Semantic Middleware for Service-Oriented Pervasive Computing

Sonia Ben Mokhtar

▶ **To cite this version:**

Sonia Ben Mokhtar. Semantic Middleware for Service-Oriented Pervasive Computing. Computer Science [cs]. Université Pierre et Marie Curie - Paris VI, 2007. English. NNT : . tel-00469457

HAL Id: tel-00469457
https://theses.hal.science/tel-00469457

Submitted on 1 Apr 2010

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.

N° Ordre de la thèse :

THÈSE

présentée

DEVANT L'UNIVERSITÉ DE PARIS 6

pour obtenir

le grade de: *DOCTEUR DE L'UNIVERSITÉ DE PARIS 6*

Mention: **Informatique**

PAR

Sonia BEN MOKHTAR

Équipe d'accueil: **INRIA, Projet ARLES**

École Doctorale: **Informatique, Télécommunications et Electronique de Paris**

TITRE DE LA THÈSE:

Intergiciel Sémantique pour les Services de l'Informatique Diffuse

SOUTENUE LE x / xx / xxxx devant la commission d'Examen

COMPOSITION DU JURY

Boualam BENATALLAH (Université de New South Wales, Australie)	Rapporteur
Gordon BLAIR (Université de Lancaster, GB)	Rapporteur
Licia CAPRA (Université Collège de Londres, GB)	Examinateur
Pierre SENS (Université de Paris 6, Paris)	Examinateur
Valérie ISSARNY (INRIA)	Directrice de Thèse
Nikolaos GEORGANTAS (INRIA)	Co-Directeur de Thèse

Abstract

The pervasive computing vision introduced by Mark Weiser in the early 90's, results from the convergence of powerful, small, affordable computing devices with networking technologies that tie them all together. Still, the development of software systems for pervasive computing environments requires dealing with numerous challenges that are primarily due to the dynamics, mobility and heterogeneity inherent to these environments.

Middleware technologies that deal with the dynamics and homogenize the diversity of technologies appear as a major enabler for the development of pervasive computing software systems. Further, service-oriented middleware (SOM) where functional and non-functional capabilities provided by pervasive networked resources are abstracted as services appear to be the most appropriate for pervasive computing environments. However, the dynamic discovery and composition of networked services by applications, which constitute two of the main functionalities of a SOM, require service requesters and providers to agree on both the functional and non-functional semantics of service capabilities. This, cannot be achieved on a syntactic basis in open pervasive computing environments. A promising approach then, relies on the semantic modelling of services functional and non-functional capabilities.

In the above direction, this thesis introduces a semantic, service-oriented middleware for pervasive computing. The most significant contributions are: (1) an extensible semantic service model and its associated conformance relations; (2) an efficient semantic service registry for highly interactive pervasive environments; and (3) the support for adaptive QoS-aware service composition that allows taking full advantage of the diverse capabilities of each pervasive environment.

Contents

1 Introduction **1**
1.1 Towards Service-Oriented Pervasive Computing 3
1.2 Thesis Contribution and Document Structure 5

2 Middleware for Service-Oriented Pervasive Computing: Vision and State Of The Art **9**
2.1 Pervasive Computing Environments 10
2.2 Service-Oriented Pervasive Computing 13
2.3 Middleware for Pervasive Computing Environments: State of The Art . . . 16
2.3.1 Proprietary Middleware 17
2.3.2 Interoperable Middleware 20
2.3.3 Semantic-aware Middleware 22
2.3.4 Discussion 25
2.4 Semantic, Service-Oriented Middleware for Pervasive Computing 26

3 Semantic Specification of Pervasive Services **31**
3.1 Semantic Service Description Languages: State Of The Art 33
3.2 Semantic Service Specification Model 37
3.2.1 Provided and Required Capabilities 37
3.2.2 Conversation Specification 40
3.2.3 Non-functional Properties 42
3.3 Formalizing the Semantic Service Model 44
3.4 Concluding Remarks 49

