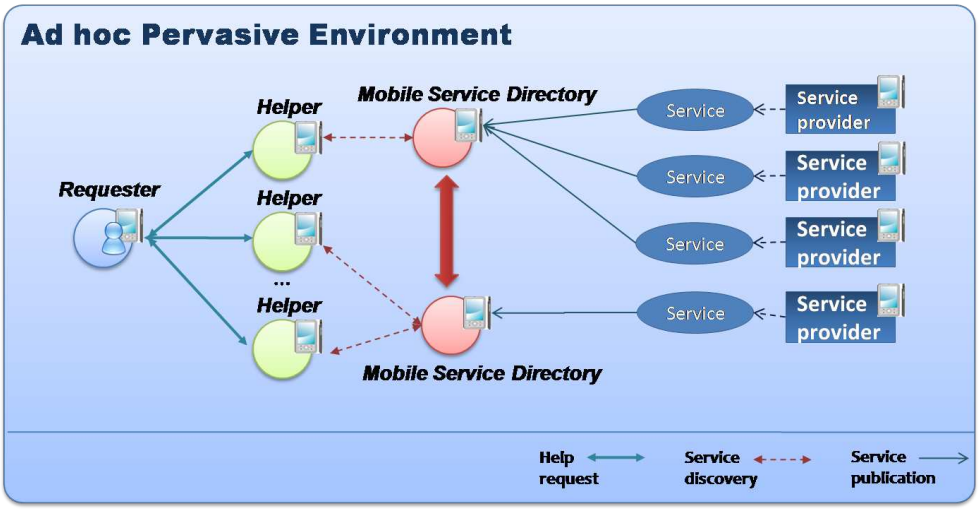

Figure IV.4 – Ad hoc pervasive environments

requests each dealing with service discovery and local selection for an abstract activity in the user task.

2. The requester device broadcasts a help message asking devices available in the environment for collaboration in order to fulfill a QoS-aware service composition request. The other devices can favourably reply to the message if they have free resources.
3. The requester schedules elementary requests with respect to the number of helper devices that accepted to participate in fulfilling the user task.
4. The requester waits up to a timeout for helpers to process their associated elementary requests. Each helper interacts with service directories available in the environment in order to discover services able to fulfill the elementary request. Once service discovery is achieved, the helper proceeds to local selection based on Algorithm 1, and it returns the selected services to the requester.
5. If the time-out expires without all the results available, the requester re-schedules the failed elementary requests.
6. When the requester acquires all services resulting from the local selection phase, it proceeds to global selection based on Algorithm 2.

The global service selection is difficult to carry out in a distributed way because it requires a global vision of QoS information and the structure of the composition [Li et al., 2010]. Additionally, it typically requires a resource-rich device, given the computational complexity of the problem. In our approach, we propose a global service selection algorithm with low computational complexity (as detailed in Section 3.4), thus it can be carried out using only

the resource-constrained device of the requester. The timeliness of our distributed algorithm is further validated by experimental results (see Chapter VI).

Finally, it is worth noting that our approach copes with further issues of distributed QoS-aware service selection, notably data privacy. Indeed, pervasive computing advocates open *ad hoc* environments which offer access to any user, hence there are no guarantees about the trustworthiness of other users operating in the environment. For this reason, user data must be protected while carrying out QoS-aware service selection in a distributed way. Towards this purpose, we recall the solution provided by Cardoso [Cardoso, 2009], which allows for carrying out distributed semantic-based service discovery and composition in pervasive environments while preserving data privacy.

5 Evaluation and Discussion

In this section, we study QoS-aware service selection algorithms proposed in the service-oriented community in general, and not only in the context of QoS-aware service-oriented middleware (as presented in Chapter II). This represent a broader field of study with many interesting heuristic algorithms. Below, we discuss the contribution of QASSA compared to existing heuristic algorithms.

Yu et al. [Yu et al., 2007] present two heuristic algorithms, WS-HEU and WFlow, to resolve the QoS-aware service selection problem. WS-HEU is specific heuristic algorithm applied to sequential workflows (i.e., workflows structured as a sequence of activities), whereas WFlow is designed for general workflow structures (i.e., sequential, conditional, parallel). The main idea of WFLow is to decompose workflows into multiple execution routes. WFlow considers a parameter ξ_i for every route indicating its probability to be executed, and it then focuses on the route with the highest probability to be executed. Therefore, WFlow can give good results for the most probably executed routes, however, when a different route is executed, the algorithm may give less good results. In QASSA, we use different QoS aggregation approaches (i.e., pessimistic, mean-value and optimistic) to evaluate the overall QoS of a workflow, and select near-optimal service compositions regardless of the way the workflow will be executed.

Likewise, Comes et al. [Comes et al., 2010] present two heuristic algorithms for QoS-aware service selection: OPTIM_HWeight and OPTIM_PRO. Both algorithms represent the selection problem as browsing a tree of abstract activities with multiple concrete services for each activity. To select near-optimal service compositions, OPTIM_HWeight uses a function $f_{HWeight}$ which sorts service candidates with respect to their influence on the overall QoS of the composition.

This function reflects jointly all QoS properties of services, thus it suffers from the problem of compensation between QoS properties. Whereas, OPTIM_PRO uses the probability of executing a node of the tree multiplied by the number of execution loops as a heuristic to guide the selection of services. This idea starts from the assumption that the nodes executed more often receive a higher importance. Similar to WFlow, OPTIM_PRO may give good results for the most probably executed branches of the tree, but also less good results when a different branch is executed.

Other QoS-aware service selection approaches present solutions based on evolutionary algorithms which browse service compositions in a random way. These solutions can be divided into three sets. A first set of solutions [Canfora et al., 2005; Cao et al., 2007; Gao et al., 2007; hong SHEN and hu YANG, 2010; Jaeger and Mühl, 2007; Jiang et al., 2011; Kobti and Zhiyang, 2007; Lécué, 2009; Vanrompay et al., 2008; Zhang et al., 2006] use the genetic algorithm (GA) to fulfill QoS-aware service selection. A second set of solutions use other evolutionary algorithms such as the Harmony Search (HS) algorithm [Jafarpour and Khayyambashi, 2010] and the Ant Colony Algorithm (ACA) [Xia et al., 2008]. Whereas, a third set of solutions propose combining GA with other algorithms, such as the Ant Colony Algorithm [Zongkai YANG and ZHAO, 2010] and the Tree Traversal Sequence (TTS) coding scheme [Kai Shuang and Su, 2009]. Although these algorithms may produce satisfying results, their application to the service selection problem presents two major drawbacks. First, the order in which service compositions are checked is randomly chosen, whereas in QASSA we check services in an ordered way to optimize the timeliness and the accuracy of our algorithm. Second, these algorithms can run endlessly, thus developers have to set a termination condition, such as the maximum number of iterations in order to stop the algorithm. However, even setting a high number of iterations gives no guarantee about the quality of the result.

A more recent trend in QoS-aware service selection algorithms [Alrifai et al., 2008, 2010; Ben Mabrouk et al., 2009; Liu et al., 2009] consists in combining local and global selection techniques similarly to QASSA. A first research effort in this context is proposed by Alrifai et al. [Alrifai et al., 2008]. The authors present a selection algorithm that starts from the global level and resolves the selection problem at the local level. Indeed, they proceed by decomposing global QoS constraints (i.e., imposed by the user on the whole composition) into a set of local constraints (i.e., for individual sub-tasks, parts of the whole composition). To do so, the algorithm uses MILP techniques to find the best decomposition of QoS constraints. The main drawback of this approach is that it relies on a greedy method for the decomposition of QoS constraints, thus yielding strict constraints discriminating a lot of service candidates.

To cope with this issue, the same authors present another approach [Alrifai et al., 2010]

combining local and global selection techniques in another way. The authors start by the local selection phase. They use two techniques to reduce the number of services investigated for each abstract activity in the user task. First, they use the *skyline* concept [Börzsönyi et al., 2001] as a technique to determine the most interesting services in terms of QoS. Once skyline services are determined, the authors cluster them into several clusters using K-means, and then they select a representative service for each cluster. At the global level, the authors compose the representative services selected at the local level, and check whether the composition meets global QoS requirements using MILP. This approach also presents several drawbacks. Concerning the algorithm itself, the authors claim finding the optimal service composition, because they assume that skyline services are the best services in terms of QoS, which is not true. Indeed, a skyline service is a service which has the highest (i.e., the best) value for one or more QoS properties, whereas for the remaining QoS properties it may have very low values. Regarding this definition, it is possible that a non-skyline service with high values (and not the highest) for all QoS properties yields a higher overall QoS than a skyline service. Additionally, the fact of using MILP to resolve the global selection phase restricts the usage of the approach, since MILP supports only additive QoS properties. Concerning the performance of the algorithm, the authors execute K-means $Z.(N/2)$ times, where N is the number of service candidates investigated for the user task, which represents a high number of iterations, especially when it deals with a high number of service candidates. In our approach, we execute K-means $Z.P$ times where P is the number of QoS properties, which is always limited compared to the number of service candidates. Additionally, at the global selection phase, the authors execute MILP iteratively until a near-optimal composition is found. In each iteration, the set of representative services with the highest QoS utilities is investigated. This approach may end by executing MILP N times, where N is the number of representative services, which may represent also a high number of iterations when it deals with a high number of representative services.

Another approach combining local and global selection techniques is presented by [Li et al., 2010]. Similar to [Alrifai et al., 2008], the authors decompose global QoS constraints into local constraints using MILP. Based on the local QoS constraints, they select services for each abstract activity in the user task. Then, they compose locally selected services and check whether the composition meets global QoS constraints, using MILP again. The main advantage of this approach is that it executes local selection in a distributed way similarly to our approach. However, they decompose global QoS constraints based on the average of QoS values associated with each abstract activity, which is not accurate and may discriminate a number of service candidates. A similar approach is presented by Jin et al. [Jin et al., 2010]. The authors decompose global QoS constraints into local constraints using MILP, then they perform local

selection. The main shortcoming of this approach is that, when they compose locally selected services, they do not check whether the resulting composition meets global QoS requirements.

A more interesting approach is presented by Liu et al. [Liu et al., 2009]. The authors propose a QoS-aware service selection algorithm which also combines local and global selection techniques. They use the *convex hull* concept [Mostofa Akbar et al., 2006] as a local selection technique. At the global level, the authors randomly establish an initial composition, and they try to enhance it using services selected by the convex hull. The main drawback of this approach is that it closely depends on the initial composition.

Much recent approaches dealing with QoS-aware service selection in the context of pervasive computing [Chang and Lee, 2009; Dutra and Junior, 2010] address the problem of compensation between QoS properties. To cope with this issue, Chang and Lee [Chang and Lee, 2009] propose a strict ordering of QoS properties with respect to their priority (from the software developer's point of view). Based on this ordering, the authors use PROMETHEE (Preference Ranking Organization METHod for Enrichment Evaluations), a well-known MCDM (Multi-Criteria Decision Making) to perform pair-wise comparisons repeatedly, and select the set of services associated with the most important QoS properties. Nevertheless, the proposed ordering is static, and it does not consider user preferences in terms of QoS. An alternative approach is proposed by Dutra and Junior [Dutra and Junior, 2010]. The authors propose a hierarchical classification of QoS properties using ADAPTREE, which is an adaptive decision tree algorithm. Based on the established classification of QoS properties, they select services providing the highest values of the most important QoS properties. The main drawback of both aforementioned approaches (i.e., [Chang and Lee, 2009; Dutra and Junior, 2010]) is that they do not consider selection under global QoS requirements. They rather perform greedy selection for each abstract activity in the user task.

In comparison with all the above analyzed research approaches, QASSA addresses service selection under global QoS requirements while coping with the problem of compensation between QoS properties. To do so, we combine local and global selection techniques while using one-dimensional K-means clustering as a local selection technique (instead of skyline and convex hull techniques). The main advantage of our algorithm is that it considers jointly: (i) end-to-end QoS requirements, (ii) both advertised and run-time QoS of services, (iii) supporting dynamic binding and adaptation of service compositions by selecting several alternative compositions instead of only one, and (iv) distributivity. To the best of our knowledge, there is no other approach addressing jointly the above requirements. Moreover, our algorithm is of low computational complexity, which makes it suitable for application to interactive pervasive environments. Chapter VI further confirms the efficiency of our algorithm based on experimental

results.

Chapter V

QoS-driven Composition Adaptation in Pervasive Environments

In the previous chapter, we proposed a QoS-aware service composition approach allowing to build complex services meeting the functional and QoS requirements of users. Nevertheless, the proposed approach is not sufficient to ensure meeting QoS requirements during the execution of service compositions, because services' QoS may vary at run-time. To cope with this issue, service compositions have to be dynamically adapted with respect to QoS fluctuations. By adaptation we refer to the ability to alter service compositions in response to changes impacting their execution. In particular, we focus on changes having impact on the QoS produced by service compositions, notably (i) changes in QoS requirements imposed on the composition (i.e., generally associated with long running service compositions), and (ii) changes in services' QoS due to the dynamics of pervasive environments (e.g., user mobility).

To cope with the aforementioned changes, we present a QoS-driven adaptation approach for service compositions that is based on two adaptation strategies. First, service substitution, which consists in replacing services forming the composition with alternative ones.

When service substitution fails, we proceed with the second adaptation strategy, which consists in adapting the behaviour according to which the user task is carried out. This strategy is based on the concept of Task Class that we introduce to define the set of abstract service compositions functionally equivalent (i.e., allowing to achieve the same user task) but having different behaviours. Based on the Task Class concept, we reduce the behavioural adaptation problem to *vertex disjoint subgraph homeomorphism* [Xiao et al., 2007].

The behavioural adaptation strategy represents a key contribution of this thesis. For this reason, the major part of this chapter is devoted to this strategy. Next, we present the baseline of our QoS-driven adaptation approach (Section 1). Then, we give the background and definitions underpinning our behavioural adaptation strategy (Section 2). After that, we concentrate on our

behavioural adaptation strategy. We first give an overview of this strategy (Section 3). Then, we show how to transform the user task into a behavioural graph (Section 4). After that, we introduce the Task Class concept, which specifies the set of functionally equivalent behavioural graphs (Section 5). Based on this concept, we show how to reduce the behavioural adaptation problem to *vertex disjoint subgraph homeomorphism* [Xiao et al., 2007] (Section 6). Finally, we discuss the features of our approach with respect to related work in the service-oriented community in general (Section 7).

1 Approach Baseline

In this section, we first present the QoS monitoring approach underpinning both adaptation strategies (i.e., service substitution and behavioural adaptation). Then, we briefly sketch an overview of the service substitution strategy (Section 1.2).

1.1 Global and Proactive QoS Monitoring

QoS monitoring is a primary requisite of QoS-driven composition adaptation in pervasive environments, since it allows for determining the run-time QoS of services, thus enabling to detect QoS requirements' violation and trigger adaptation actions.

Nevertheless, QoS monitoring is generally carried out at the scope of individual services forming QoS-aware compositions. Such an approach of QoS monitoring provides an incomplete view of QoS at run-time since it does not cover the whole composition, thus it cannot control and ensure global QoS requirements imposed on QoS-aware compositions.

Additionally, QoS monitoring is generally enacted along with the invocation of services forming the composition. Therefore, QoS monitoring does not support early detection of services providing unsatisfactory QoS. In this section, we identify two main features leveraging QoS monitoring approaches to cope with the aforementioned issues.

First, QoS monitoring should be enacted *a priori*, i.e., just after the selection of services and before their binding and invocation. We refer to this approach as proactive QoS monitoring. It allows for determining the run-time QoS of all the services selected by the QASSA algorithm (detailed in Chapter IV). Based on this anticipated QoS monitoring, dynamic binding can choose the best service (in terms of QoS) to be enacted for each abstract activity in the user task. Hence, services providing unsatisfactory QoS are early detected and discarded. However, we note that proactive QoS monitoring is not always possible to accomplish, depending on whether the services are invoked by other clients, and if they allow to be monitored by third

parties.

Second, QoS monitoring should be performed at the level of the whole composition. We refer to this as global QoS monitoring. It can be carried out by aggregating the run-time QoS of basic services participating in the composition [Comes et al., 2009]. Global QoS monitoring provides a holistic view of the running service composition and assesses its status with respect to global QoS requirements. Global QoS monitoring enables predicting the violation of QoS requirements and taking anticipated adaptation actions (e.g., re-composition). Indeed, given QoS requirements of the user, and given the run-time QoS of the executed services and those to be executed, global QoS monitoring can determine whether the run-time QoS of the overall composition can meet QoS requirements. If not, it triggers adaptation actions. In accordance with the above, our adaption approach assumes a global and proactive QoS monitoring method (e.g., KAMI [Epifani et al., 2009]).

1.2 Service Substitution

Service substitution is triggered when one or more running services provide unsatisfactory QoS. It consists in substituting these services using alternatives available in pervasive environments. Alternative services can be determined in two ways. First, based on those services previously selected by the QASSA algorithm (detailed in Chapter IV). If these services are no longer available or they also provide unsatisfactory QoS, a second solution may be carried out. It consists in performing QoS-aware service discovery for the failed activity (or activities). For each activity, the discovered services must provide a QoS that is at least equal to the worst service previously selected for the considered activity.

Service substitution based on QoS-aware discovery is more suitable when we deal with long running user tasks and highly dynamic pervasive environments, so that: (i) new services may join the environment during the execution of the user task, and (ii) the duration of service discovery is negligible compared with the execution time of the user task.

Service substitution is not enough to cope with QoS fluctuations in pervasive environments since it is not always possible to find substituting services. In this case, it may be possible to find other abstract compositions that can fulfill the user task, but behaving differently, and for which fitting services can be found in the environment. In this chapter, we investigate this idea to leverage QoS-driven composition adaptation with a behavioural adaptation strategy. Below, we present the theoretical background and definitions underpinning this strategy.

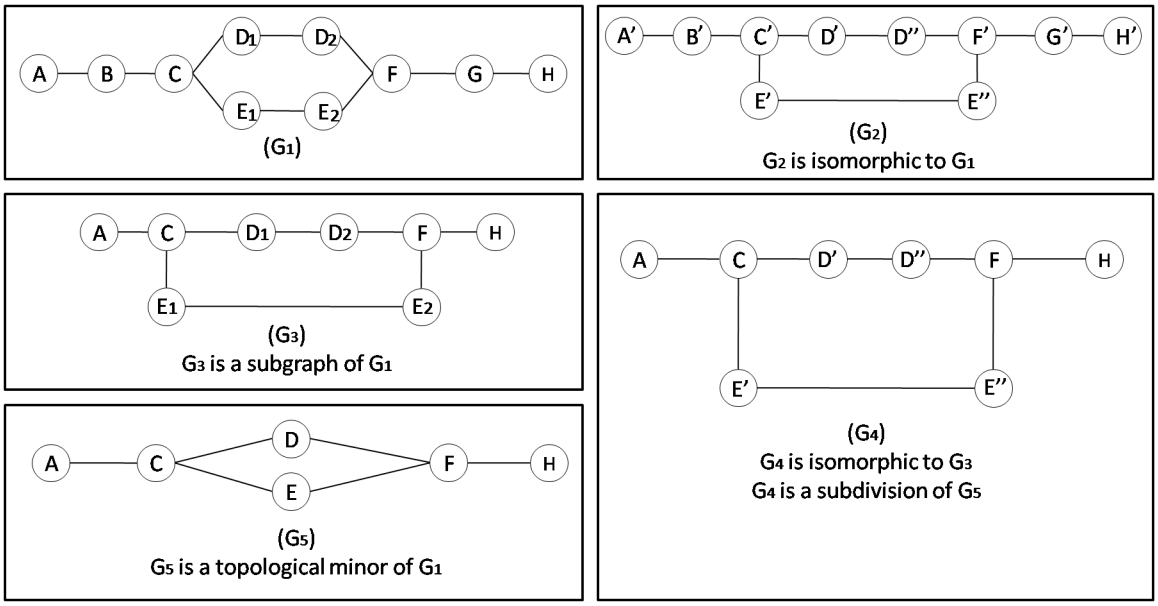

Figure V.1 – Illustrating graph definitions

2 Background and Definitions

Our behavioural adaptation strategy is based on the notion of *vertex disjoint Subgraph Homeomorphism (vdSH)* [Xiao et al., 2007]. vdSH allows for defining the similarity between subgraphs where vertex skipping is allowed. vdSH works on labelled undirected graphs, whereas the user tasks (i.e., which we represent as graphs) have known directions. To enable vdSH determination, we transform the user tasks into undirected graphs by adding start and end vertices. Further details about this transformation are given in Section 4. To explain the notion of vdSH, we give the following graph theory definitions [Diestel, 2005]. Let L denote a set of labels.