4 Efficient Semantic Service Registry for Pervasive Computing Environments **51**
4.1 Efficient Semantic Service Matching: State Of The Art 52
4.1.1 Matching Semantic Service Capabilities 52
4.1.2 Cost of Semantic Service Matching 54
4.1.3 Optimizations to Semantic Service Matching 55

4.1.4 Discussion . . . 57
4.2 Efficient Semantic Service Registry . . . 58
4.2.1 Service Matching . . . 59
4.2.2 Efficient Semantic Service Matching . . . 74
4.2.3 Registry Service Index . . . 79
4.2.4 Service Publication . . . 80
4.2.5 Service Location . . . 86
4.3 Assessing the Efficiency of the Semantic Service Registry . . . 88
4.4 Concluding Remarks . . . 88

5 Service Composition in Pervasive Computing Environments 91
5.1 Service Composition: State Of The Art . . . 92
5.1.1 Interface-Based Service Composition . . . 93
5.1.2 Conversation-Based Service Composition . . . 94
5.2 Semantic-, QoS-aware Conversation-driven Conversation Integration: Overview 97
5.3 Service Discovery Client . . . 98
5.4 Service Conformance . . . 103
5.4.1 Data Flow and Data Constraints . . . 104
5.4.2 Ordering Constraints . . . 107
5.5 Service Coordination . . . 115
5.5.1 Problem Definition . . . 115
5.5.2 Integrating Service Conversations . . . 117
5.5.3 Support of Conversation Interleaving . . . 120
5.5.4 Support of Adaptive User Tasks . . . 121
5.6 Matching Global Non-Functional Properties of Composed User Tasks . . . 124
5.7 On the fly User Task Realization for Meeting Pervasive Computing Requirements . . . 127
5.8 Assessing the Efficiency of the Composition Model . . . 133
5.9 Concluding Remarks . . . 134

6 PERSE: Pervasive Semantic-aware Middleware 137
6.1 Baseline MUSDAC Middleware for Multi-Network, Multi-Protocol Service Discovery in Pervasive Computing Environments . . . 137
6.2 The PERSE Middleware Architecture . . . 139
6.2.1 Multi-Network Management . . . 141
6.2.2 Service Discovery in PERSE . . . 141
6.2.3 Service Composition in PERSE . . . 142
6.2.4 Service Access in PERSE . . . 142
6.2.5 Interoperable Service Description Language . . . 143
6.3 Prototype Implementation and Performance Evaluation . . . 146
6.3.1 Performance of the Prime Number-Based Ontology Encoding Algorithm . . . 148
6.3.2 Cost of Legacy to ISDL Translation . . . 149

6.3.3 Performance of the Interoperable Service Matching 150
6.3.4 Efficiency of the PERSE Service Registry 154
6.3.5 Performance of the QoS-Aware, Conversation-Based Service Composition 158

7 Conclusion **163**
7.1 Contribution 164
7.2 Perspectives 167

A Proofs **169**
A.1 Proof of the property [Prop 1] 169
A.2 Proof of the property [Prop 2] 170
A.3 Proof of the property [Prop 3] 171
A.4 Proof of the transitivity of the relation FunctionalCapabilityMatch() 171
A.5 Proof of transitivity of the relation ConceptMatch() 172
A.6 Complexity of the ConversationMatch() relation 175

Bibliography **178**

List of Figures

2.1 Pervasive Computing Environments . . . 10
2.2 SOA Actors . . . 13
2.3 The Role of Service Aggregator . . . 14
2.4 SOA Conceptual Elements . . . 15
2.5 Asserted (left) and Inferred (right) Hierarchies of Concepts . . . 23
2.6 Semantic Service-Oriented Middleware for Pervasive Computing . . . 28

3.1 Semantic Service Specification Model . . . 38
3.2 Required and Provided Capabilities . . . 40
3.3 Example of an Ontology of Resources . . . 40
3.4 EASY-COM Service . . . 41
3.5 Data Flow Graphical Representation . . . 42
3.6 Specifying Non-Functional Properties . . . 43
3.7 Example of QoS Ontology . . . 44
3.8 Basic Control Flow Patterns Modelled with Finite State Automata . . . 47