Definition 1 *An undirected labelled graph G is a tuple $G = (V, E, \mu)$, where*

- *V is the set of vertices,*
- *$E \subseteq V \times V$ is the set of edges,*
- *$\mu : V \rightarrow L$ is a function assigning labels to vertices.*

We use the undirected labelled graph defined above to specify abstract compositions representing the user tasks. In this definition, vertices represent abstract activities, composition patterns as well as the start and final nodes of the user task. The edges are not labelled since it is not useful in our approach.

Definition 2 ***Graph isomorphism***
Given two graphs $G = (V, E, \mu)$ and $G' = (V', E', \mu')$. A bijective function $\delta : V \rightarrow V'$ is a graph isomorphism from G to G' if:
For any edge $e = (v_1, v_2) \in E$, there exists an edge $e' = (\delta(v_1), \delta(v_2)) \in E'$.

Informally, graph isomorphism enables expressing the functional equivalence between two abstract compositions having the same set of abstract activities, but differing in the order according to which the activities are executed.

Definition 3 ***Vertex and graph subdivision***
Let $G = (V, E, \mu)$ be a graph and $v \in V$.
The subdivision of v is obtained by decomposing v into two vertices v_1 and v_2, all edges having v as destination will have v_1 as destination, where all edges having v as source will have v_2 as source, and an edge will be added between v_1 and v_2.
A subdivision of G is any graph which can be obtained by recursively subdividing vertices in G.

Starting from the assumption that G represents an abstract composition, the vertex subdivision means that a coarse-grained abstract activity is replaced by two finer-grained activities. The subdivision of G represents then an alternative abstract composition allowing to achieve the user task based on finer-grained abstract activities.

Definition 4 ***Subgraph***
Given a graph $G = (V, E, \mu)$, a subgraph of G is a graph $G_s = (V_s, E_s, \mu_s)$ such that:

1. $V_s \subseteq V$
2. $E_s = E \cap V_s \times V_s$
3. μ_s *is a restriction of* μ, *i.e.,* $\mu_s(v) = \begin{cases} \mu(v) & \text{if } v \in V_s \\ undefined & otherwise \end{cases}$

In our approach, we use the subgraph to represent the part of an abstract composition that is not executed when a failure of the composition occurs.

Definition 5 ***Topological minor***
A graph M is called a topological minor of a graph G if there exists a graph M' such that:
- *M' is a subgraph of G*

– *M' is isomorphic to some subdivision of M.*

We use the notation $M \lessapprox G$ to indicate that M is a topological minor of G.

Definition 6 ***Independent paths***

Given a graph $G = (V, E, \mu)$, a path P in G is a sequence of vertices $v_1, v_2, ..., v_k$, where $v_i \in V$ and $(v_i, v_{i+1}) \in E$ for $1 \leq i < k$. The vertices v_1 and v_k are linked by P and are called its ends. A path is simple if its vertices are all distinct. Particularly, a group of paths in G are independent if none of the paths has an inner vertex in another path.

Definition 7 ***vertex disjoint Subgraph Homeomorphism (vdSH)*** *Given two graphs G and M, if M is a topological minor of G, then there exists a corresponding vertex disjoint subgraph homeomorphism from M into G, which is a pair of injective mappings (α, β) from M into G [Xiao et al., 2007]. α is an injective mapping from the vertex set of M into that of G, and β is an injective mapping from edges of M into simple paths of G such that: (i) for each $e(v_1, v_2) \in E(M)$, $\beta(e)$ is a simple path in G with $\alpha(v_1)$ and $\alpha(v_2)$ as two ends, and (ii) all mapped paths are pairwise independent, i.e., they have disjoint inner vertices (only the vertices of the extremity can be common to several paths) [Xiao et al., 2007].*

Literally, vdSH determination is performed by mapping each edge of M to an independent path in G having the same extremity vertices.

3 Behavioural Adaptation Strategy: Overview

Our behavioural adaptation strategy consists in carrying out the user task using an alternative abstract composition allowing to achieve the required user task.

To determine alternative abstract compositions, we introduce the concept of Task Class, which defines abstract compositions that are functionally equivalent, i.e., abstract compositions allowing to achieve the same user task. The task Class concept starts from the idea that the user task can be fulfilled in different ways. These ways can be obtained either by changing the order in which abstract activities are executed, or by dividing coarse-grained abstract activities into fine-grained activities, respectively by merging fine-grained activities into coarse-grained abstract activities.

To motivate our approach, we recall Bob's shopping scenario. Let us assume the case where Bob performs the first activity in the task, which is about ordering a book. Meanwhile, the monitoring service notifies that the selected services for the activities *ordering an MP3 player and headphones* and *ordering food and a drink* are no longer available (Figure V.2), e.g., because

they are overloaded. To cope with this situation, we have to replace the non-executed sub-composition (i.e., activities that are not yet executed; henceforth we refer to them as failed sub-composition) with an alternative sub-composition behaving differently, but functionally equivalent.

To do so, we propose to explore the task class associated with the user task and analyse available alternative abstract compositions. More specifically, we aim at finding a sub-composition that is functionally equivalent to the failed one, but behaving differently.

To enable the behavioural comparison of abstract compositions, we transform abstract compositions into graphs. Graphs represent indeed a powerful and universal data structure that can be used for multiple purposes, including the specification of abstract compositions' behaviours. Thus, the problem of comparing abstract sub-compositions can be formulated as a search for correspondences between subgraphs. Such correspondences are generally established by subgraph isomorphism detection (e.g., [Messmer and Bunke, 2000]). Nevertheless, subgraph isomorphism can simply determine exact matching between subgraphs, where only the order of abstract activities changes. In our approach, we do not consider subgraph isomorphism, because when the service substitution fails, changing only the order in which abstract activities are executed does not provide any alternative solution.

For this reason, we rather focus on inexact and flexible subgraph matching where vertex subdivision is supported. To address this issue, we propose using the concept of *topological minor* [Xiao et al., 2007], which enables subgraph matching with vertex subdivision. Using the Graph Minor theory [Robertson and Seymour, 1995], the relation between a subgraph and its topological minor can be described as *vertex disjoint Subgraph Homeomorphism (vdSH)* (also known as node disjoint Subgraph Homeomorphism [Xiao et al., 2007]).

Topological minors allow for capturing the high-level topological structure of a subgraph while tolerating vertex subdivision. Related to this, vdSH allows for matching two subgraphs and checking whether they have roughly the same structure while tolerating vertex subdivision. Nevertheless, having roughly the same structure is not enough to state that two subgraphs (representing two abstract sub-compositions) are functionally equivalent. We further need to ensure that: (i) abstract activities of both subgraphs cover the same set of functionalities, and (ii) both subgraphs consume the given inputs and produce the required outputs. Related to this, we formulate our behavioural adaptation approach as a two-phase process (Figure V.2):

1. We first perform two preliminary verifications. The first verification consists in establishing a vertex mapping between the failed subgraph and each investigated subgraph. This verification ensures that both subgraphs cover the same functionalities, thus allo-

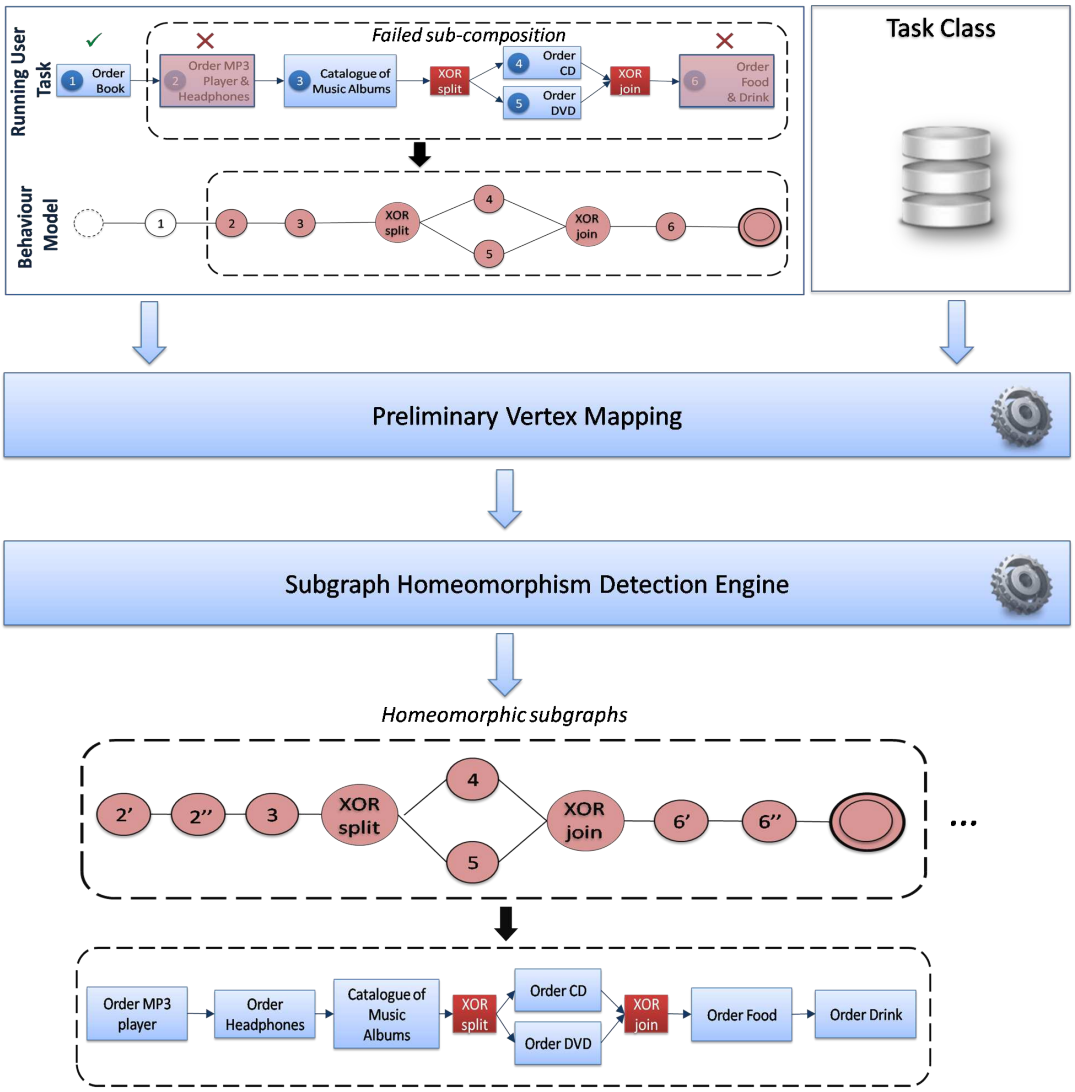

Figure V.2 – Overview of the behavioural adaptation strategy

wing to discard subgraphs having more or less functionalities (not defined in the failed sub-composition). The second verification concerns ensuring that both subgraphs consume the given inputs and produce the required outputs. The former verification (i.e., vertex mapping) can be carried out offline or online, i.e., when an abstract composition is added to a task class, or just before performing vdSH determination, respectively. Further details about both preliminary verifications are given in Section 6.1.

2. Second, we carry out vdSH determination based on the algorithm defined in [Xiao et al., 2007]. vdSH determination allows for ensuring that both subgraphs have similar structures while tolerating vertex subdivision. We further propose some extensions to the vdSH

determination algorithm in order to preserve the functional semantics of the investigated subgraphs (i.e., sub-compositions). These extensions are mainly about (i) supporting semantic matching of vertices, and (ii) preserving data constraints, during behavioural adaptation. These extensions are detailed in Section 6.2.

Next, we show how to transform the abstract compositions (representing user tasks) into behavioural graphs (Section 4). Based on behavioural graphs, we introduce the concept of task class, which encompasses abstract compositions that are functionally equivalent (Section 5). After that, we address the detail of our behavioural adaptation strategy (Section 6).

4 From a User Task to a Behavioural Graph

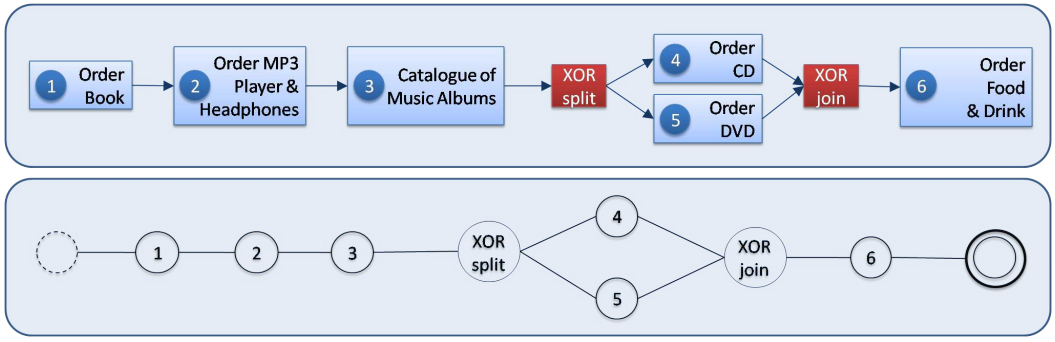

Figure V.3 – Bob's shopping task and its equivalent labelled graph

As already introduced in the previous chapter, a user task $T = \langle A_1, .., A_z \rangle$ is defined as a set of abstract activities coordinated using composition patterns (i.e., Sequence, AND, XOR). Each activity is defined by a function f, a set of inputs I and a set of outputs O. To enable behavioural adaptation, we transform the user task into a behavioural graph [Liu et al., 2008], which describes the evolution of the user task execution in time. Formally, the behaviour of the user task can be defined as an undirected labelled graph. Let s, F, V_A and V_P denote respectively, the start vertex, the set of final vertices, the set of vertices associated with abstract activities and the set of vertices associated with composition patterns in the user task.

Definition 8 *A user task is an undirected labelled graph* $G_T = (V, E, \mu)$*, where*

- V *is the set of vertices;* $V = s \cup F \cup V_A \cup V_P$*,*
- $E \subseteq V \times V$ *is the set of edges,*
- $\mu : V \rightarrow L$ *is a function assigning labels to vertices,*

$$\mu = \begin{cases} (f, I, O) & if\ v \in V_A \\ p & if\ v \in V_P \\ s & if\ v = s \\ \epsilon & if\ v \in F \end{cases}$$

where (f, I, O) defines the label of elements which are associated, respectively, with the function, the set of inputs and the set of outputs of an abstract activity, and $p \in$ {Sequence, AND_split, AND_join, XOR_split, XOR_join, Loop} denotes the label of a composition pattern.

To simplify the graph representation, we replace the Sequence composition pattern by a simple edge. Whereas for the loop composition pattern, we transform each loop activity in the user task with a simple activity with an index k (Figure V.4), where k is the number of loops.

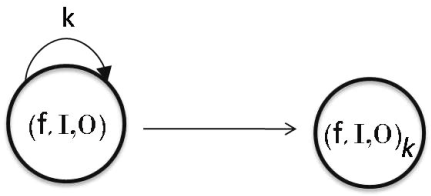

Figure V.4 – Simplifying loop activities

5 The Task Class Concept

We introduce the novel concept of Task Class, which enables behavioural adaptation of service compositions in pervasive environments. We first give an overview of this concept, then we define it in a formal way.

5.1 Overview

The Task Class concept starts from the observation that the user task can be carried out in different ways. In practice, the abstract composition identified by the user represents only one possible way to achieve the intended task. To illustrate this idea, let us recall the pervasive shopping scenario, in which Bob wants to carry out a shopping task composed of the following activities: (1) buying a book, (2) getting the catalogue of MP3 players and headphones, (3)

ordering an MP3 player and headphones, (4) buying either a DVD or a CD of a given music album, (5) getting food menus, and finally (6) ordering food and a drink.

As depicted by Figure V.5, Bob's shopping task can be fulfilled using various abstract compositions. These alternative compositions can be defined using two main methods:

1. By changing the order in which abstract activities are carried out, still respecting to data constraints (data constraints are detailed in Section 6.2) ;
2. By replacing one or more abstract activities with other activities having different granularity (i.e., either finer or more coarse granularity).

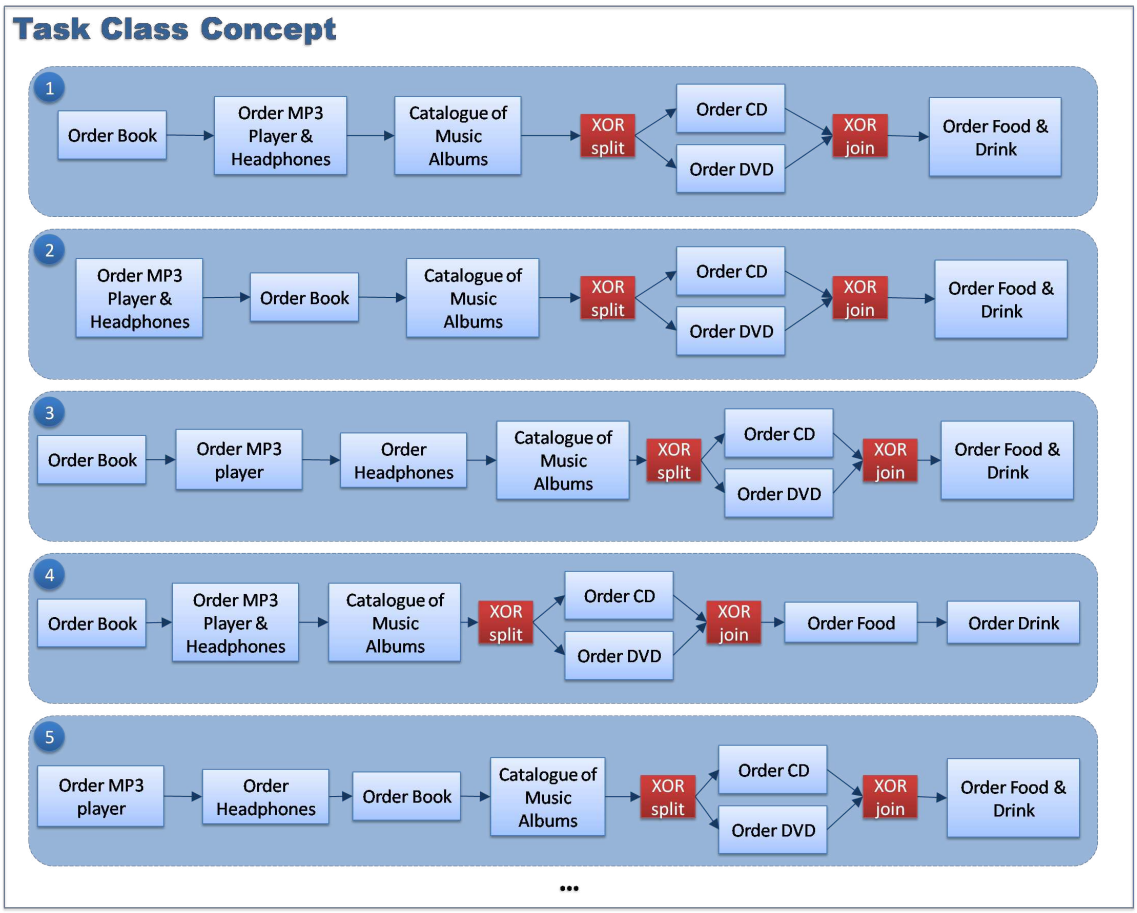

Figure V.5 – Illustrating the Task Class concept

Figure V.5 illustrates the aforementioned methods of generating alternative abstract compositions allowing to realize the user task. Assuming that composition 1 is the abstract composition defined by the user, composition 2 shows how to change the order in which abstract activities are executed, e.g., ordering the book is postponed after ordering the MP3 player and headphones. Abstract compositions 3 and 4 depict how to replace a coarse-grained activity

(e.g., ordering an MP3 player and headphones, respectively ordering food and a drink) with finer-grained activities (e.g., ordering the MP3 player then ordering the headphones, respectively, ordering the food then ordering the drink). Finally, the abstract composition number 5 combines both methods, e.g., ordering the MP3 player and headphones is divided into two activities and ordering the book is executed subsequently.

Based on the above description, we can state that the task class concept represents a set of abstract compositions which are functionally equivalent, i.e., they allow to achieve the same user task. In this section, we aim to profit from the full functional potential enabled by the task class concept to propose further solutions for service composition adaptation in pervasive environments. Our approach starts from the assumption that task classes are built progressively based on learning techniques and user feedbacks. More specifically, each time the user defines a new task T, our middleware platform (installed on the user device) uses planning techniques to automatically generate alternative abstract compositions allowing to achieve the required task T. Later, when users ask for the task T, our middleware platform proposes the generated abstract compositions to the users and gets their feedback. If an alternative abstract composition is accepted and validated by users, it is added to the same task class as T.

Task classes and their underlying abstract compositions are further used to assist users defining their required tasks. Users can choose the way their tasks are executed among various abstract compositions. Based on the abstract composition specified by the user, we carry out our approach of QoS-aware service composition in pervasive environments presented in the previous chapter.

At run-time, if the produced composition provides unsatisfactory QoS and the service substitution fails, we proceed to behavioural adaptation. To explain this adaptation strategy, next we define the Task Class concept in a formal way. Based on this definition, we express the behavioural composition adaptation as a subgraph homeomorphism problem.

5.2 Formal Definition

Formally, the task class concept can be defined as follows:

Definition 9 ***Task Class***
Let $G_T = (V, E, \mu)$ be the graph representing the user task. A task class is a set of graphs $\boldsymbol{\mathcal{G}} = \{G_1, .., G_n\}$ where for each graph $G_i = (V_i, E_i, \mu_i)$ ($i \in [1, n]$):

1. *G_i is isomorphic to G_T, or*
2. *G_i is a subdivision of G_T, or*

3. G_T is a subdivision of G_i.

This definition means that the task class concept comprehends abstract compositions which are obtained either by changing the order in which the activities are executed, or by dividing one or more coarse-grained activity into finer-grained activities, or by merging one or more fine-grained activities into coarse-grained activities. The statement number (4) in the task class definition means that the task class may further comprehend abstract compositions which are obtained by simultaneously: (i) dividing one or more coarse-grained activities in the user task into finer-grained activities and (ii) merging one or more fine-grained activities into coarse-grained activities.

6 A Subgraph-Homeomorphism-based Approach to Behavioural Adaptation

In this section, we explain how to reduce the behavioural adaptation problem to vdSH determination. We start from the assumption that when a failure occurs during the execution of a user task T, and service adaptation is not feasible, we determine the subgraph G_f representing the part of T that is not executed. Then, we attempt to find an alternative behavioural graph (in the same task class as T), which contains a subgraph G_a that is functionally equivalent to G_f, but behaving differently.

By different behaviour we mean having abstract activities with different granularities. Two cases should be considered to this regard. First, the user task is adapted by dividing one or more activities in G_f into finer-grained activities, i.e., G_f is a topological minor of G_a. In this case we are interested in determining if there is a vdSH from the G_f into G_a.

Second, the user task is adapted by merging one or more activities in G_f into coarse-grained activities. In this case we are interested in determining whether G_a is a topological minor of G_f, i.e., there is a vdSH from G_a into G_f.

In order to determine the subgraph G_a, we consider the difference between each graph in the task class and the activities of T that are already executed. To do so, we use the notion of difference graph defined below.

Definition 10 ***Difference graph***
Given a graph $G = (V, E, \mu)$ and a subgraph $S = (V_s, E_s, \mu_s)$ of G, the difference of G and S is the subgraph denoted $G - S$ and defined by the set of vertices $V - V_s$.

Based on the definition of difference graph, we formulate both aforementioned cases of behavioural adaptation as follows:

1. Given a subgraph G_e (representing the executed part of the user task), a subgraph G_f (representing the non-executed part of the user task) and a graph G, we aim at determining whether $G_f \lessapprox G - G_e$. More generally, given a family of graphs $\boldsymbol{\mathcal{G}} = \{G_1, .., G_n\}$ (representing the task class to which G_e and G_f belong), we aim at determining whether there is a graph G_i $(1 \leq i \leq n)$ such that $G_f \lessapprox G_i - G_e$. Solving this problem is equivalent to determine if there exists a vertex disjoint subgraph homeomorphism from G_f into each graph $G_i - G_e$.
2. Given a subgraph G_e (representing the executed part of the user task), a subgraph G_f (representing the non-executed part of the user task) and a graph G, we aim at determining whether $G - G_e \lessapprox G_f$. More generally, given a family of graphs $\boldsymbol{\mathcal{G}} = \{G_1, .., G_n\}$ (representing the task class to which G_e and G_f belong), we aim at determining whether there is a graph G_i $(1 \leq i \leq n)$ such that $G_i - G_e \lessapprox G_f$. Solving this problem is equivalent to determine if there exists a vertex disjoint subgraph homeomorphism from each graph $G_i - G_e$ into G_f.

To resolve the aforementioned behavioural adaptation problem, we recall the algorithm for vdSH determination defined in [Xiao et al., 2007]. This algorithm is based on a state-space searching [Nilsson, 1980] and employs heuristics to prune the search space. The basic idea of a space-state search is to divide the mapping process into several states and to define transitions from one state to another. Each state ω can be associated with a partial mapping solution $\mathcal{M}_\omega = (VM, EM)$, where VM and EM are respectively the set of vertex mapping and the set of edge-path mapping at state ω. For each state ω, there is an evaluation function $\varphi(\omega)$ which describes the quality of the represented solution. The states are expanding themselves according to the value of $\varphi(\omega)$. In the mapping process, the next state is more complete than the previous one and will probably be a subset of the final mapping.

Nevertheless, as already mentioned in Section 3, the vdSH determination algorithm is not enough to state that two subgraphs (representing two abstract sub-compositions) are functionally equivalent. To cope with this issue, we present two main solutions. First, before proceeding to vdSH determination, we add a preliminary step of vertex mapping aiming to verify that both subgraphs (subjects of vdSH determination) cover the same set of functionalities. Second, we introduce a set of extensions to the vdSH determination algorithm. These extensions aim at ensuring that functional semantics are preserved during vdSH determination, consequently during

behavioural adaptation.

Next, we detail the preliminary step of vertex mapping (Section 6.1). We use the term *vertex mapping* to denote the correspondence between two sets of vertices belonging to two subgraphs, and we use the term *vertex matching* to denote the semantic matching between two single vertices based on their descriptions.

6.1 Preliminary Verifications

The preliminary verifications aim at reducing the number of subgraphs investigated for vdSH determination. It consists in two main steps. First, vertex mapping which aims at verifying that each couple of investigated subgraphs (denoted G_f and $G_i - G_e$) cover the same functionalities. To do so, we attempt to establish a vertex mapping which ensures that all the vertices (i.e., activities) in the subgraph $G_i - G_e$ are functionally related to one or more vertices (i.e., activities) in G_f. That is, both subgraphs cover similar functionalities, but only the granularity of abstract activities and their order are different.

As mentioned in the user task definition (see Definition 8), vertices are labelled using complex labels formed of the tuple (f, I, O) denoting respectively the function, the set of inputs and set of outputs of an abstract activity. Starting from the assumption that abstract compositions may be labelled using heterogeneous labels (as they may be defined by different users), we opt for a semantic vertex matching in order to cope with the syntactic heterogeneity of labels. More specifically, we suppose that each element in the tuple (f, I, O) is characterized by a semantic concept *cpt* in a domain ontology.

To enable the vertex mapping between subgraphs, we proceed to a semantic vertex matching, which is defined upon a concept matching with concepts associated to the tuple (f, I, O). We begin by defining the concept matching, after that we address the semantic vertex matching.

In our approach, we adopt the definition of concept matching proposed in [Paolucci et al., 2002]. Given two concepts cpt_1 and cpt_2 belonging to the same ontology Ω. Paolucci et al. define four values of concept matching M_Ω:

1. $M_\Omega(cpt_1, cpt_2)$= Exact, if cpt_1 and cpt_2 are equivalent concepts,
2. $M_\Omega(cpt_1, cpt_2)$= PlugIn, if cpt_1 is a super concept of cpt_2,
3. $M_\Omega(cpt_1, cpt_2)$= Subsume, if cpt_1 is a sub-concept of cpt_2,
4. $M_\Omega(cpt_1, cpt_2)$= Fail, if cpt_1 and cpt_2 do not verify the above conditions.

Starting from the above definition of concept matching, we present the following definitions specifying the semantic vertex matching:

Definition 11 ***Semantic label matching***
Given two labels l_1 and l_2, and a domain ontology Ω, l_1 is semantically matched to l_2 if:
- *l_1 and l_2 are respectively associated with two concepts c_1 and c_2 within Ω ; and*
- *$M_\Omega(cpt_1, cpt_2)$= Exact, or PlugIn, or Subsume.*

The labels l_1 and l_2 can be associated with functions, Inputs, or Outputs.

Definition 12 ***Semantic vertex matching***
Given two subgraphs G_1 and G_2, and two vertices $v_1 = (f_1, I_1, O_1)$ and $v_2 = (f_2, I_2, O_2)$ with $v_1 \in G_1$ and $v_2 \in G_2$, v_1 is semantically matched to v_2 if:
- *f_1 is semantically matched to f_2 ; and*
- *there exists $i \in I_1$ and $i' \in I_2$ such that i is semantically matched to i', (denoted $I_1 \cap_s I_2 \neq \emptyset$): and*
- *there exists $o \in O_1$ and $o' \in O_2$ such that o is semantically matched to o', (denoted $O_1 \cap_s O_2 \neq \emptyset$).*

In this definition, we introduce a flexible semantic vertex matching ensuring that the vertex v_2 is semantically matched a vertex v_1 if: (i) it performs either the same functionality as v_1, or a sub-functionality of v_1, or a more complete functionality than v_1, and (ii) it processes at least one input of v_1, and (iii) it produces at least one output of v_1.

Such a flexible definition enables matching a coarse-grained activity to two or more fine-grained activities, respectively, matching a fine-grained activity to a coarse-grained activity (see the example of Figure V.6). Based on the above definition of semantic vertex matching, we can establish a correspondence between the vertices of two subgraphs in order to verify whether they encompass equivalent functionalities.

Given two subgraphs $G_1 = (V_1, E_1, \mu_1)$ and $G_2 = (V_2, E_2, \mu_2)$, we state that both subgraphs G_1 and G_2 cover the same set of functionalities if there is a function $f : V_{A_1} \rightarrow V_{A_2}$ such that:

1. $f(v_1) = v_2$ *iff* v_1 is semantically matched to v_2.
2. For any $v_2 \in V_{A_2}$ there exists a vertex $v_1 \in V_{A_1}$ such that $f(v_1) = v_2$.

Where $V_{A_1} \subseteq V_1$ and $V_{A_2} \subseteq V_2$ denote respectively the set of vertices associated with abstract activities in G_1 and G_2. We recall that the vertex mapping deals only with the vertices associated with abstract activities.

When a vertex mapping is established between G_1 (representing the failed subgraph) and G_2 (representing the investigated subgraph), we proceed to the second preliminary verification

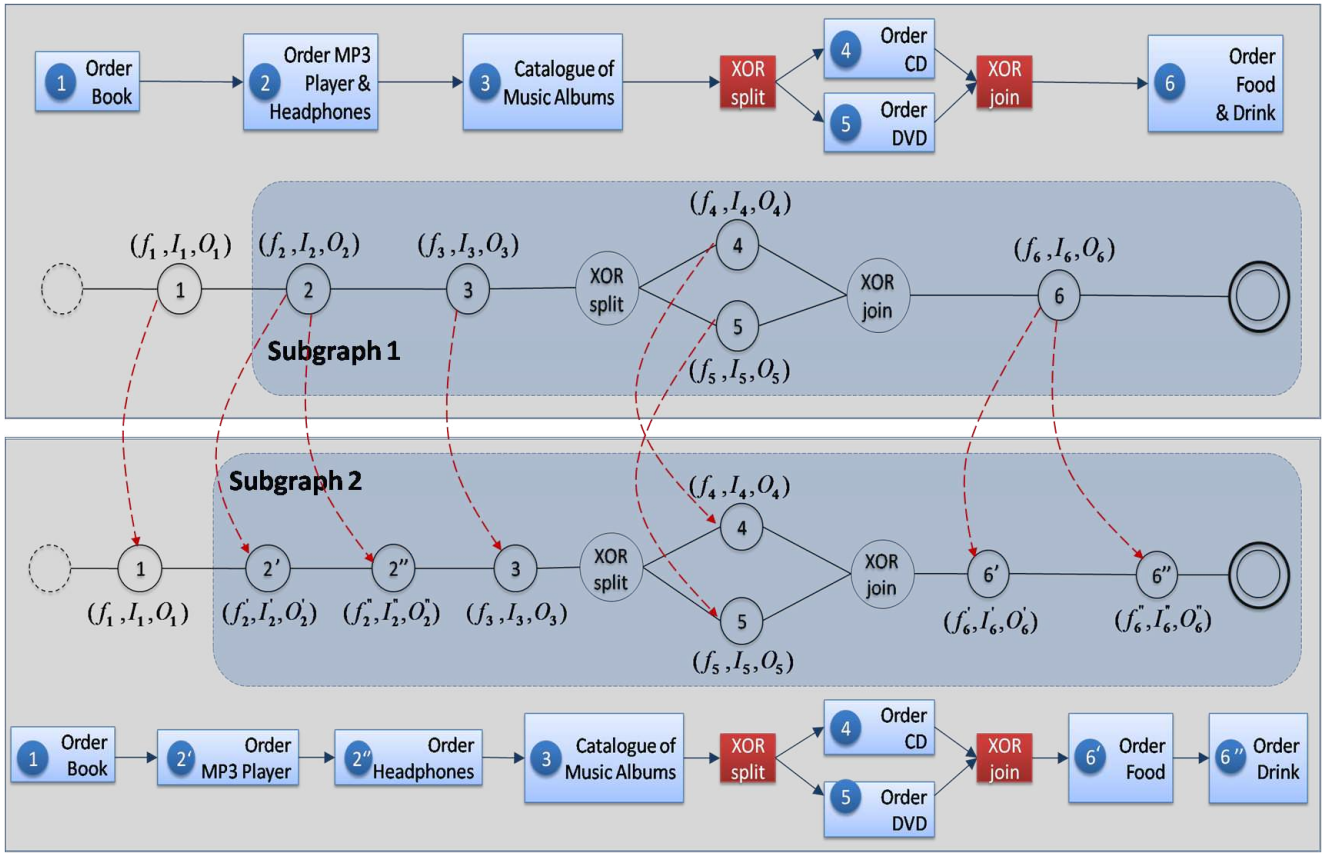

Figure V.6 – Illustrating the preliminary vertex mapping

which consists in checking that: (i) the investigated subgraph does not require more inputs than the failed subgraph, i.e., all the inputs required by G_2 are provided by G_1, and (ii) the investigated subgraph produces all the outputs of the failed subgraph, i.e., all the outputs of G_1 are included in the outputs of G_2. Formally, this can be expressed as follows.

Given two subgraphs G_1 and G_2 with two sets of vertices associated with abstract activities, respectively, V_{A_1} and V_{A_2}. We state that there is a correspondence between the inputs and outputs of G_1 and G_2, if two conditions are satisfied:

1. $\bigcup I_i \subseteq \bigcup I_j$
2. $\bigcup O_j \subseteq \bigcup O_i$

where I_i (respectively I_j) denotes the set of inputs associated the vertex V_i (respectively V_j), O_i (respectively O_j) denotes the set of outputs associated the vertex V_i (respectively V_j), and $V_i \in V_{A_1}$ (respectively $V_j \in V_{A_2}$).

The above verification of inputs and outputs is not carried out at the level of individual vertices, but rather at the level of the whole subgraph (i.e., globally). To illustrate this verifica-

tion, we consider the example depicted in Figure V.6. In this example, we have two subgraphs with two sets of vertices associated with abstract activities, respectively, $\{V_1, V_2, V_3, V_4, V_5, V_6\}$ and $\{V_1, V_2', V_2'', V_3, V_4, V_5, V_6', V_6''\}$. Before investigating both subgraphs for vdSH determination, we must verify that : (i) $\bigcup_{I_i} \subseteq \bigcup_{I_j}$, and (ii) $\bigcup_{O_j} \subseteq \bigcup_{O_i}$, where $i \in \{1, 2, 3, 4, 5, 6\}$ and $j \in \{1, 2', 2'', 3, 4, 5, 6', 6''\}$.

Both preliminary verifications (i.e., vertex mapping and input/output verification) are recursively executed in order to check the functional, input and output correspondence between the failed subgraph and each alternative subgraph (obtained from the task class). These verifications represent a preliminary graph filtering allowing to discard subgraphs requiring more inputs, producing less outputs, or having extra or less functionalities (with respect to the failed subgraph), hence reducing the number of subgraphs to be investigated for vdSH determination.

6.2 Extended Vertex Disjoint Subgraph Homeomorphism

In this section, we introduce some extensions to vdSH determination defined in [Xiao et al., 2007] in order to preserve the functional semantics of the user task during behavioural adaptation and also reduce the number of graphs to be investigated during vdSH determination.

6.2.1 Semantic vertex matching

vdSH determination is based on an injective vertex mapping α which assumes a simple and syntactic vertex labelling function. That is, a vertex has a simple label l and it can be mapped only to vertices having the same label. However, as explained above, in our approach we consider a semantic vertex matching, thus enabling to cope with the syntactic heterogeneity of labels. For this reason, the injective vertex mapping α performed during vdSH determination is extended to support semantic matching of vertices, as detailed in Section 6.1.

6.2.2 Data constraints

vdSH algorithms do not consider any constraints on edges connecting vertices. However in our context, we have to consider data constraints relating abstract activities, which imposes certain constraints on vertices' ordering. For instance, in our pervasive shopping scenario, the activity *catalogue of music albums* has data links with the activities *Order CD* and *Order DVD*. Indeed, the title and the reference of the required CD or DVD must be selected from the catalogue. Thus, the activities *Order CD* and *Order DVD* must be always preceded by the activity *catalogue of music albums*. Such data constraints must be preserved when adapting the

behaviour of the user task. To do so, we extend the injective edge mapping β performed during vdSH determination so that it takes into consideration data constraints. Towards this purpose, we first propose a definition of a *data link*, then we define our extension to the injective edge mapping.

Definition 13 ***Data link***
Given two vertices $v_1 = (f_1, I_1, O_1)$ and $v_2 = (f_2, I_2, O_2)$, there is a data link from v_1 to v_2 if there exists $o \in O_1$ and $i \in I_2$ such that i is semantically matched to o, (denoted $O_1 \cap_s I_2 \neq \emptyset$).

Definition 14 ***Extended injective edge mapping***
Given two subgraphs $G_1 = (V_1, E_1, \mu_1)$ and $G_2 = (V_2, E_2, \mu_2)$, and the vdSH problem $G_1 \lessapprox G_2$, an injective edge mapping from G_1 into G_2 is a function $\beta : E(G_1) \rightarrow P(G_2)$ such that:

- *for each $e(v_1, v_2) \in E(G_1)$, $\beta(e)$ is a simple path p in G_2 with $\alpha(v_1)$ and $\alpha(v_2)$ as two ends, and*
- *all mapped paths are pairwise independent, and*
- *if there is a data link from v_1 to v_2, there must be a data link from v_i to v_k for each $e(v_i, v_k) \in p$.*

$P(G_2)$ denotes the set of simple paths in G_2, and α is the extended injective vertex mapping (see Section 6.2.1).