4.1 Registry Architecture Overview . . . 58
4.2 Ontology Example . . . 60
4.3 Matching and Evaluating the Semantic Distance Between Capabilities . . . 64
4.4 Degree of Match Between Concepts . . . 71
4.5 Encoded Resource Ontology . . . 79
4.6 Indexing Graphs of Related Capabilities . . . 80
4.7 Service Publication Example . . . 86

5.1 Composition Models . . . 95
5.2 Flexibility of the Conversation-Driven Conversation Integration . . . 97
5.3 Overall Service Composition Process . . . 99
5.4 User Task Conversation . . . 100
5.5 Selected Pervasive Services . . . 103
5.6 Data Flow and Data Constraints Example . . . 106
5.7 Filtering Automaton for the Support of Conversation Integration . . . 111
5.8 Filtering Automaton for the Support of Conversation Interleaving . . . 113
5.9 Raw Automaton . . . 119
5.10 Data Dependency Graph . . . 122

5.11 Rescheduling Automaton . . . 123
5.12 Reduction Rules for Estimating Quantitative Properties . . . 127
5.13 QoS Formulae of the EASY-Movie User Task . . . 128
5.14 On The Fly User Task Realization . . . 131
5.15 QoS-aware User Task Realization . . . 132

6.1 MUSDAC Architecture . . . 139
6.2 PERSE Middleware . . . 140
6.3 Example of an ISD Description . . . 144
6.4 Interoperability Enabled by ISDL . . . 146
6.5 SAWSDL Editor . . . 147
6.6 Matching Using Online Semantic Reasoning . . . 151
6.7 PERSE Matching Performance . . . 153
6.8 PERSE Registry Scalability . . . 154
6.9 Time to Organize a PERSE Registry . . . 156
6.10 Publication Time in a PERSE Registry . . . 156
6.11 Benefit Of Grouping Service Advertisements . . . 157
6.12 WSDL vs PERSE Service Registries . . . 158
6.13 Performance of the Composition Algorithm (Increasing the Number of Services) . . . 160
6.14 Performance of the Composition Algorithm (Increasing the Task Size) . . . 161
6.15 Performance of QoS-aware Service Composition . . . 162

List of Tables

1 List of Symbols viii

2.1 Service-Oriented Pervasive Environments 17

3.1 SOA Concepts Supported by Semantic Service Specification Languages . . . 36

5.1 Capabilities of the EASY-Movie User Task and Selected Pervasive Services 102
5.2 Matching Between Capabilities of the EASY-Movie User Task and Selected Pervasive Services 104
5.3 Probabilistic Quantitative Properties Evaluation 126
5.4 Pessimistic Quantitative Properties Evaluation 127

6.1 Comparison of Encoding Length of a Single Class (in bits) 148
6.2 Legacy to ISDL Translation (in micro-seconds) 150

Notation	Meaning
$\mathcal{O}$	Set of ontologies
$\mathcal{N}$	Set of concepts across $\mathcal{O}$
$\mathcal{S}$	Set of pervasive services
$\mathcal{C}$	Set of capabilities supported by pervasive services
$\mathcal{T}$	Set of user tasks
I	Set of service inputs
O	Set of service outputs
cat	A service category
$\mathcal{P}$	Set of non-functional properties
$\mathcal{P}_{\mathcal{QL}}$	Set of qualitative non-functional properties
$\mathcal{P}_{\mathcal{QN}}$	Set of quantitative non-functional properties
A	Automaton
Σ	Set of automaton symbols
δ	Automaton transition function
Q	Set of automaton states
F	Set of automaton final states
st_0	Automaton initial state
$\oplus$	some-values from annotation type
$\otimes$	all-values from annotation type
$m(p)$	mean value of the property p
$\Delta(p)$	standard deviation of the property p
τ_i	coefficient for the evaluation of the semantic distance between concepts
w_i	coefficient specifying the preference among QoS properties
λ_i	coefficient specifying the preference between service functional and QoS properties