6.2.3 Particular vertex mappings

The last restrictions that we impose on vdSh determination concern the mapping of the start and final vertices, vertices associated with composition patterns V_p, and vertices associated with loop activities. Given two subgraphs $G_1 = (V_1, E_1, \mu_1)$ and $G_2 = (V_2, E_2, \mu_2)$, and the vdSH problem $G_1 \lessapprox G_2$:

- The start vertex s_1 of G_1 must be mapped only to the start vertex s_2 of G_2.
- A final vertex $f \in F_1$ must be mapped only to a final vertex $f' \in F_2$.
- A vertex $v \in V_{p_1}$ with a label l must be mapped only to a vertex $v' \in V_{p_2}$ having the same label l.
- A loop vertex $v \in V_{A_1}$ must be mapped only to a loop vertex $v' \in V_{A_2}$ (obviously v must be semantically matched to v').

F_1 and F_2 denote respectively the final vertices of G_1 and G_2, V_{p_1} and V_{p_2} are the vertices associated with composition patterns in G_1 and G_2, respectively. Whereas, V_{A_1} and V_{A_2} denote

the vertices associated with abstract activities in G_1 and G_2, respectively.

Intuitively, the aforementioned extensions may improve the timeliness of the vdSH determination algorithm as they impose several restrictions that may reduce the number of investigated subgraphs. However, further studies should be established to confirm this statement. Besides, there is a second question that should be resolved towards a comprehensive behavioural adaptation solution. This question concerns the case where vdSH determination produces more than one solution, i.e., there exists several alternative subgraphs (i.e., abstract sub-compositions) that can be applied for behavioural adaptation. To cope with this issue, a certain metric should be introduced to measure the degree to which an alternative subgraph represents a satisfactory behavioural adaptation solution. This metric should reflect parameters such as vertex subdivisions, as well as the consumed inputs and the produced outputs of the alternative subgraph ; this is subject of ongoing research.

7 Evaluation and Discussion

The need to take into account adaptation of service compositions is underlined by tremendous research efforts in the service oriented community. A current trend is to study the equivalence between services in order to enable the replacement of defective services [Ibrahim et al., 2011].

In the literature, we distinguish two broad approaches addressing services' equivalence. The first approach deals with services as black boxes and determines their equivalence based on their functional and QoS descriptions. The second approach considers services as white boxes and focuses on their behavioural equivalence.

Concerning the first approach, we distinguish two types of service equivalence, namely syntactic equivalence and semantic equivalence [Kareliotis et al., 2009]. Syntactic equivalence is carried out by matching services' descriptions that are defined using a common syntax.

In semantic equivalence, services are described using different syntactic terms that reference a common domain ontology. Services' equivalence is then carried out by matching semantic concepts within services' descriptions. In [Paolucci et al., 2002], the authors define four values of semantic concept matching: Exact, PlugIn, Subsume and Fail (see Section 6.1 for details).

However, in [Zeng et al., 2008] the authors define three values of semantic concept matching: (i) Exact match, which means that two concepts c_1 and c_2 directly match with no traversal of the ontological concept hierarchy, (ii) Direct match, when c_1 is a direct subclass of c_2, and (iii)

Taxonomic match, when c_1 is a descendant concept in the taxonomic hierarchy, i.e., there are arbitrary number of levels of inheritance.

Based on the semantic matching defined in [Paolucci et al., 2002], Ibrahim et al. [Ibrahim et al., 2011] introduce a semantic service substitution approach for pervasive environments. The authors present a semantic service model that comprehends the operation (i.e., function), inputs, outputs and QoS properties of services. Based on this model, the authors specify the semantic equivalence between services as well as the degree of match between their QoS properties.

Other approaches (e.g., [Kareliotis et al., 2009],[Lim and Thiran, 2010]) establish communities of services (also known as similarity group of services [Ruta et al., 2008]), which encompass functionally equivalent services. Within these communities, services are sorted with respect to their QoS. For instance, in [Kareliotis et al., 2009] the process of sorting services based on their QoS is driven by the user-defined QoS policy, specifying the importance of QoS properties, as well as the lower and upper bounds for each QoS property.

Nevertheless, equivalence established based on the functional and QoS properties of services is not always enough to enable accurate replacement of services. This is particularly true for services having complex behaviours. For this reason, other adaptation approaches establish service equivalence based on behavioural specifications.

PERSE [Ben Mokhtar, 2007] presents a preliminary idea towards this purpose. The author generates from the user task specification a global graph that contains all behavioural (adaptation) possibilities of the user task. However, PERSE does not focus on adaptation as a separate problem from service composition.

Chafle et al. [Chafle et al., 2006] propose a behavioural adaptation approach, which comprehends two phases: (i) Logical composition, which uses planning techniques to produce several behaviours allowing to fulfill the user task ; (ii) Physical composition, which generates a concrete service composition based on a specific behaviour defined at the logical composition stage and selected according to a ranking function. At run-time, if the concrete service composition fails, another behaviour is selected and implemented using concrete services. The main drawback of this approach is that it does not consider the case where the composition is partially executed, hence where adaptation should ideally be applied only for the failed part of the composition.

To enable partial adaptation of the user task, SIROCCO [Fredj, 2009] considers stateful services and uses the annotated finite state automata (aFSA) formalism to model their behaviour. SIROCCO further proposes a service substitution framework that takes into account the state of a service at the moment of the failure, thus ensuring a transparent adaptation and continuous execution of the service.

A recent discovery approach which is very close to our behavioural adaptation strategy, is introduced by Grigori et al. [Grigori et al., 2010]. The authors define service compositions as BPEL processes, and they aim at determining the functional equivalence between them. To do so, they transform these processes into behavioural graphs, then they apply subgraph isomorphism to determine whether they are functionally equivalent. To enable inexact matching of graphs (e.g., vertex skipping), the authors further define edit operations on graphs such as deleting and inserting edges and vertices. Despite the importance of the approach, it is not as flexible as our matching enabled by vdSH determination.

Behavioural equivalence of services is also known in the literature as *replaceability analysis* [Ponge et al., 2010]. Briefly stated, replaceability refers to the ability for a service to replace another one without inducing incompatibilities [Dumas et al., 2008]. Further definitions of replaceability are provided in [Elabd et al., 2009].

In the ServiceMosaic project [Benatallah and Motahari-Nezhad, 2006], services' behaviour is represented using a specific class of timed automata, called protocol timed automata (PTA). Based on PTA, two main classes of replaceability are defined: subsumption and equivalence. A PTA p_1 subsumes a PTA p_2 if p_1 supports at least all the execution traces that p_2 supports. A service s_1 (with PTA p_1) can replace a service s_2 (with PTA p_2). If p_1 subsumes p_2 and p_2 subsumes p_1, then p_1 and p_2 are equivalent, and services s_1 and s_2 can be used interchangeably.

In the same project, Ponge et al. [Ponge et al., 2010] provide an approach for replaceability analysis that determines either full or partial replaceability. The latter kind of replaceability means that a PTA p_1 does not support all the execution traces of p_2, but there is at least one possible execution of p_2 that is supported by p_1.

Other approaches (e.g., [Beyer et al., 2005] and [Elabd et al., 2009]) represent services' behaviour as timed protocols and provide several algorithms for replaceability analysis. Globally, the main drawback of addressing behavioural equivalence using timed automata is that such automata are not always available (i.e., developers seldom provide such specifications).

In our approach, we generate the behaviour specification automatically from the user task definition (e.g., abstract BPEL). We define the behavioural adaptation problem using graph minor theory, and we solve it using an existing algorithm for vertex disjoint subgraph homeomorphism. We propose some extensions to the vdSH determination algorithm in order to cope with the requirements of behavioural adaptation. Compared to related work, our approach focuses on adapting only a part (i.e., the failed part) of a running composition and not the whole composition, thus avoiding the problem of rolling back the executed services.

As yet, in our approach we show how to reduce behavioural adaptation to vdSH, but we do not claim that applying behavioural adaptation is always possible or efficient. Further studies

should be established to this regard. These studies should consider many factors including: (i) the delay of the behavioural adaptation strategy, (ii) the execution time of the user task, and (iii) the number of alternative abstract compositions in the task class. Indeed, if we deal with long running user tasks, the effectiveness of applying behavioural adaptation increases. However, the effectiveness and efficiency of this adaptation also depend on the number of existing alternative abstract compositions. If this number is high, vdSH determination may produce more solutions, but it may take longer time.

Chapter VI

QASOM: A QoS-Aware Service-Oriented Middleware for Pervasive Environments

In this chapter, we present QASOM, a QoS-aware Service-Oriented Middleware which provides a comprehensive solution for service composition and adaptation in pervasive computing environments. QASOM implements the contributions presented in this thesis, notably QoS-aware service selection (Chapter IV) and QoS-driven composition adaptation (Chapter V). The remainder of this chapter is structured as follows. In Section 1, we present an overview of the SemEUsE research project to which this thesis has contributed. In Section 2, we introduce the QASOM middleware. Finally, we present a prototype implementation and performance evaluation of QASOM in Section 3.

1 SemEUsE Research Project

This thesis has contributed to the SemEUsE research project [1] funded by the French National Research Agency (ANR). This project has aimed at leveraging SOA middleware technologies, viz., Enterprise Service Bus (ESB), by taking into account semantic services and QoS requirements, while composing services and running resulting complex services. SemEUsE enables services to be used anywhere from any kind of device (pervasive services), while addressing QoS requirements associated with service provisioning. Next, we give an overview of the SemEUsE architecture, then we highlight our contribution to this project.

1. SemEUsE project: http://www.semeuse.org/

1.1 SemEUsE Architecture

Figure VI.1 depicts the overall architecture of SemEUsE. It adheres to the Service Oriented Computing paradigm and is designed according to modularity and flexibility requirements in order to provide distinct and easily manageable system components. The features of the SemEUsE architecture are thus designed as a set of components deployed on top of PEtALS, an existing open-source ESB (Enterprise Service Bus) for large SOA architectures.

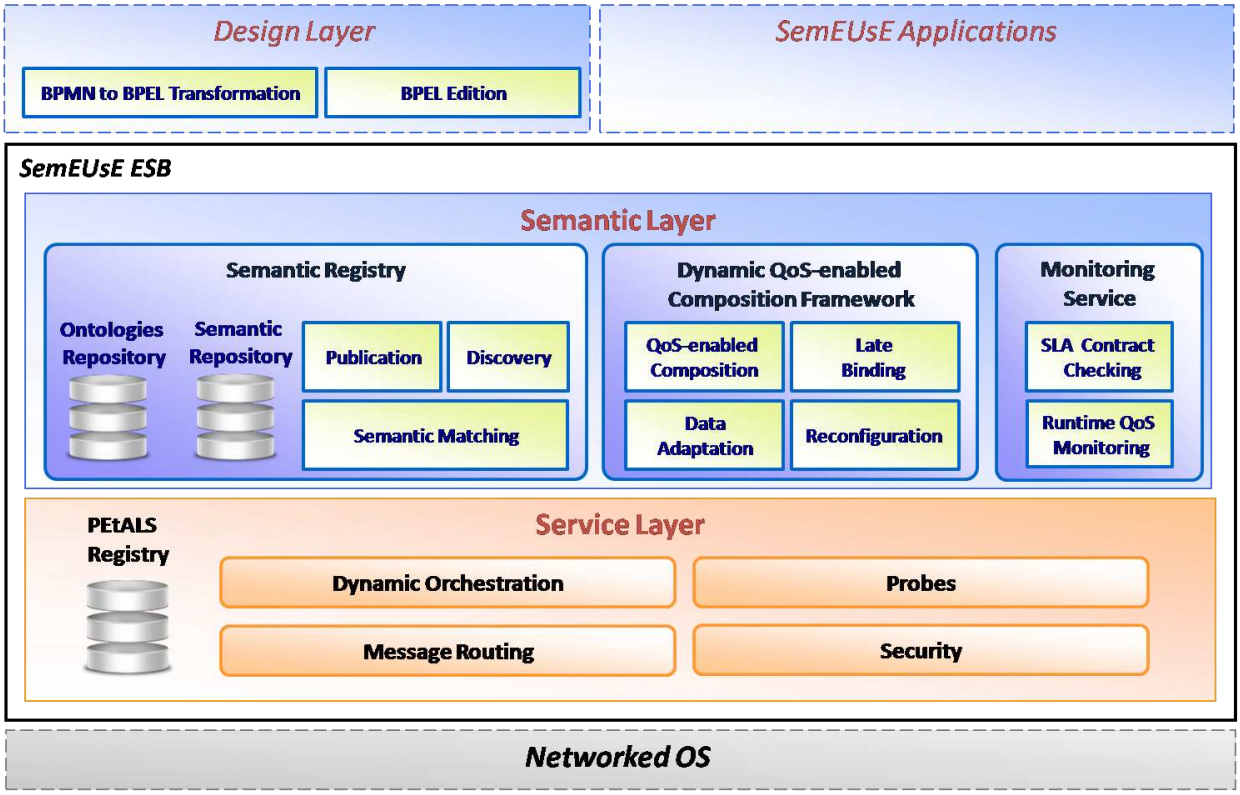

Figure VI.1 – SemEUsE ARchitecture

Built in accordance with the layering approach, the SemEUsE architecture services comprises two main parts: (i) a service design part, and (ii) a run-time part. The former part enables the specification of user requests, which are formed of abstract user tasks with associated global QoS requirements. This part makes the SemEUsE architecture user-friendly and accessible to non-experts thanks to a graphical editor and a translation tool from high level languages such as BPMN into BPEL. Whereas the SemEUsE run-time part comprehends two layers:

1. The Service Layer enabled by the PEtALS ESB corresponds to the backbone of SemEUsE and provides low level capabilities to deploy, invoke services, and monitor them with probes. This layer comprehends five components: (1) PEtALS Registry, (2) Dynamic

Orchestration, (3) Probes, (4) Message Routing, and (5) Security.

2. The Semantic Layer corresponds to the main contribution of the SemEUsE project. It is composed of a set of middleware services offering semantic-enabled capabilities: (i) the use of semantic service and task descriptions that combine functional, non-functional (e.g. QoS, security and context) and user profiles aspects ; and (ii) the ability to dynamically discover, select, compose, monitor and reconfigure services at run-time, while enforcing QoS. In a coarse-grained view, the semantic layer comprehends three main components: (1) Semantic Registry, (2) Dynamic QoS-enabled Composition Framework, and (3) Monitoring Service. Further details about the SemEUsE architecture are provided in the SemEUsE website [2].

1.2 Contribution to SemEUsE

Our contribution to SemEUsE concerns the *Dynamic QoS-enabled Composition Framework*. More specifically, we have contributed to two components within this framework: *QoS-enabled Composition* and *Reconfiguration* (see Figure VI.1).

1.2.1 QoS-enabled Composition

QoS-enabled Composition plays a primary role in the SemEUsE architecture. It is the component responsible for building an executable composition able to fulfill the user task and meet its associated global QoS requirements. QoS-enabled Composition implements QASCO, our QoS-aware service composition (including our selection algorithm QASSA).

1.2.2 Reconfiguration

Dynamic reconfiguration is an important feature in SemEUsE. It allows for coping with QoS fluctuations of services during the execution of the composition. The Reconfiguration component addresses this purpose by reconfiguring dynamically the composition at run-time without aborting its execution.

Our contributions to SemEUsE, notably QoS-enabled Composition and Reconfiguration represent the core of our QoS-aware Service-Oriented Middleware for pervasive environments (QASOM).

2. http ://www.semeuse.org/architecture.html

2 QASOM Middleware

We present in this section, the architecture and the implementation techniques of QASOM. As already introduced, QASOM has been initially developed as a part of the SemEUsE middleware, more specifically, it implements the QoS-awareness aspect in SemEUsE.

2.1 QASOM Architecture

As depicted in Figure VI.2, QASOM is composed of two frameworks: *QoS-aware Service Composition Framework*, and *QoS-driven Composition Adaptation Framework*. The former framework implements the contributions presented in Chapter IV, whereas the latter implements the contributions of Chapter V of this thesis. Concerning service discovery and monitoring, QASOM is designed in a flexible way so that it can be used with several proper discovery and monitoring approaches. In the context of SemEUsE, QASOM relies on those services offered by the SemEUsE middleware (since service discovery and monitoring are not in the scope of this thesis).

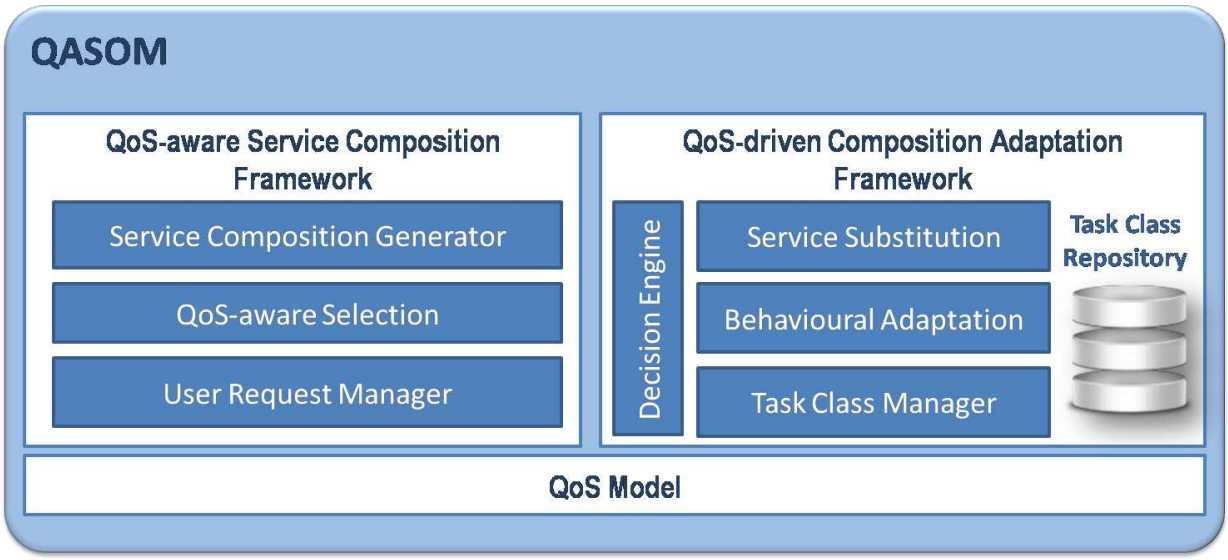

Figure VI.2 – QASOM Architecture

2.1.1 QoS-aware Service Composition Framework

As depicted by Figure VI.3, the QoS-aware Service Composition Framework comprehends the following components:

- *User Request Manager* is the component that receives and processes the user request. It comprehends two sub-components: (i) User Request Parser, which is responsible for parsing the user request and providing information such as the activities forming the user task and global QoS constraints, and (ii) User Request Splitter, which splits the user request into elementary requests that are later used for services' discovery or for the distributed execution of QASSA. The User Request Splitter requires information provided by the User Request Parser.
- *QoS-aware Service Selection* implements the QASSA algorithm. It comprehends two sub-components: (i) QoS Aggregator, which is responsible for determining the overall QoS of a service composition based on QoS aggregation formulae explained in Chapter IV, and (ii) Composition Selector, which is responsible for selecting and ranking near-optimal service compositions.
- *Service Composition Generator* is responsible for generating an executable service composition based on (i) the specification of the user task, and (ii) the services selected by QASSA. Compared to the abstract user task, the executed composition mainly interpolates information needed to bind and enact a service for each abstract activity int the user task.

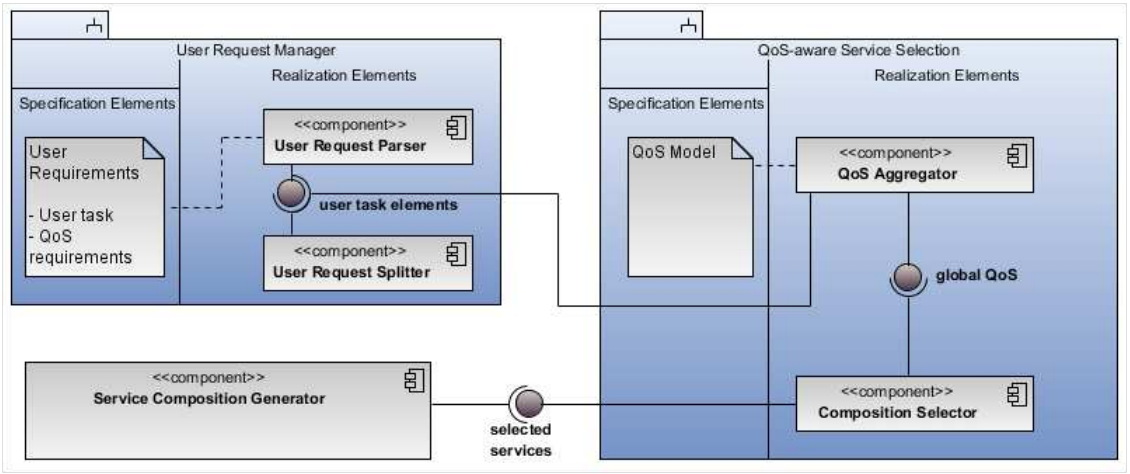

Figure VI.3 – QoS-aware Service Composition Framework

The QoS-aware Service Composition Framework takes as input a user request which comprehends the description of the user's task and the set of global QoS constraints imposed on this task. To fulfill the required task, it proceeds through the following steps:

1. User Request Parser parses the user task's specification and determines its underlying abstract activities and the set of global QoS constraints imposed on the whole task.

2. User Request Manager invokes the SemEUsE discovery service to acquire a set of service candidates for each activity in the user's task.
3. QoS-aware Service Selection chooses a set of services among the candidates of each activity, which meet global QoS constraints and maximize the overall QoS offered to the user.
4. Based on the selected services, Service Composition Generator builds an executable composition able to fulfill the user's task.

2.1.2 QoS-driven Composition Adaptation Framework

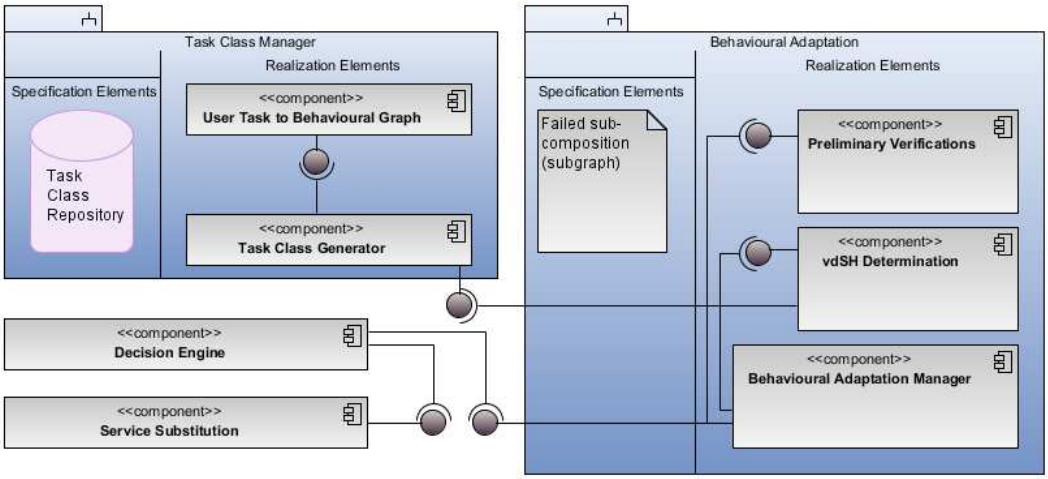

Figure VI.4 – QoS-driven Composition Adaptation Framework

As depicted in Figure VI.4, QoS-driven Composition Adaptation Framework comprehends four components:

- *Decision Engine* is responsible for evaluating the status of a running service composition with respect to QoS requirements of the user, and decides about the appropriate adaptation strategy accordingly.
- *Service substitution* is responsible for substituting the failed services of a running composition according to the service substitution strategy.
- *Task Class Manager* takes in charge the management of task classes in pervasive environments. It comprehends two elementary components: (i) *User Task To Behavioural Graph*, which transforms a given abstract service composition to a behavioural graph, and (ii) *Task Class Generator*, which generates and stores task classes used in a specific pervasive environment.

- *Behavioural Adaptation* implements the behavioural adaptation strategy. Given a failed user task, it calls Task Class Manager to acquire its associated task class, then it proceeds to vdSH determination using the following components:
  - *Preliminary Verifications* is responsible for performing the preliminary verifications (i.e., semantic vertex mapping and input/output verification) before carrying out the vdSH determination.
  - *vdSH Determination* implements the extended vdSH determination algorithm defined in Chapter V.
  - *Behavioural Adaptation Manager* coordinates the sub-components of Behavioural Adaptation (i.e., Preliminary Verifications and vdSH Determination) in order to fulfill behavioural adaptation requests triggered by Decision Engine.

2.2 Prototype Implementation

We have implemented a prototype of QASOM using Java technologies, notably Java 1.6. to implement our service selection and adaptation algorithms, J2EE to support Web Services and SOA technologies, and Android SDK 2.2 to implement the distributed version of QASSA. To specify service compositions we use the BPEL [3] language. The usage of BPEL is motivated by the fact that it defines service compositions with different abstraction levels. Jaeger [Jaeger, 2006] divide composition languages into three classes: (i) Abstract level languages, (ii) Concrete level languages, and (iii) languages covering both levels. BPEL belongs to the third class and represents according to the authors a typical service composition language. Hence, we use BPEL to specify both user tasks and concrete service compositions. The importance of this approach is twofold. First, the user task is specified using a standard language of service compositions. Second, the user task can be easily transformed into a concrete and executable service composition by replacing opaque elements with concrete execution details, as discussed in the next section.

2.3 Specifying User Tasks

BPEL provides a sub-specification named *Abstract Process*, which defines abstract Web Service compositions by omitting their execution details and keeping only an abstract description of the function, inputs and outputs of Web Services. Abstract Process uses the notion of opacity to omit execution details of Web Service compositions. According to this, BPEL elements

3. BPEL 2.0: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

related to aspects such as Web Service's binding and data processing are replaced with opaque tokens (i.e., ##opaque).

Nevertheless, Abstract Process is a general specification, which covers several use cases and different abstraction degrees. To delimit the usage of Abstract Process, the notion of *profile* is used. BPEL specifies several profiles including the *Template* profile, which provides a high-level representation of Web Service compositions and hides almost any execution detail [4]. Below, we give a code snippet of the abstract BPEL associated with the pervasive shopping scenario introduced in Chapter I.

```
<process name="PervasiveShoppingProcess" xmlns="http ://docs.oasis-open.org/wsbpel/2.0/process/abstract"
targetNamespace="http ://www-rocq.inria.fr/arles/bpel/examples"
xmlns :tns="http :// http ://www-rocq.inria.fr/arles/bpel/examples "
suppressJoinFailure="yes"
xmlns :xsd="http ://www.w3.org/2001/XMLSchema"
xmlns :ext="http ://example.com/bpel/some/extension"
xmlns :template="http ://docs.oasis-open.org/wsbpel/2.0/process/abstract/simple-template/2006/08"
abstractProcessProfile="http ://docs.oasis-open.org/wsbpel/2.0/process/abstract/simple-template/2006/08">
<extensions>
<extension namespace="http ://docs.oasis-open.org/wsbpel/2.0/process/abstract/simple-template/2006/08"
mustUnderstand="yes"/>
</extensions>
<partnerLinks>
<partnerLink name="BookShop" partnerLinkType="##opaque" partnerRole="BookVendor">
...
</partnerLink>
</partnerLinks>
<variables>
<variable name="BookRequestVar" element="##opaque" />
<variable name="BookReplyVar" element="##opaque" />
...
</variables>
<assign>
<documentation> Transform book purchase information into the format accepted by the book shop service.
</documentation>
<from opaque="yes" />
<to variable="BookRequestVar"/>
</assign>
<sequence>
<scope>
<faultHandlers>
<catchAll> ... </catchAll>
</faultHandlers>
<sequence>
<opaqueActivity>
<documentation> opaque receive activity </documentation>
</opaqueActivity>
```

4. Abstract Process Profile for Templates: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

```
<invoke partnerLink="BookShop" operation="BookPurchaseOrder" inputVariable="BookRequestVar" />
<receive partnerLink="BookShop" operation="BookPurchaseOrder" variable="BookReplyVar"/>
</sequence>
</scope>
...
</sequence>
</process>
```

2.4 Specifying Executable Service Compositions

Based on the user task specification and the services selected by QASSA, we generate an executable service composition. Compared to the user task specification, the generated composition mainly interpolates concrete details about Web Service's binding and data exchanged between Web Services. BPEL allows binding exactly one concrete Web Service for each abstract activity in the user task. Nevertheless, as introduced in Chapter IV, in our approach we need to bind multiple Web Services for each activity in the user task in order to support dynamic binding of services.

To cope with this issue, we use the *Extension Activity* of BPEL, which allows for defining customized activities fitting specific use cases of Web Service compositions. In our context, we use Extension Activity to define a special dynamic binding activity, which supports binding multiple concrete services to each abstract activity in the user task. Below, we give a code snippet of the extension activity associated with the book ordering abstract activity (of the pervasive shopping scenario).

```
<extensionActivity>
<qasom :DynamicBindingInvoke invocationID="BookPurchaseOrder" inputVariable="BookPurchaseRequest"
outputVariable="BookPurchaseReply"/>
<qasom :candidateServices>
<qasom :service EPR="BookShop1" portType="BookShop1PT" operation="purchaseBook"/>
<qasom :service EPR="BookShop2" portType="BookShop2PT" operation="orderBook"/>
...
</qasom :candidateServices>
</extensionActivity>
```

3 Experimental Results

We conducted a set of experiments to assess the centralized and distributed versions of QASSA, and our QoS-driven composition adaptation approach, notably the transformation of the user task into a behavioural graph. Concerning the evaluation of the extended vdSH determination algorithm, it makes part of our future work. As yet, the reader can refer to the experimental evaluation of vdSH determination presented in [Xiao et al., 2007].

3.1 Experimental set up

Table VI.1 describes the experimental set up used in our experiments. For the evaluation of QASSA, we are interested in two metrics:

1. **Execution time** measures the timeliness of QASSA with respect to the size of the selection problem in terms of the number of activities and the number of services per activity.
2. **Optimality** measures how optimal is the QoS utility provided by QASSA. This is determined by the ratio of the QoS utility resulting from QASSA over the optimal QoS utility given by a brute-force algorithm (that we developed for the purpose of the experiments). The optimality metric is then given by the following formula:

$$Optimality = \mathcal{F}/\mathcal{F}_{opt}$$

where F is the QoS utility given by our heuristic algorithm (see Chapter IV, Section 3.3), and F_{opt} is the optimal QoS utility given by the brute force algorithm.

To determine F_{opt}, we developed a brute force algorithm that we execute using a cluster of computers called *riob*[5]. To the best of our knowledge, there are no other works that use the brute-force algorithm to determine the optimality of QoS-aware service selection heuristics. Instead, most of the existing works (e.g., [Alrifai et al., 2010]) use MILP solvers such as the IBM ILOG CPLEX Optimizer[6] to determine optimality. Nevertheless, MILP solvers can not replace the brute-force algorithm, since they also use heuristics to determine the optimal service composition, thus the optimality results given by these works are not accurate.

Centralized algorithm	**Distributed algorithm**
- Machine: Dell	- Machine: HTC Desire
- Processor: AMD Athlon 1.80GHz	- Processor: Qualcomm QSBD8250 1GHz
- RAM: 1.8 GB	- RAM: 576 Mo
- OS: Windows XP	- OS: Android 2.2 (Froyo)
- Programming language: J2SE 1.6	- Programming language: Android SDK 2.2 (based on J2SE 1.5)

Table VI.1 – Experimental set up

For the purpose of our experiments, we developed a *Composition Generator*, which randomly generates service compositions used for experimenting with QASSA. The Composition Generator takes as parameters the number of activities a and the number of candidate services

5. https://www.rocq.inria.fr/intranet/miriad/themes/cluster/riob.htm
6. http ://www-01.ibm.com/software/integration/optimization/cplex-optimizer/about/ ?S_CMP=rnav

per activity k, and it proceeds through two steps: (i) yielding an abstract service composition which comprehends a activities structured with respect to randomly chosen composition patterns, (ii) binding k concrete services to each activity in the composition. QoS values associated with these services are acquired from the QWS dataset available online [7]. This dataset consists of 5000 real Web services, each with a set of 9 QoS properties measured using commercial benchmark tools [Al-Masri and Mahmoud, 2007a,b].

Once service compositions are generated, we further need to configure the execution of QASSA with respect to the following parameters:

1. *Aggregation approach*: As already introduced in Chapter IV, our algorithm supports three QoS aggregation approaches: pessimistic, optimistic and mean-value. We opt for the pessimistic approach as the default method for aggregating QoS values. We further perform experimentations with respect to the three aggregation approaches in order to study their impact on the timeliness and optimality of QASSA.
2. *User QoS constraints*: QASSA requires as input global user QoS constraints imposed on the whole composition. As we do not have real user requirements, we opt for a statistical method to determine global QoS constraints. For each QoS property (e.g., response time) we calculate the mean value m_i of the service candidates associated with each activty A_i, then we aggregate all mean values (i.e., m_1, m_2,.., m_n) with respect to the structure of the composition. That is, we set the global QoS constraint of each QoS property to the aggregated mean value of service candidates associated with each abstract activity. Additionally, we set global QoS constraints to different statistical values in order to analyze their impact on the timeliness and optimality of QASSA. Further details are given in Section 3.2.2.

3.2 Performance of QASSA (the centralized version)

In this section, we present the experimental evaluation of the centralized version of QASSA. We begin by evaluating the timeliness of the algorithm. Towards this purpose, we vary the number of activities between 10 and 50, the number of services per activity between 50 and 200, and the number of QoS constraints between 2 and 5. For the sake of precision, we execute each experiment 20 times and we calculate the mean value of the obtained results.

Figure VI.5 (a) depicts the execution time of QASSA with respect to the number of services per activity. We fix the number of QoS constraints to 5, vary the number of activities between

7. http://www.uoguelph.ca/∽ qmahmoud/qws/index.html

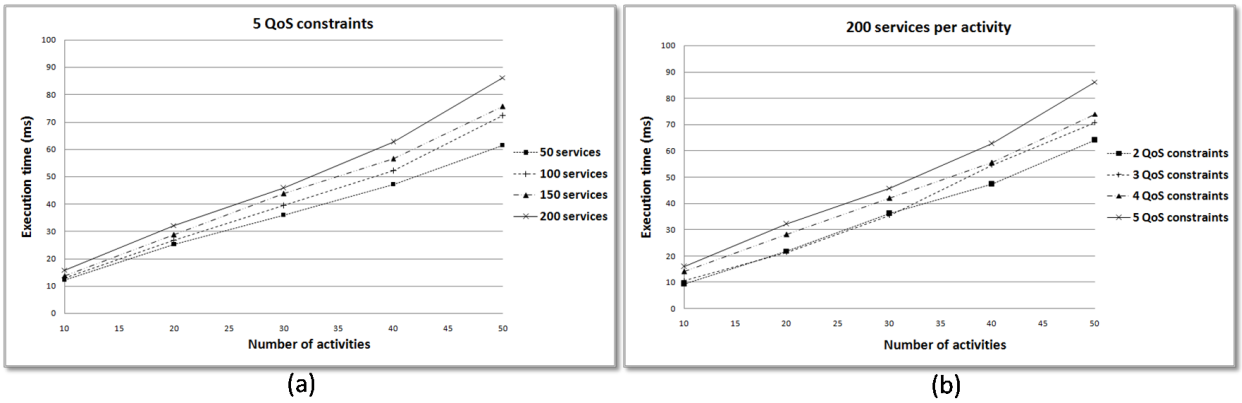

Figure VI.5 – Execution time while varying (a) the number of services per activity and (b) the number of QoS constraints

10 and 50, and vary the number of services per activity between 50 and 200. The obtained results show that the execution time of our algorithm increases (up to 89ms) along with the number of services, which is an expected result.

Figure VI.5 (b) depicts the execution time of QASSA with respect to the number of QoS constraints. We fix the number of services per activity to 200 and vary QoS constraints between 2 and 5. The obtained results show that the execution time of our algorithm increases (up to 89ms) along with the number of QoS constraints, which is also an expected result, i.e., a higher number of QoS constraints requires more computational effort, hence a longer execution time.

Both figures show that the execution time of our algorithm increases almost linearly along with the number of activities in the composition. In general, our algorithm executes in a timely manner (i.e., less than 0.09s) with respect to spontaneous interaction with users aimed at by pervasive computing.

Concerning the optimality of QASSA, we measure it while varying the number of activities between 5 and 10. We choose such a low number of activities because the execution time of the brute-force algorithm for a high number of activities is too long (because of the computational complexity of the problem).

Figure VI.6 (a) depicts the optimality of QASSA while fixing the number of QoS constraints to 5, varying the number of activities between 5 and 10 and varying the number of services per activity between 50 and 200. It shows that the optimality of QASSA is generally more than 90%, and it can reach 100%. However, for a low number of activities and a low number of services per activity, the optimality decreases to 60%, which can be explained by the fact that when the number of services decreases, the probability to find services with a satisfactory value for all QoS properties decreases also, hence yielding a low optimality.

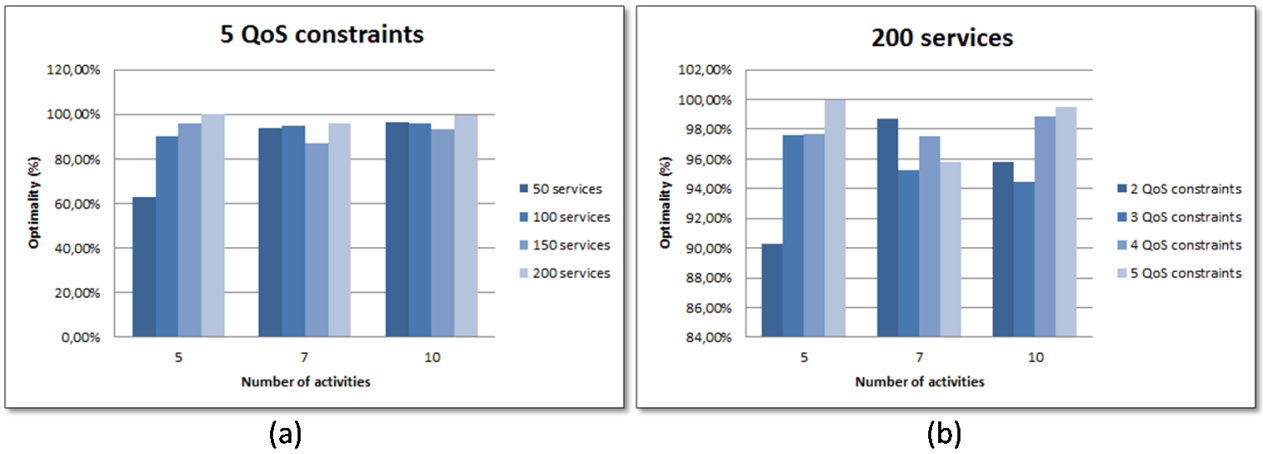

Figure VI.6 – Optimality measurements while (a) varying the number of services and (b) the number of QoS constraints

Additionally, we measure the optimality of QASSA while fixing the number of services to 200, varying the number of activities between 5 and 10 and varying the number of QoS constraints between 2 and 5. Figure VI.6 (b) shows that the optimality of our algorithm is generally satisfactory (more than 90%) and it can reach 100%.

Overall, both figures show that the optimality of our algorithm varies between 90% and 100% independently from the number of services and the number of QoS constraints.

3.2.1 Impact of the Aggregation Approach

As already introduced, QASSA supports different QoS aggregation approaches (i.e., pessimistic, optimistic and mean-value). These approaches produce different QoS values of service compositions, which may impact the results obtained by our algorithm. For this reason, we propose to evaluate QASSA with respect to various QoS aggregation approaches, notably pessimistic, optimistic, and mean-value approach.

Towards this purpose, we fix the number of QoS constraints to 5, we vary the number of activities in the composition between 10 and 50, and we vary the number of services per activity between 50 and 200. Figure VI.7 depicts the execution time of our algorithm associated with the pessimistic, mean-value, and optimistic aggregation approaches, respectively. The reader can clearly notice that aggregation approaches have no effect on the execution time of our algorithm, which can be explained by the fact that these approaches require nearly the same computational effort (i.e., since they perform similar aggregation operations).