Table 1: List of Symbols

Chapter 1

Introduction

Pervasive computing [Weiser, 1991] envisions the unobtrusive diffusion of computing and networking resources in physical environments, enabling users to benefit from their provided functionalities anywhere and at any time. This is further realized in a user-centric, interactive way, i.e., where the system seamlessly adapts to the characteristics, preferences and current situation of the user and his/her surrounding environment. Assisting mobile users in their daily tasks by combining available networked functionalities and adapting to the specifics of each pervasive environment is one of the major challenges in achieving the pervasive computing vision. To illustrate the kind of situations that we expect to make commonplace through our research, we present the following scenario inspired from [Ducatel et al., 2001, Ben Mokhtar et al., 2006a]:

"...Today, Rozalie is taking a long haul flight to Australia, where she has an important working seminar. For such a working trip, Rozalie can now travel much lighter than a decade ago, when she had to carry a collection of so-called personal computing devices (laptop PC, mobile phone, electronic organizer, and even portable beamers and printers). Her computing system is now reduced to a single device, EASY-Com, that she wears on her wrist. Rozalie does not have to stop at the security check, as EASY-Com deals with her ID verification while she is walking through metal detectors and passport controls. Today, exceptionally, Rozalie arrives early at the airport. When she enters the V.I.P room, nobody is there. She decides to watch a movie while waiting for the boarding announcement and

having a massage in a massage chair. EASY-Com uses the EASY-Movie application, one of the various embedded applications on Rozalie's wrist. EASY-Movie is able to discover and browse the content of available video servers, as well as to select the most appropriate display devices in Rozalie's reach (e.g., the one having the largest screen). Furthermore, EASY-Movie is able to adapt the surrounding environment to Rozalie's preferences (e.g., room lighting, movie sound level). Hence, EASY-Movie starts displaying the movie selected by Rozalie on a large plasma screen that was disseminating the news. Half an hour later, EASY-Com informs Rozalie that she has to go for boarding. After getting on the plane and paying attention to the security demonstration, Rozalie is asked by EASY-Movie whether she would like to continue watching the movie on the personal LCD panel mounted on the seat back in front of her. When she arrives to Sydney's airport, a rented car has been booked for her and is in the airport parking. While walking to the car, she receives a phone call from her husband Stan. This phone call is managed by the EASY-Phone application, which allows Rozalie to benefit from the devices in her reach to improve the quality of her vocal and video communications. The car opens as she approaches thanks to EASY-Com, which manages to identify her to the car identification system. When she enters the car, EASY-Phone transfers her phone call to the car audio system, which is more comfortable than her hands-free headset. Furthermore, as her husband is using their home video system, the video signal is now displayed by EASY-Phone on the car LCD screen...".

In this scenario, a number of key concepts of pervasive computing are highlighted, among which the ability of users to access relevant functionalities anywhere and at any time. For instance, while waiting for her flight at the airport, during her flight or inside a rental car, Rozalie discovers and accesses the diverse networked functionalities available in her vicinity. This feature is governed by the environment's dynamics. While moving, Rozalie may notice the appearance of new functionalities and the disappearance of others. For instance, the identification system of the rental car becomes active when Rozalie approaches the car. Another feature of pervasive environments is the ability of applications to adapt to the current situation of the user: upon the discovery of the LCD panel in front of Rozalie's seat on the plane, the EASY-Movie application proposes to display the movie on that screen. Finally, a key feature is the combination of functionalities available at the

specific time and location to realize user tasks: EASY-Movie combines the video streaming functionality of a video server with the display functionality of a plasma screen and the room lighting system.

1.1 Towards Service-Oriented Pervasive Computing

Realizing the above illustrated vision of pervasive computing requires dealing with a number of issues, mainly due to the environment's heterogeneity, dynamics and user-centrism. Middleware technologies that deal with the dynamics, homogenize the diversity of technologies in pervasive environments while providing base support to user-centric considerations appear as a major enabler for the development of pervasive computing software applications.

Among the various investigated middleware paradigms, service-oriented middleware (SOM) appears to be most appropriate for pervasive environments. Indeed, building upon the Service-Oriented Architecture (SOA), functionalities provided by software and hardware resources of the pervasive environment may conveniently be abstracted as services. These services are independent software entities with well defined interfaces, and may be accessed without any knowledge about their underlying technologies, such as hardware platforms, operating systems, programming languages. In this context, the role of SOM is to provide applications with middleware functionalities that allow them to dynamically discover and access networked services that fit their requirements, and to dynamically compose these services to help users in realizing their daily tasks. Hence, service discovery, access and composition are three essential SOM functionalities, which obtain particular meaning and importance in pervasive environments.

Nevertheless, the affluence of SOM technologies and platforms that have been put forward to address the heterogeneity and dynamics of pervasive environments has engendered a new kind of heterogeneity, i.e., middleware heterogeneity. Specifically, this heterogeneity concerns the protocols associated to base middleware functionalities, which are service discovery and service access. This heterogeneity is further increased by the heterogeneity of networks in which service providers and requesters may reside. Thus, a SOM for pervasive

computing should provide multi-protocol and multi-network interoperability mechanisms. As a result, service providers and requesters are able to locate and interact with each other even if they employ heterogeneous underlying middleware and networking technologies.

Still, even after interoperability has been established at the networking and middleware levels, the dynamic discovery and composition of networked services by applications further require service providers and requesters to agree on the semantics of services, so that they can integrate and interact in a way that guarantees dependable service provisioning and consumption. Such an agreement may be carried out at the syntactic level, assuming that service providers and requesters use a common syntax for denoting service semantics. This assumption is actually made by most software platforms for pervasive computing (e.g., Aura [Sousa and Garlan, 2002], Gaia [Shiva Chetan and Campbell, 2005], WSAMI [Issarny et al., 2005], Oxygen [Walker, 2004], Pico [Kumar et al., 2003]). However, such strong assumption that services are described with identical terms worldwide, is hardly achievable in open pervasive environments. This raises the issue of *syntactic heterogeneity* of service descriptions. Then, a promising approach towards addressing syntactic heterogeneity relies on semantic modelling of service features by employing technologies that come from the knowledge representation domain and have been identified in this decade as a key enabler for the Semantic Web [Berners-Lee et al., 2001]. Semantic modelling enables global common understanding of service semantics as well as machine reasoning on it. Research efforts have then investigated semantic-aware middleware for pervasive computing [Masuoka et al., 2003, Singh et al., 2005, Chakraborty et al., 2006, Chakraborty et al., 2005]. Nevertheless, assessing the conformance between service semantics as announced by service providers and requested by service requesters induces costly semantic reasoning (in terms of time and computation), which makes existing solutions inappropriate for the highly interactive and resource constrained pervasive environment.

Besides dealing with the functional features of services, user-centrism of pervasive environments calls for the awareness of service non-functional features, i.e., Quality of Service (QoS). QoS is the set of information related with a service (e.g., latency, availability, security), which affects the service's ability to satisfy users requirements [Liu, 2006]. QoS plays a decisive role in enhancing the user's experience of the pervasive environment.

Hence, service discovery, access and composition functionalities provided by a SOM should be aware of the QoS characteristics of services, and should take into account the respective requirements of users. Same as for functional features, semantic modelling of service non-functional features enables their common understanding by service consumers and service providers in open pervasive environments.

From the above discussion, it is evident that even if key enablers of pervasive computing such as service-orientation and semantic technologies have been the focus of intensive research, there are still major challenges for realizing the pervasive computing vision. These challenges can be addressed by an efficient, semantic, QoS-aware middleware for service-oriented pervasive computing that supports multi-network and multi-protocol interoperability.

1.2 Thesis Contribution and Document Structure

To address the above challenges, this thesis introduces a semantic, service-oriented middleware for pervasive computing. The most significant contributions of the proposed middleware are structured along this document as follows:

In Chapter 2, we present our vision of pervasive computing environments and analyse the challenges that underpin the realization of such vision. We further discuss the principles of service-oriented pervasive computing and survey related research efforts in the area of middleware for pervasive computing. From this analysis, we derive our motivation for a new semantic middleware for service-oriented pervasive computing. We further outline the architecture of the proposed middleware, which comprises of a set of functionalities developed in the next chapters of the thesis.