Concerning the optimality of QASSA, we measure it while fixing the number of QoS constraints to 5, varying the number of activities in the composition between 5 and 10 and

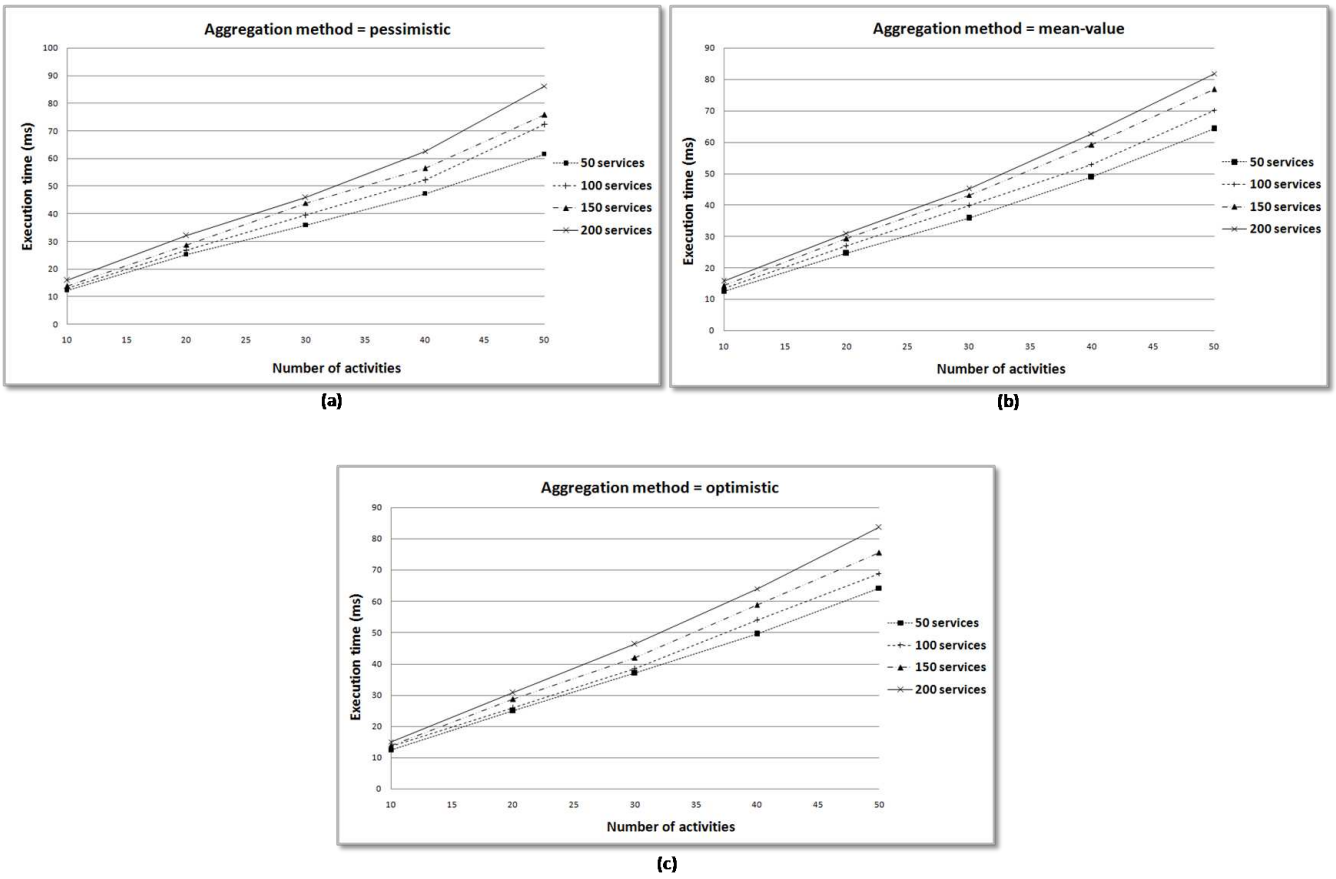

Figure VI.7 – Execution time wrt to the (a) pessimistic, (b) optimistic and (c) mean-value aggregation methods

varying the number of services per activity from 50 to 200. Figure VI.8 depicts the optimality of our algorithm associated with the (a) pessimistic, (b) mean-value, and (c) optimistic aggregation approaches. This figure shows that the optimality of the algorithm slightly decreases from (a) to (c). The best optimality is associated with the pessimistic aggregation approach, it reaches 100% ; for the mean-value aggregation approach the optimality does not exceed 99% ; whereas for the mean-value aggregation approach it is limited to 97%. This can be explained by the fact that when our algorithm is too stringent and considers the worst QoS values, it discards more service compositions and keeps only those compositions with high QoS.

3.2.2 Impact of User QoS requirements

The degree to which users are demanding, i.e., how strict are their QoS requirements, obviously impacts the number of service compositions able to fulfill these requirements, which impacts the results given by QASSA. For this reason, we propose to evaluate our algorithm

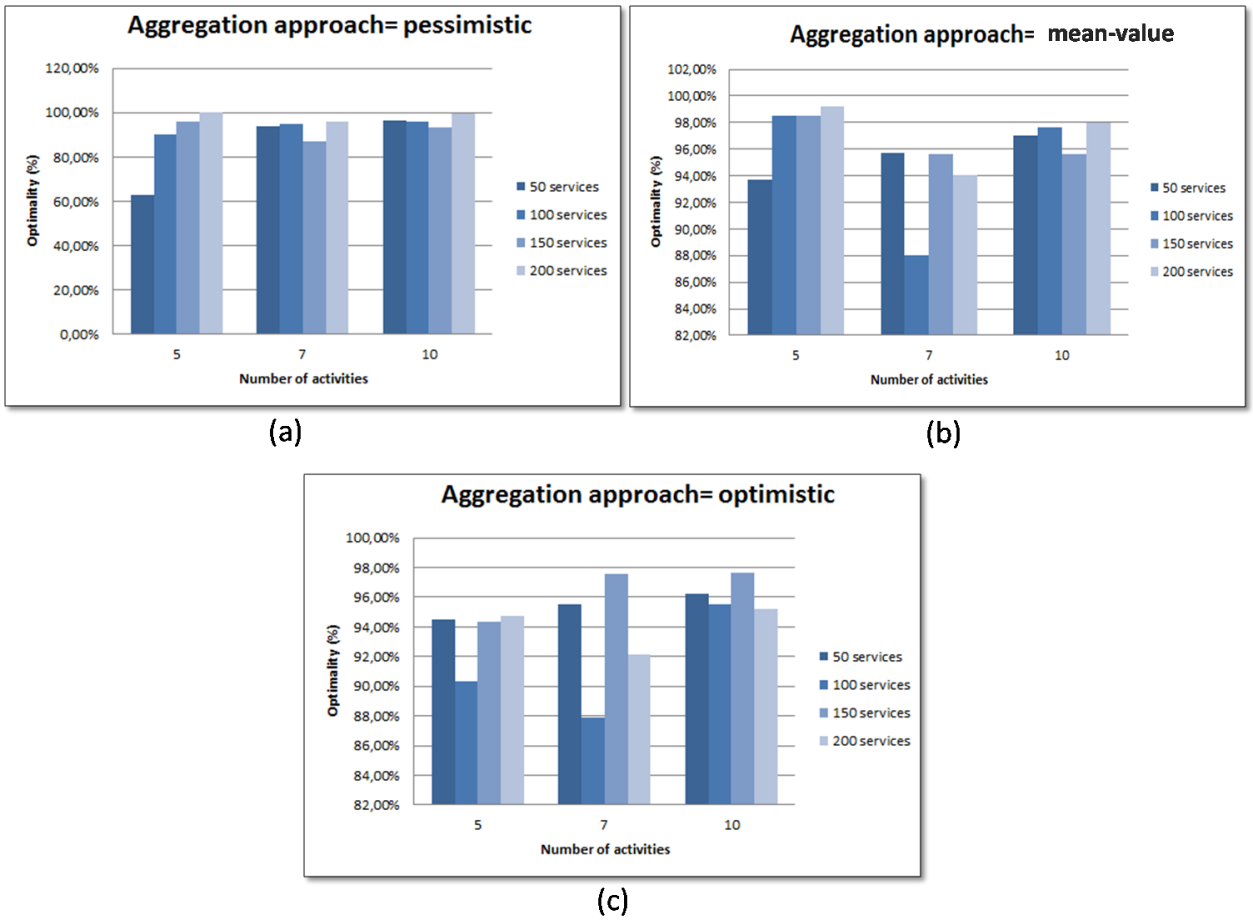

Figure VI.8 – Optimality of the algorithm wrt to the (a) pessimistic, (b) optimistic and (c) mean-value aggregation methods

with respect to various values of the user QoS requirements.

In practice, determining such requirements for evaluation of the algorithm is not evident and requires real-world scenarios (i.e., real requirements), as well as it depends on the user profile (e.g., whether users are demanding or not). Existing QoS-aware service selection algorithms do not give systematic method for setting meaningful user QoS requirements.

To cope with this issue, we opt for a statistical approach which allows for determining global QoS requirements based on QoS values q_i of service candidates. Related to this, we test our algorithm while setting the global QoS requirements QC_i (associated with each QoS property P_i) to two values:

For positive QoS properties: $$QC_i = \begin{cases} Agg(m) \\ Agg(m + \sigma) \end{cases}$$

For negative QoS properties: $QC_i = \begin{cases} Agg(m) \\ Agg(m-\sigma) \end{cases}$

Where Agg is a function aggregating QoS values with respect to the services and structure of the composition (see Table IV.1), and m and σ are respectively the mean value and standard deviation of QoS values q_i of service candidates associated with each abstract activity.

Under the assumption that we deal with a large number of service candidates, the central limit theorem [Hogben et al., 2007] states that the randomly generated QoS values follow the normal distribution law. Thus, setting the values m and $m+\sigma$ (respectively, m and $m-\sigma$ for negative QoS properties) as local QoS constraints (i.e., for each abstract activity), allows for discarding 50% and 84,1% of service candidates, respectively.

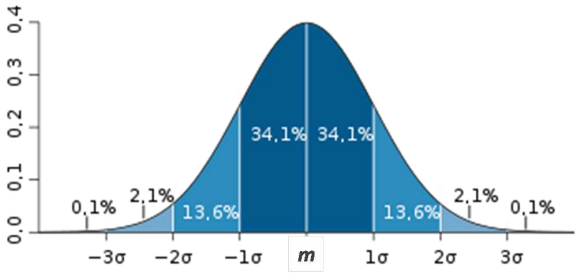

Figure VI.9 – The normal distribution law

Consequently, setting the user QoS requirements to $Agg(m)$ and $Agg(m+\sigma)$ (respectively, $Agg(m)$ and $Agg(m-\sigma)$ for negative QoS properties) may discard approximately 50% and 84,1% of service candidates, respectively.

Figure VI.10 depicts the execution time of our algorithm associated with user QoS requirements set to m and $m+\sigma$, respectively. In these figures, we notice that the execution time slightly decreases when the value of user constraints increases, which can be explained by the fact that the algorithm discards more service compositions when user QoS requirements are more constraining, which reduces the number of investigated compositions, hence reducing the execution time of the algorithm.

Concerning the optimality results, we measure it while fixing the number of QoS constraints to 5, varying the number of activities in the composition between 5 and 10 and varying the number of services per activity from 50 to 200.

Figure VI.11 depicts the optimality of our algorithm associated with user QoS requirements respectively set to m and $m+\sigma$. This figure shows that the optimality produced by our al-

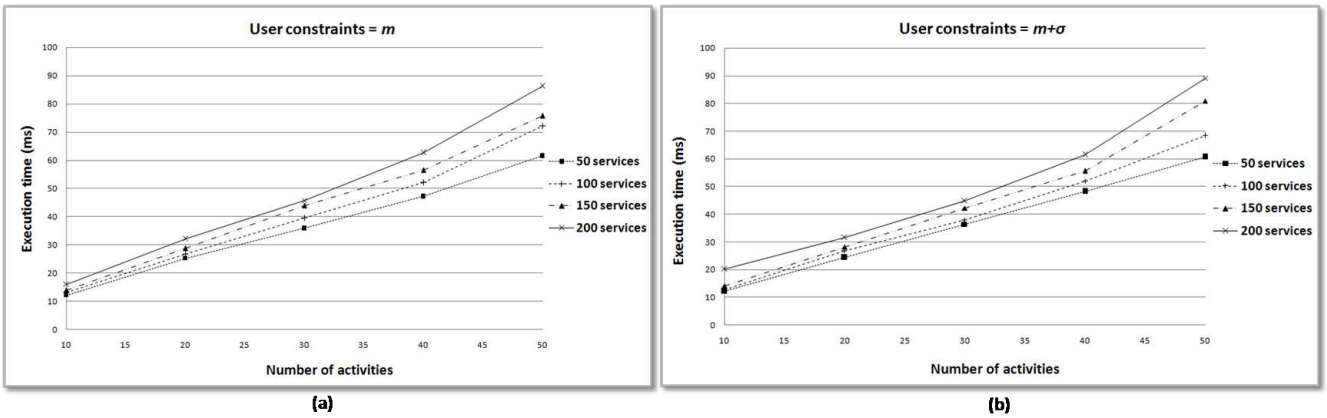

Figure VI.10 – Execution time while fixing global QoS requirements to (a) m and (b) $m + \sigma$

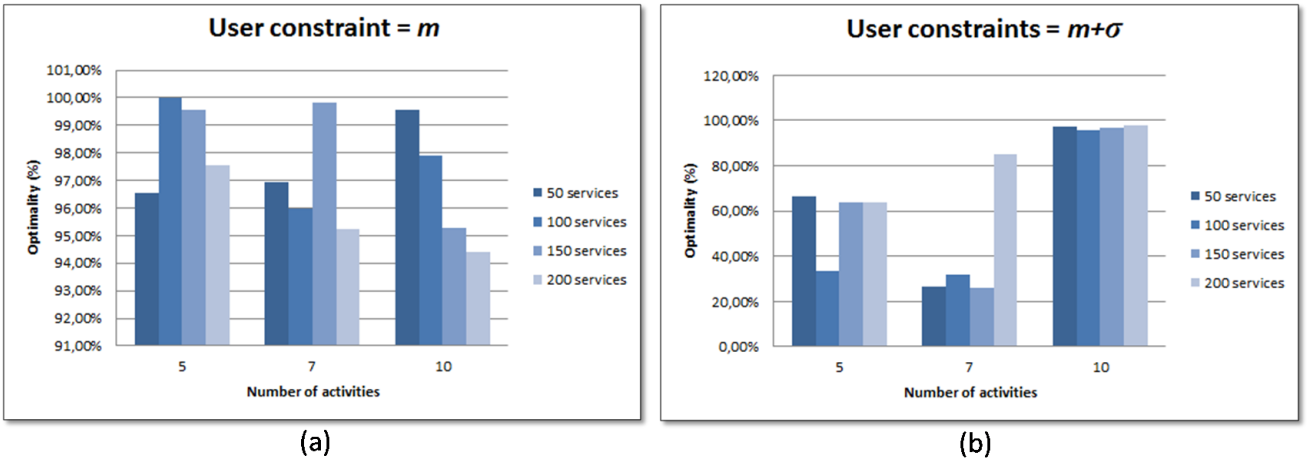

Figure VI.11 – Optimality of the algorithm while fixing global QoS requirements to (a) m, (b) $m + \sigma$

gorithm considerably decreases (it declines to 26% in some cases) when the value of user QoS requirements increases, which is an expected result. Indeed, QASSA discards more service compositions when the value of user QoS requirements increases, hence the probability to find a service composition with high optimality declines.

3.3 Performance of QASSA (the distributed version)

In this section, we evaluate the distributed version of QASSA using the experimental setup detailed in Table VI.1. Distributing QASSA changes two main features compared with the centralized version, notably: (i) the communication cost of the algorithm (specifically, the communication cost between the devices participating in fulfilling the user task), and (ii) the hardware setup underpinning the execution of the algorithm. Both features do not impact the

optimality of QASSA, thus in our experiments, we focus only on the execution time metric. Additionally, we assume that the communication cost is neglectable compared with the overall execution time of the algorithm. Thus, the execution time presented in these experiments concerns only the local and global selection algorithms of the distributed version of QASSA.

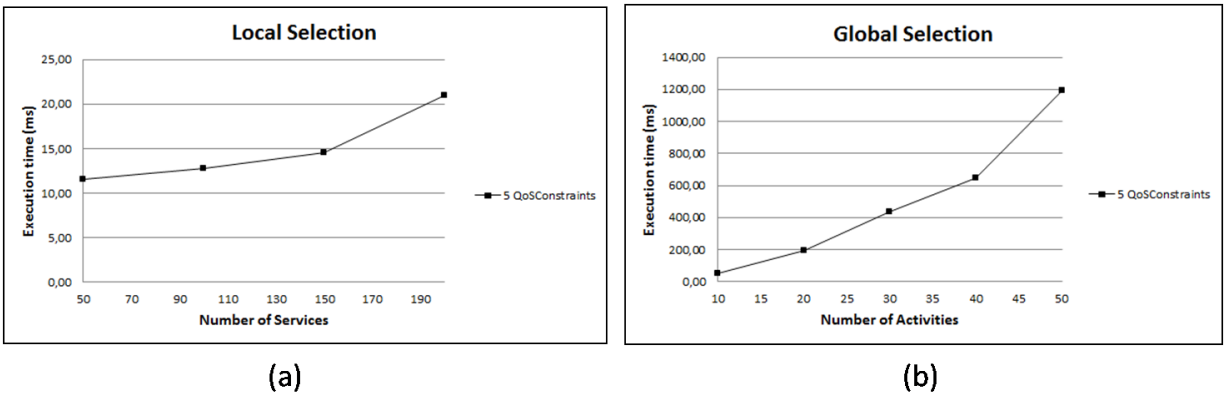

Figure VI.12 – Execution time of the (a) local selection and (b) global selection of our distributed QoS-aware service selection algorithm

Figure VI.12 depicts the execution time of the local selection and global selection of the distributed version of QASSA. For the local selection, the execution time is measured for one abstract activity (indeed, each helper device processes local selection for one activity in the user task). We fix the number of QoS constraints to 5 and vary the number of services between 50 and 200. Whereas for the global selection, we fix the number of QoS constraints to 5, the number of services to 200 and we vary the number activities in the user task between 10 and 50.

Despite the limited hardware resources used in these experiments, QASSA shows satisfactory timeliness with respect to spontaneous interaction with users aimed at by pervasive environments. Indeed, the local selection is executed in at most 25 ms, whereas the global selection is executed in at most 1.2 s, thus the overall algorithm can be accomplished in less than 1.5s, depending on the size of the user task and the number of services per activity.

3.4 Performance of Transforming the User Task into a Behavioural Graph

In this section, we partially evaluate the performance of our QoS-driven composition adaptation approach. We particularly focus on transforming the user task into a behavioural graph. Concerning the evaluation of the extended vdSH determination algorithm, it makes part of our future work.

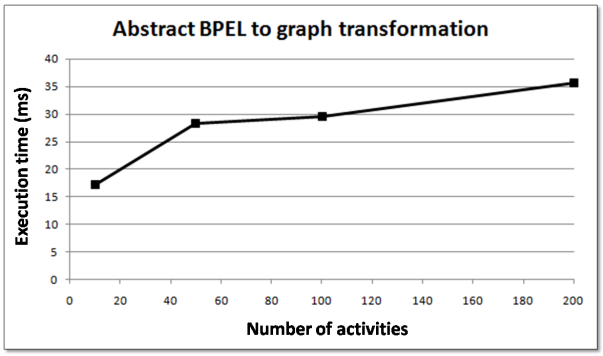

Figure VI.13 – Execution time of transforming abstract BPEL specifications into behavioural graphs

As we use the BPEL language to represent user tasks, we aim at evaluating the transformation of an abstract BPEL specification into a behavioural graph. Towards this purpose, we developed an *abstract composition generator*, which automatically generates abstract BPEL specifications given the number of opaque invoke activities as input. Opaque invoke activities are structured according to randomly chosen BPEL composition patterns.

Once the abstract BPEL is generated, we apply the BPEL to graph transformation defined by Grigori et al. [Grigori et al., 2010] in order to generate the equivalent behavioural graph. The idea behind the transformation is to map BPEL activities to respective behavioural graph elements. To do so, we traverse the nested structure of the BPEL document in a top-down manner and apply recursively a transformation procedure specific to each type of activities. We use the StAX [8] Java API for XML to parse the BPEL document, and the JGraphT [9] API to build and handle graphs.

We measure the execution time of the transformation starting from parsing the abstract BPEL specification until the complete building of the graph. We perform 20 executions for each transformation and we give the average execution time. Figure VI.13 depicts the obtained results. It shows that the transformation is executed in a small amount of time, i.e., nearly 35ms.

8. StAX API: http://stax-ex.java.net/
9. JGraphT API: http://jgrapht.sourceforge.net/

Chapter VII

Conclusion

Pervasive computing environments are characterized by their limited computational resources and wireless connectivity, which may cause Quality of Service (QoS) decline or fluctuation. Therefore, delivering a satisfactory QoS to users represents a major challenge for the pervasive computing community. Indeed, without QoS guarantees pervasive computing looses much of its interest.

To ensure a satisfactory level of QoS, user tasks need to be accomplished while being 'QoS aware', i.e., aware about QoS conditions in pervasive environments during their execution. The notion of QoS awareness involves (i) establishing a proper QoS model that enables the effective understanding and use of QoS, (ii) considering QoS aspects of services during the composition and deployment of the user task, and (iii) adapting the execution of the user task with respect to QoS changes in pervasive environments.

In this thesis, we opt for a middleware solution to address the aforementioned QoS awareness issues. Our middleware contributes to the state of the art through the provision of efficient QoS awareness solutions for pervasive computing environments. Below, we present the contributions of our middleware, then we give the short term and long term perspectives of our research work.

1 Contributions

In this thesis, we presented a QoS-aware service-oriented middleware for pervasive computing environments, and a prototype implementation of this middleware, viz., QASOM.

The first contribution of our middleware is a semantic QoS model that supports interoperability between heterogeneous QoS descriptions used by users and service providers in pervasive environments. Our model is based on a QoS standardization effort established by OASIS [1], and offers a set of QoS ontologies that include an extensive categorization of QoS properties.

1. OASIS Web Services Quality Model TC: http://www.oasis-open.org/committees/tc_home.php ?wg_abbrev=wsqm

The proposed ontologies cover QoS on an-end-to-end basis by considering QoS properties of all the factors involved in fulfilling the user task, notably applications services, their underlying infrastructure, and end-users, hence providing a comprehensive of vision of QoS. Our model is further extendible so that domain specific QoS properties can be added easily. With all the aforementioned features, our QoS model provides the appropriate ground for QoS awareness in pervasive environments and supports our middleware in fulfilling the user task while delivering satisfactory QoS to users.

As part of the QASOM prototype, we developed the QoS ontologies of our model using the OWL-DL [2] language. To produce semantically rich QoS specifications, the developed ontologies are combined with an emergent QoS specification language proposed by the OASIS Web Services Quality Model Technical Committee, namely WSQDL (Web Service Quality Description Language). More specifically, we have enriched WSQDL specifications with semantic annotations referencing QoS concepts in our ontologies. Specifying services' QoS is subsequently achieved by associating semantically enriched WSQDL specifications with WSDL descriptions (WSDL is an XML-based schema describing Web Services' interfaces).

To assist users of pervasive environments in realizing their daily tasks, our middleware provides a QoS-aware service composition approach that assembles, dynamically on-the-fly, proper services available in the environment in order to accomplish users' tasks. Our composition approach introduces QASSA, a novel and efficient QoS-aware service selection algorithm that copes with major challenges of service selection in pervasive computing environments. Thanks to a novel way of using clustering techniques, QASSA is highly selective and can execute in a timely manner with respect to spontaneous interaction with users. Additionally, QASSA can be executed in a centralized or distributed fashion depending on the infrastructure of pervasive environments (i.e., stationary and centralized infrastructure or *ad hoc* infrastructure). Furthermore, QASSA considers end-to-end QoS properties and supports QoS-driven composition adaptation by (i) selecting multiple alternative service compositions, and (ii) supporting dynamic binding of services, i.e., binding services just before their invocation based on the run-time QoS of services.

QASSA is implemented as a part of the QASOM prototype. The experimental evaluation of QASSA shows that both versions of the algorithm (i.e., centralized and distributed versions) execute in a timely manner with respect to spontaneous interaction with users aimed at by pervasive computing. At the same time, QASSA meets the user QoS requirements while achieving a high QoS optimality.

2. OWL-DL: http://www.w3.org/TR/owl-features/

To ensure delivering the intended QoS steadily during the execution of service compositions, our QoS-aware service-oriented middleware introduces a novel QoS-driven composition adaptation approach that focuses on adapting the behaviour of service compositions. Behavioural adaptation represents an important alternative to service substitution widely investigated by existing adaptation approaches.

Our QoS-driven composition adaptation represents user tasks as graphs and reduces the behavioural adaptation problem to a well-known graph problem, viz., vertex disjoint Subgraph Homeomorphism (vdSH) [Xiao et al., 2007]. The advantage of expressing the behavioural adaptation problem as vdSH is two-fold. First, it enables a flexible matching of behavioural subgraphs where vertex subdivision is allowed. This means that behaviours covering the same functionalities but having different granularities can be matched, thus enabling to take advantage of many potential ways for accomplishing the user task.

Second, expressing the behavioural adaptation problem as vdSH allows for applying behavioural adaptation for subgraphs instead of complete graphs (which is the case of existing behavioural adaptation approaches). This means that only the part of the user task which is not accomplished is concerned by behavioural adaptation, thus avoiding to roll-back the whole composition and switching from a behaviour to another in a transparent and flexible manner.

Finally, it is worth mentioning that the main objective of our QoS-driven composition adaptation approach has been to bring the behavioural adaptation problem to a well-known graph field where multiple solutions and algorithms are already proposed, hence opening new perspectives to efficiently resolve this problem. The experimental validation of our QoS-driven composition adaptation approach makes part of our future work.

Overall, the QASOM prototype implementation of our middleware constitutes an efficient and comprehensive QoS-awareness solution for pervasive computing environments, and has been successfully integrated in the ANR SemEUsE project.

2 Perspectives

Besides the contributions presented above, short term and long term perspectives are still to be investigated towards a valuable solution for QoS awareness in pervasive environments. Short term perspectives represent potential enhancements of our middleware, whereas long term perspectives are about future research directions that we deem important to enable the QoS vision aimed at by pervasive computing.

2.1 Short-term perspectives

In order to improve the actual state of our middleware, we mainly focus on enhancing our QoS-driven composition adaptation approach. Three short-term perspectives can be investigated towards this purpose. First, we aim at theoretically validating our QoS-driven composition approach. Particularly, we are interested in establishing proofs about the correctness of our approach in the sense that the functional equivalence established between subgraphs based on vdSH determination is correct. The second short term perspective is about fully validating our QoS-driven composition adaptation approach experimentally. Specifically, we are interested in studying the response time of our extended vdSH algorithm with respect to timeliness requirements in pervasive environments. Finally, we are interested in extending our approach in order to handle the case where multiple behavioural adaptation solutions are found. That is, we shall define a utility function based on which we can determine the best behavioural adaptation solution.

2.2 Long-term perspectives

Integrating QoS Awareness and Context Awareness The notion of context represents any useful information (besides functional and QoS information) characterizing an entity and the world in which this entity operates [Dey et al., 2001]. Related to this, context awareness is defined as "the use of context to provide task-relevant information and/or services to a user, wherever they may be"[Abowd et al., 1999].

The extension of our middleware with the support of context awareness would enable gathering and processing information about user characteristics and state of the physical environment, which can be used to improve the QoS delivered to users [Wac, 2005]. Nevertheless, addressing QoS awareness and context awareness simultaneously brings about a number of challenging issues to investigate regarding middleware for pervasive environments.

The first issue concerns the establishment of an integrated QoS and context model that relates QoS properties to context elements and defines the impact of context variations on the QoS delivered to users. As modelling only one aspect (i.e., QoS or context) is not a trivial task, establishing an integrated QoS and context model seems to be even more challenging.

Another issue concerns managing the trade-off between QoS and context during service discovery, composition and adaptation. Indeed, in many situations the user required QoS and the user context may be conflicting, e.g., choosing a service with a medium QoS and which is next to the user's location or selecting another service a little bit further from the user but offering a higher QoS. This raises the issue of conflict-free discovery, composition and adaptation

approaches. A preliminary idea towards resolving this issue is to find the appropriate balance between QoS and context using, e.g., MCDM (Multi-Criteria Decision Making), or to prioritize the conflicting QoS and context constraints as suggested in [Lupu and Sloman, 1997].

From QoS Awareness to QoS Enhancement The notion of QoS awareness as addressed in this thesis is roughly about managing QoS provided by different resources in pervasive environments so as to deliver the best 'offered QoS' to users. Nevertheless, it may happen that the offered QoS is not enough to gain the user satisfaction because of the resource limitations in pervasive environments.

For this reason, we argue that research efforts should switch from the notion of QoS awareness to a *QoS enhancement* perspective, which envisions improving the QoS delivered to users instead of only managing existing QoS. One idea towards this purpose is to endow user applications with software elements that may enhance specific QoS properties. For instance load balancing, fault tolerance, security software elements may enhance, respectively, the response time, availability and reliability, and the security of user applications.

Related to this, pervasive computing requires a specific software engineering fashion that is able to cope with the inherent challenges of pervasive computing (notably QoS) at the architectural level. This involves defining an architectural style that allows for designing and building user applications while considering software elements able to improve the overall QoS delivered to users. Existing approaches addressing this purpose (e.g., WSAMI [Issarny et al., 2005]) use connectors, considered as key architectural elements, to weave QoS-supporting mechanisms into service-oriented user applications, where hosting and integrating such connectors is accounted for in the design and implementation of the middleware layer.

However, in order to fit on-the-fly design and construction of user applications in pervasive environments, middleware connectors shall be dynamically synthesized in an automated and implicit way, given the specification of the required QoS properties and available QoS-supporting mechanisms.

Middleware connectors dealing with QoS enhancement shall further be composed when dealing with several QoS properties. Same as for the integration of application services, middleware connectors must be also integrated in a way that guarantees the correct and efficient use of connectors, i.e., connecting applications services correctly while improving the overall QoS of the application.

The above raises a number of multi-disciplinary challenges going from formal foundation to run-time engineering of middleware connectors [Issarny et al., 2011].

References

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. 1999. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. HUC '99. Springer-Verlag, London, UK, 304–307. 130

Al-Masri, E. and Mahmoud, Q. 2007a. QoS-based Discovery and Ranking of Web Services. *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, 529–534. 117

Al-Masri, E. and Mahmoud, Q. H. 2007b. Discovering the Best Web Service. In *WWW '07 : Proceedings of the 16th international conference on World Wide Web*. ACM, New York, NY, USA, 1257–1258. 117

Alrifai, M., Risse, T., Dolog, P., and Nejdl, W. 2008. A Scalable Approach for QoS-based Web Service Selection. In *1st International Workshop on Quality-of-Service Concerns in Service Oriented Architectures (QoSCSOA'08) in conjunction with ICSOC 2008*. Sydney. 61, 78, 79

Alrifai, M., Skoutas, D., and Risse, T. 2010. Selecting Skyline Services for QoS-based Web Service Composition. In *Proceedings of the 19th international conference on World wide web*. WWW '10. ACM, New York, NY, USA, 11–20. 78, 116

Amundsen, S. L. and Eliassen, F. 2006. Combined Resource and Context Model for QoS-Aware Mobile Middleware. In *ARCS*. 84–98. 9, 21, 33

Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., and Plebani, P. 2007. PAWS : A Framework for Executing Adaptive Web-Service Processes. *IEEE Softw. 24*, 39–46. 18, 23, 24, 26, 27, 33

Arthur, D. and Vassilvitskii, S. 2007. K-means++ : the Advantages of Careful Seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. 1027––1035. 65

References

Bacon, J. 2002. Toward Pervasive Computing. *IEEE Pervasive Computing 1,* 2, 84. 1

Balasubramanian, K., Wang, N., Gill, C., and Schmidt, D. C. 2003. Towards Composable Distributed Real-time and Embedded Software. In *in WORDS 2003 : 8th International Workshop on Object-oriented Real-Time Dependable Systems*. IEEE, 15–17. 9

Bellavista, P., Corradi, A., and Foschini, L. 2007. Context-aware Handoff Middleware for Transparent Service Continuity in Wireless Networks. *Pervasive Mob. Comput. 3,* 4, 439–466. 6

Ben Mabrouk, N., Beauche, S., Kuznetsova, E., Georgantas, N., and Issarny, V. 2009. QoS-aware Service Composition in Dynamic Service Oriented Environments. In *Middleware '09 : Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag New York, Inc., New York, NY, USA, 1–20. 8, 64, 65, 78

Ben Mabrouk, N., Georgantas, N., and Issarny, V. 2009. A Semantic End-to-End QoS Model for Dynamic Service Oriented Environments. In *PESOS '09 : Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*. IEEE Computer Society, Washington, DC, USA, 34–41. 41

Ben Mokhtar, S. 2007. Semantic Middleware for Service-Oriented Pervasive Computing. Ph.D. thesis, University of Paris 6. 11, 18, 19, 21, 23, 26, 31, 32, 103

Ben Mokhtar, S., Georgantas, N., and Issarny, V. 2007. COCOA : COnversation-based service COmposition in pervAsive computing environments with QoS support. *J. Syst. Softw. 80,* 12, 1941–1955. 21, 27

Ben Mokhtar, S., Kaul, A., Georgantas, N., and Issarny, V. 2006. Efficient Semantic Service Discovery in Pervasive Computing Environments. In *Middleware '06 : Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*. Springer-Verlag New York, Inc., New York, NY, USA, 240–259. 58

Ben Mokhtar, S., Liu, J., Georgantas, N., and Issarny, V. 2005. QoS-aware Dynamic Service Composition in Ambient Intelligence Environments. In *ASE '05 : Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, New York, NY, USA, 317–320. 18, 19, 21, 23, 26, 27

Ben Mokhtar, S., Preuveneers, D., Georgantas, N., Issarny, V., and Berbers, Y. 2008. EASY : Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. *J. Syst. Softw. 81,* 5, 785–808. 58

Benatallah, B. and Motahari-Nezhad, H. R. 2006. Servicemosaic project : modeling, analysis and management of web services interactions. In *Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling - Volume 53*. APCCM '06. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 7–9. 36, 104

Benbernou, S., Brandic, I., Cappiello, C., Carro, M., Comuzzi, M., Kertész, A., Kritikos, K., Parkin, M., Pernici, B., and Plebani, P. 2010. Modeling and Negotiating Service Quality. In *Service Research Challenges and Solutions for the Future Internet*, M. Papazoglou, K. Pohl, M. Parkin, and A. Metzger, Eds. Lecture Notes in Computer Science, vol. 6500. Springer Berlin / Heidelberg, 157–208. 19, 34, 41, 42

Beyer, D., Chakrabarti, A., and Henzinger, T. A. 2005. Web Service Interfaces. In *Proceedings of the 14th international conference on World Wide Web*. WWW '05. ACM, New York, NY, USA, 148–159. 104

Bezdek, J. and Pal, N. 1998. Some New Indexes of Cluster Validity. *Systems, Man, and Cybernetics, Part B : Cybernetics, IEEE Transactions on 28,* 3 (jun), 301 –315. 65

Börzsönyi, S., Kossmann, D., and Stocker, K. 2001. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 421–430. 79

Brans, J. P. and Vincke, P. 1985. A Preference Ranking Organisation Method : (The PROMETHEE Method for Multiple Criteria Decision-Making). *Management Science 31,* 6, pp. 647–656. 28

Canfora, G., Di Penta, M., Esposito, R., and Villani, M. L. 2005. An Approach for QoS-aware Service Composition Based on Genetic Algorithms. In *GECCO '05 : Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, New York, NY, USA, 1069–1075. 78

Cao, L., Li, M., and Cao, J. 2007. Using Genetic Algorithm to Implement Cost-Driven Web Service Selection. *Multiagent Grid Syst. 3,* 1, 9–17. 78

Capra, L., Zachariadis, S., and Mascolo, C. 2005. Q-CAD : QoS and Context Aware Discovery Framework for Mobile Systems. In *ICPS*. 453456. 52

Cardoso, R. S. 2009. A Privacy-Enhanced Service-Oriented Middleware for Pervasive Computing. Ph.D. thesis. 77

Cavallaro, L. 2010. An Approach to Develop Self-assembling Self-adaptive Service Oriented Applications. Ph.D. thesis, Politecnico Di Milano. 18

Chafle, G., Dasgupta, K., Kumar, A., Mittal, S., and Srivastava, B. 2006. Adaptation in Web Service Composition and Execution. In *Proceedings of IEEE International Conference on Web Services (ICWS'06)*. 18, 25, 27, 32, 33, 103

Chang, H. and Lee, K. 2009. Quality-Driven Web Service Composition for Ubiquitous Computing Environment. In *Proceedings of the 2009 International Conference on New Trends in Information and Service Science*. IEEE Computer Society, Washington, DC, USA, 156–161. 20, 28, 80

References

Châtel, P., Malenfant, J., and Truck, I. 2010. QoS-based Late-Binding of Service Invocations in Adaptive Business Processes. In *ICWS*. 227–234. 13, 58

Colombo, M., Di Nitto, E., and Mauri, M. 2006. SCENE : A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In *Service-Oriented Computing – ICSOC 2006*, A. Dan and W. Lamersdorf, Eds. Lecture Notes in Computer Science, vol. 4294. Springer Berlin / Heidelberg, 191–202. 18, 26, 27, 33

Comes, D., Baraki, H., Reichle, R., Zapf, M., and Geihs, K. 2010. Heuristic Approaches for QoS-Based Service Selection. In *Service-Oriented Computing*, P. Maglio, M. Weske, J. Yang, and M. Fantinato, Eds. Lecture Notes in Computer Science, vol. 6470. Springer Berlin / Heidelberg, 441–455. 10.1007/978-3-642-17358-5-30. 77

Comes, D., Bleul, S., Weise, T., and Geihs, K. 2009. A Flexible Approach for Business Processes Monitoring. In *Proceedings of the 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*. DAIS '09. Springer-Verlag, Berlin, Heidelberg, 116–128. 85

Coulson, G., Blair, G. S., Davies, N., Robin, P., and Fitzpatrick, T. 1999. Supporting Mobile Multimedia Applications through Adaptive Middleware. *IEEE JOURNAL 17*, 1651–1659. 9

Crawley, E., Nair, R., Rajagopalan, B., and Sandick, H. 1998. A Framework for QoS-based Routing in the Internet. 5

Davies, D. L. and Bouldin, D. W. 1979. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence 1*, 224–227. 65, 66

Dept, G. L. and Lodi, G. 2002. End-To-End QoS-Aware Middleware Services. 9

Dey, A. K., Abowd, G. D., and Salber, D. 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Hum.-Comput. Interact. 16*, 97–166. 130

Di Penta, M., Esposito, R., Villani, M. L., Codato, R., Colombo, M., and Di Nitto, E. 2006. WS-Binder : a Framework to Enable Dynamic Binding of Composite Web Services. In *SOSE '06 : Proceedings of the 2006 international workshop on Service-oriented software engineering*. ACM, New York, NY, USA, 74–80. 13, 33, 57

Diestel, R. 2005. *Graph Theory*, Third ed. Graduate Texts in Mathematics, vol. 173. Springer-Verlag, Heidelberg. 86

Dobson, G., Lock, R., and Sommerville, I. 2005. QoSOnt : a QoS Ontology for Service-Centric Systems. In *EUROMICRO '05 : Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, Washington, DC, USA, 80–87. 52

Dobson, G. and Sanchez-Macian, A. 2006. Towards Unified QoS/SLA Ontologies. In *SCW '06 : Proceedings of the IEEE Services Computing Workshops*. IEEE Computer Society, Washington, DC, USA, 169–174. 52

Dumas, M., Benatallah, B., and Nezhad, H. R. M. 2008. Web Service Protocols : Compatibility and Adaptation. *IEEE Data Eng. Bull*, 40–44. 104

Dutra, R. G. and Junior, M. M. 2010. Dynamic Adaptive Middleware Services for Service Selection in Mobile Ad-Hoc Networks. In *MOBILWARE*. 189–202. 18, 19, 21, 23, 28, 80

Elabd, E., Coquery, E., and Hacid, M.-S. 2009. Compatibility and Replaceability Analysis of Timed Web Services Protocols. In *Proceedings of the 2009 Second International Conference on Computer and Electrical Engineering - Volume 02*. ICCEE '09. IEEE Computer Society, Washington, DC, USA, 15–19. 104

Epifani, I., Ghezzi, C., Mirandola, R., and Tamburrelli, G. 2009. Model Evolution by Run-time Parameter Adaptation. *Software Engineering, International Conference on 0*, 111–121. 85

Erradi, A. and Maheshwari, P. 2005. wsBus : QoS-Aware Middleware for Reliable Web Services Interactions. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*. EEE '05. IEEE Computer Society, Washington, DC, USA, 634–639. 18

Erradi, A., Maheshwari, P., and Tosic, V. 2006. Policy-Driven Middleware for Self-Adaptation of Web Services Compositions. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*. Middleware '06. Springer-Verlag New York, Inc., New York, NY, USA, 62–80. 18

Fredj, M. 2009. Dynamic Reconfiguration of Service-oriented Architectures. Ph.D. thesis, University of Paris 6. 18, 103

Funk, C., Schultheis, A., Linnhoff-Popien, C., Mitic, J., and Kuhmunch, C. 2007. Adaptation of Composite Services in Pervasive Computing Environments. *International Conference on Pervasive Services 0*, 242–249. 26, 27

Gao, C., Cai, M., and Chen, H. 2007. QoS-aware Service Composition Based on Tree-Coded Genetic Algorithm. In *COMPSAC '07 : Proceedings of the 31st Annual International Computer Software and Applications Conference*. IEEE Computer Society, Washington, DC, USA, 361–367. 78

González, L. and Ruggia, R. 2010. Towards Dynamic Adaptation within an ESB-based Service Infrastructure Layer. In *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*. MONA '10. ACM, New York, NY, USA, 40–47. 18

References

Grigori, D., Corrales, J. C., Bouzeghoub, M., and Gater, A. 2010. Ranking BPEL Processes for Service Discovery. *IEEE Transactions on Services Computing 3*, 178–192. 104, 125

Gu, X. 2004. Spidernet : a quality-aware service composition middleware. Ph.D. thesis, Champaign, IL, USA. 18

Hilari, M. O. 2009. Quality of Service (QoS) in SOA Systems. M.S. thesis, Universitat Politècnica de Catalunya. 10

Hobbs, J. R. and Pan, F. 2006. Time Ontology in OWL. http ://www.w3.org/TR/owl-time/. 46

Hogben, L., Greenbaum, A., Brualdi, R., and Mathias, R. 2007. *Handbook of Linear Algebra*. Chapman & Hall. 122

hong SHEN, Y. and hu YANG, X. 2010. A Self-Optimizing QoS-aware Service Composition Approach in a Context Sensitive Environment. *Journal of Zhejiang University-SCIENCE C (Computers and Electronics) 12,* 3, 221–238. 78

Ibrahim, N., Le Mouël, F., and Frénot, S. 2009. MySIM : a Spontaneous Service Integration Middleware for Pervasive Environments. In *Proceedings of the 2009 international conference on Pervasive services*. ICPS '09. ACM, New York, NY, USA, 1–10. 18

Ibrahim, N., Le Mouël, F., and Frénot, S. 2011. Semantic Service Substitution in Pervasive Environments. *International Journal of Services, Economics and Management (IJSEM)*. "Service-Oriented Engineering" special issue. 102, 103

ISO 1994. *Quality Management and Quality Assurance Vocabulary*, 8402 ed. ISO. 5

Issarny, V., Bennaceur, A., and Bromberg, Y.-D. 2011. Middleware-layer connector synthesis : Beyond state of the art in middleware interoperability. In *SFM*. 217–255. 131

Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., Gerosa, M., and Hamida, A. 2011. Service-Oriented Middleware for the Future Internet : State of the Art and Research Directions. *Journal of Internet Services and Applications 2*, 23–45. 10, 11, 18, 23, 32

Issarny, V., Sacchetti, D., Tartanoglu, F., Sailhan, F., Chibout, R., Levy, N., and Talamona, A. 2005. Developing ambient intelligence Systems : A Solution Based on Web Services. *Journal of Automated Software Engineering 2004*, 2005. 131

ITU-T Rec. 1993. *Terms and Definitions Related to Quality of Service and Network Performance Including Dependability*, E.800 ed. ITU-T Rec. 5

Jaeger, M. C. 2006. Modelling of Service Compositions : Relations to Business Process and Workflow Modelling. In *ICSOC Workshops*. 141–153. 113

Jaeger, M. C. and Mühl, G. 2007. QoS-based Selection of Services : The Implementation of a Genetic Algorithm. In *Kommunikation in Verteilten Systemen (KiVS 2007) Industriebeträge, Kurzbeiträge und Workshops*, T. Braun, G. Carle, and B. Stiller, Eds. VDE Verlag, Berlin und Offenbach, Bern, Switzerland, 359–350. 78

Jaeger, M. C., Muhl, G., and Golze, S. 2005. QoS-Aware Composition of Web Services : A Look at Selection Algorithms. In *ICWS '05 : Proceedings of the IEEE International Conference on Web Services*. IEEE Computer Society, Washington, DC, USA, 807–808. 26, 28, 29

Jafarpour, N. and Khayyambashi, M. 2010. QoS-aware Selection of Web Service Composition Based on Harmony Search Algorithm. In *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*. Vol. 2. 1345 –1350. 78

Jiang, H., Yang, X., Yin, K., Zhang, S., and Cristoforo, J. A. 2011. Multi-path QoS-Aware Web Service Composition using Variable Length Chromosome Genetic Algorithm. *Information Technology Journal 10,* 1, 113–119. 78

Jin, J., Zhang, Y., Cao, Y., Pu, X., and Li, J. 2010. ServiceStore : A Peer-to-Peer Framework for QoS-Aware Service Composition. In *Network and Parallel Computing*. Lecture Notes in Computer Science, vol. 6289. Springer Berlin / Heidelberg, 190–199. 79

Kai Shuang, S. Y. and Su, S. 2009. TTS-Coded Genetic Algorithm for QoS-driven Web Service Selection. In *IEEE International Conference on Communications Technology and Applications, ICCTA'09*. 885–890. 78

Kalasapur, S., Kumar, M., and Shirazi, B. 2006. Evaluating Service Oriented Architectures (SOA) in Pervasive Computing. In *PERCOM '06 : Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications*. IEEE Computer Society, Washington, DC, USA, 276–285. 48

Kalasapur, S., Kumar, M., and Shirazi, B. A. 2007. Dynamic Service Composition in Pervasive Computing. *IEEE Trans. Parallel Distrib. Syst. 18,* 7, 907–918. 18, 22, 24, 26, 27, 28, 29, 30, 31, 32

Kareliotis, C., Vassilakis, C., Rouvas, E., and Georgiadis, P. 2009. QoS-aware Exception Resolution for BPEL Processes : a Middleware-based Framework and Performance Evaluation. *Int. J. Web Grid Serv. 5*, 284–320. 18, 19, 20, 21, 102, 103

Kazhamiakin, R., Benbernou, S., Baresi, L., Plebani, P., Uhlig, M., and Barais, O. 2010. Adaptation of Service-Based Systems. In *Service Research Challenges and Solutions for the Future Internet*, M. Papazoglou, K. Pohl, M. Parkin, and A. Metzger, Eds. Lecture Notes in Computer Science, vol. 6500. Springer Berlin / Heidelberg, 117–156. 29, 30, 31, 32, 33

References

Kim, H. M., Sengupta, A., and Evermann, J. 2005. MOQ : Web Services Ontologies for QOS and General Quality Evaluations. In *European Conference on Information Systems (ECIS 2005)*. 52

Kobti, Z. and Zhiyang, W. 2007. An Adaptive Approach for QoS-Aware Web Service Composition Using Cultural Algorithms. In *Australian Conference on Artificial Intelligence*, M. A. Orgun and J. Thornton, Eds. Lecture Notes in Computer Science, vol. 4830. Springer, 140–149. 78

Koskela, M., Rahikainen, M., and Wan, T. 2003. Software Development Methods : SOA vs. CBD, OO and AOP. 10

Kuehne, B., Estrella, J., Peixoto, M., Tavares, T., Santana, R., and Santana, M. 2010. Dynamic Web Service Composition Middleware : A New Approach for QoS Guarantees. In *Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on*. 174 –177. 18

Lécué, F. 2009. Optimizing QoS-Aware Semantic Web Service Composition. In *The Semantic Web - ISWC 2009*. Lecture Notes in Computer Science, vol. 5823. Springer Berlin/Heidelberg, 375–391. 78

Lee, S. and Lee, J. 2006. Dynamic Service Composition Model for Ubiquitous Service Environments. In *PRIMA*. 742–747. 27

Li, J., Zhao, Y., Liu, M., Sun, H., and Ma, D. 2010. An Adaptive Heuristic Approach for Distributed QoS-based Service Composition. *Computers and Communications, IEEE Symposium on 0*, 687–694. 76, 79

Lim, E. and Thiran, P. 2010. Sustaining High-Availability and Quality of Web Services. In *ICWE Workshops*. 560–565. 103

Liu, C.-H., Chen, S.-L., and Li, X.-Y. 2008. A WS-BPEL Based Structural Testing Approach for Web Service Compositions. In *Proceedings of the 2008 IEEE International Symposium on Service-Oriented System Engineering*. IEEE Computer Society, Washington, DC, USA, 135–141. 91

Liu, D., Shao, Z., Yu, C., and Fan, G. 2009. A Heuristic QoS-Aware Service Selection Approach to Web Service Composition. In *Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science*. IEEE Computer Society, Washington, DC, USA, 1184–1189. 78, 80

Lloyd, S. 1957. Least Squares Quantization in PCM. Unpublished memorandum, Bell Laboratories. 73

Lupu, E. and Sloman, M. 1997. A Policy Based Role Object Model. In *EDOC '97 : Proceedings of the 1st International Conference on Enterprise Distributed Object Computing*. IEEE Computer Society, Washington, DC, USA, 36–47. 131

Maia, M. E., Rocha, L. S., and Andrade, R. M. 2009. Requirements and Challenges for Building Service-Oriented Pervasive Middleware. In *Proceedings of the 2009 international conference on Pervasive services*. ICPS '09. ACM, New York, NY, USA, 93–102. 18, 49

Marchetti, C., Pernici, B., and Plebani, P. 2004. A Quality Model for Multichannel Adaptive Information. In *WWW Alt. '04 : Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, New York, NY, USA, 48–54. 47, 52

Maximilien, E. M. and Singh, M. P. 2004. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing 8,* 5, 84–93. 52

Mcnamara, L., Mascolo, C., and Capra, L. 2006. Trust and Mobility Aware Service Provision for Pervasive Computing. In *First International Workshop on Requirements and Solutions for Pervasive Software Infrastructures*. 52

Messmer, B. and Bunke, H. 2000. Efficient Subgraph Isomorphism Detection : A Decomposition Approach. *Knowledge and Data Engineering, IEEE Transactions on 12,* 2 (mar/apr), 307 –323. 89

Milligan, G. and Cooper, M. 1985. An Examination of Procedures for Determining the Number of Clusters in a Data Set. *Psychometrika 50,* 2 (June), 159–179. 65

Min, D., Kim, E., Lee, Y., Kang, G., and Clark, J. B. 2007. Web Services Quality Model. http ://www.oasis-open.org/committees/tc_home.php ?wg_abbrev=wsqm. 42

Mostofa Akbar, M., Sohel Rahman, M., Kaykobad, M., Manning, E. G., and Shoja, G. C. 2006. Solving the Multidimensional Multiple-choice Knapsack Problem by Constructing Convex Hulls. *Comput. Oper. Res. 33*, 1259–1273. 80

Nahrstedt, K., Xu, D., Wichadakul, D., and Li, B. 2001. QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. *IEEE Communications Magazine 39*, 140–148. 9

Nilsson, N. J. 1980. Principle of Artificial Intelligence. 96

Nitto, E. D., Ghezzi, C., Metzger, A., Papazoglou, M., and Pohl, K. 2008. A Journey to Highly Dynamic, Self-Adaptive Service-Based Applications. *Automated Software Engg. 15,* 3-4, 313–341. 49

Oldham, N., Verma, K., Sheth, A., and Hakimpour, F. 2006. Semantic WS-Agreement Partner Selection. In *WWW '06 : Proceedings of the 15th international conference on World Wide Web*. ACM, New York, NY, USA, 697–706. 44

Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. 2002. Semantic Matching of Web Services Capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*. ISWC '02. Springer-Verlag, London, UK, UK, 333–347. 97, 102, 103

References

Papaioannou, I. V., Tsesmetzis, D. T., Roussaki, I. G., and Anagnostou, M. E. 2006. A QoS Ontology Language for Web-Services. In *AINA '06 : Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*. IEEE Computer Society, Washington, DC, USA, 101–106. 48, 52

Pautasso, C. and Alonso, G. 2005. Flexible Binding for Reusable Composition of Web Services. In *Proceedings of the 4th Workshop on Software Composition (SC 2005)*. Edinburgh, Scotland. 13, 57

Ponge, J., Benatallah, B., Casati, F., and Toumani, F. 2010. Analysis and Applications of Timed Service Protocols. *ACM Trans. Softw. Eng. Methodol. 19*, 11 :1–11 :38. 104

Raverdy, P.-G., Riva, O., de La Chapelle, A., Chibout, R., and Issarny, V. 2006. Efficient Context-aware Service Discovery in Multi-Protocol Pervasive Environments. In *MDM*. 3. 74

Resta, G. and Santi, P. 2008. WiQoSM : An Integrated QoS-Aware Mobility and User Behavior Model for Wireless Data Networks. *IEEE Transactions on Mobile Computing 7,* 2, 187–198. 50, 52, 53

Robertson, N. and Seymour, P. D. 1995. Graph Minors .XIII. The Disjoint Paths Problem. *Journal of Combinatorial Theory, Series B 63,* 1, 65–110. 89

Rosenberg, F. 2009. QoS-Aware Composition of Adaptive Service-Oriented Systems. Ph.D. thesis, Technischen Universität Wien, Fakultät für Informatik. 18, 19, 21, 22, 23, 27

Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., Mamelli, A., and Scholz, U. 2009. Software engineering for self-adaptive systems. Springer-Verlag, Berlin, Heidelberg, Chapter MUSIC : Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments, 164–182. 18

Ruta, M., Di Noia, T., Di Sciascio, E., Paolucci, M., Scioscia, F., and Tinelli, E. 2008. A Semantic-based Registry Enabling Discovery, Composition and Substitution of Pervasive Services. In *Proceedings of the Seventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*. MobiDE'08. ACM, New York, NY, USA, 63–70. 103

Schmidt, D. C., Levine, D. L., and Mungee, S. 1997. The Design of the TAO Real-Time Object Request Broker. *Computer Communications 21*, 294–324. 9

Shankar, M., De Miguel, M., and Liu, J. 1999. An End-to-end QoS Management Architecture. In *Real-Time Technology and Applications Symposium, 1999. Proceedings of the Fifth IEEE*. 176–189. 9

Siljee, J., Bosloper, I., Nijhuis, J., and Hammer, D. 2005. DySOA : Making Service Systems Self-Adaptive. In *International Conference on Service Oriented Computing (ICSOC05*. 12–15. 18, 33

Sousa, J. P., Poladian, V., Garlan, D., Schmerl, B., and Shaw, M. 2006. Task-based Adaptation for Ubiquitous Computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part C : Applications and Reviews, Special Issue on Engineering Autonomic Systems 36(3)*, 328–340. 18, 26, 28, 29, 31, 32

Van Hoecke, S., Vlaeminck, K., De Turck, F., and Dhoedt, B. 2005. Open Web Services-based Middleware for Brokering of Composed eHomeCare Services. *Proceedings of Middleware for Web Services MWS 2005*, 46–52. 18

Vanegas, R., Zinky, J., Loyall, J., Karr, D., Schantz, R., and Bakken, D. 1998. QuO's Runtime Support for Quality of Service in Distributed Objects. In *Middleware 98 : the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing*. 9

Vanrompay, Y., Rigole, P., and Berbers, Y. 2008. Genetic Algorithm-based Optimization of Service Composition and Deployment. In *SIPE '08 : Proceedings of the 3rd international workshop on Services integration in pervasive environments*. ACM, New York, NY, USA, 13–18. 78

Verma, K., Gomadam, K., Sheth, A. P., Miller, J. A., and Wu, Z. 2005. The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. Tech. rep. 18, 21, 23

Wac, K. 2005. Towards QoS-Awareness of Context-Aware Mobile Applications and Services. 751–760. 5, 130

Wichadakul, D., Nahrstedt, K., Gu, X., and Xu, D. 2001. 2KQ+ : An Integrated Approach of QoS Compilation and Reconfigurable, Component-Based Run-Time Middleware for the Unified QoS Management Framework. In *Middleware '01 : Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. Springer-Verlag, London, UK, 373–394. 6, 9

Xia, Y.-m., Chen, J.-l., and Meng, X.-w. 2008. On the Dynamic Ant Colony Algorithm Optimization Based on Multi-pheromones. In *Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*. IEEE Computer Society, Washington, DC, USA, 630–635. 78

Xiao, W., Soong, B. H., Law, C. L., and Guan, Y. L. 2004. Evaluation of Heuristic Path Selection Algorithms for Multi-Constrained QoS Routing. In *IEEE Int'l Conf. Networking, Sensing and Control*. 112–116. 28, 29

Xiao, Y., Wu, W., 0009, W. W., and He, Z. 2007. Efficient Algorithms for Node Disjoint Subgraph Homeomorphism Determination. *CoRR abs/0709.1227*. 83, 84, 86, 88, 89, 90, 96, 100, 115, 129

References

Yang, K., Galis, A., and Chen, H.-H. 2009. QoS-Aware Service Selection Algorithms for Pervasive Service Composition in Mobile Wireless Environments. *Mobile Networks and Applications*. 20, 26, 28, 47

Yu, T., Zhang, Y., and Lin, K.-J. 2007. Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. *ACM Trans. Web 1,* 1, 6. 14, 26, 35, 60, 64, 77

Zeng, L., Benatallah, B., H.H. Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H. 2004. QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng. 30*, 311–327. 18, 60

Zeng, L., Ngu, A. H., Benatallah, B., Podorozhny, R., and Lei, H. 2008. Dynamic Composition and Optimization of Web Services. *Distrib. Parallel Databases 24*, 45–72. 102

Zhai, Y., Zhang, J., and Lin, K.-J. 2009. SOA Middleware Support for Service Process Reconfiguration with End-to-End QoS Constraints. In *Proceedings of the 2009 IEEE International Conference on Web Services*. ICWS'09. IEEE Computer Society, Washington, DC, USA, 815–822. 18, 19, 20

Zhang, B., Shi, Y., and Xiao, X. 2007. A Policy-Driven Service Composition Method for Adaptation in Pervasive Computing Environment. *The Computer Journal 53,* 2 (December), 152–165. 20, 31, 32

Zhang, C., Su, S., and Chen, J. 2006. A Novel Genetic Algorithm for QoS-Aware Web Services Selection. In *Data Engineering Issues in E-Commerce and Services*, S. Berlin/Heidelberg, Ed. 224–235. 78

Zhang, Y., Zhang, S., and Songqiao, H. 2006. A New Methodology of QoS Evaluation and Service Selection for Ubiquitous Computing. In *Wireless Algorithms, Systems, and Applications*, S. B. . Heidelberg, Ed. 69–80. 20

Zheng, Z. and Lyu, M. R. 2008. A QoS-Aware Middleware for Fault Tolerant Web Services. In *Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*. IEEE Computer Society, Washington, DC, USA, 97–106. 18

Zongkai YANG, Chaowang SHANG, Q. L. and ZHAO, C. 2010. A Dynamic Web Services Composition Algorithm Based on the Combination of Ant Colony Algorithm and Genetic Algorithm. *Journal of Computational Information Systems 6,* 8, 2617–2622. 78