In Chapter 3, after a survey of existing semantic service description languages, we identify the requirement for a new semantic service model to support interoperability between these languages, which is at the heart of interoperability enabled by our middleware. The proposed model supports the specification of both semantic and syntactic service descriptions. For semantic-based service descriptions, our model further enhances the specification of semantic annotations where an additional source of heterogeneity has

been identified. This enables service providers and requesters to provide more accurate semantic specifications, which allows our middleware to perform more accurate semantic service matching. The formal specification of service conversations as finite state automata is supported by our model [Ben Mokhtar et al., 2005b]. This enables the automated reasoning about service behaviour independently from the underlying conversation specification language. Hence, pervasive service conversations described with different service conversation languages can be integrated towards the realization of a user task. Finally, our model supports the specification of service non-functional properties based on existing QoS models to meet the specific requirements of each pervasive application.

In Chapter 4, we present an efficient semantic service registry for pervasive computing environments [Ben Mokhtar et al., 2006c, Ben Mokhtar et al., 2006b]. The proposed registry supports a set of conformance relations for matching both syntactic and rich semantic service descriptions as well as their heterogeneous non-functional properties [Ben Mokhtar et al., 2007b]. As finding a service that exactly matches a client request is rather the exception than the rule in pervasive environments, our registry identifies the semantic distance between semantic service descriptions, and rates services with respect to their suitability for a specific client request, so that selection can be made among them. The evaluated semantic distance takes into account both functional and non-functional characteristics of services. Additionally, our registry supports the efficient reasoning on semantic service descriptions, which makes it applicable for highly interactive pervasive environments. Service descriptions in our registry are semantically organized to enable both efficient service publication and location. Thanks to the proposed optimizations we prove that our registry performs better than existing semantic service registries that opt for overloading the service publication phase to achieve efficiency at service location.

In Chapter 5, we present our service composition middleware functionality [Ben Mokhtar et al., 2005b, Ben Mokhtar et al., 2006a]. This functionality supports flexible QoS-aware service composition towards the realization of user-centric tasks abstractly described in the user's handheld [Ben Mokhtar et al., 2005c, Ben Mokhtar et al., 2005a, Ben Mokhtar et al., 2007a]. Flexibility is enabled by a set of composition algorithms that may be run according to the current resource constraints of the user's device. These algorithms further support

the assessment of the QoS requirements of user tasks by aggregating the QoS provided by the composed networked services. Unlike existing research efforts on service composition that assume complex behaviour for either services or tasks but not both, our proposed composition algorithms support the integration of services that have a complex behaviour to realize a user task also specified with a complex behaviour. This allows taking the full advantage of the diverse pervasive functionalities in the vicinity of a user at the specific time and place. Furthermore, we prove that our service composition is performed efficiently as it relies on our efficient semantic service registry to discover services and on efficient formal verification algorithms to build the user task realizations.

Chapter 6 presents a prototype implementation of our semantic service-oriented middleware complemented with the multi-network and multi-protocol interoperability methods coming from the MUSDAC middleware [Raverdy et al., 2006]. The overall prototype, which constitutes an innovative, efficient and comprehensive solution towards the realization of the pervasive computing vision, is evaluated in terms of the execution overhead of each of its constituent middleware functionalities presented in this thesis.

Chapter 7 summarizes our contributions presented in this thesis and discusses further research perspectives to be explored beyond this thesis.

Chapter 2

Middleware for Service-Oriented Pervasive Computing: Vision and State Of The Art

A computer is a machine able to store and process information according to a program. Computers had an incredible evolution in the last century going from room-size expensive calculators manipulated by experts in the 50's, to affordable personal computers in the 90's. Nowadays, computing facilities are embedded in thin devices, which start to vanish in our environments, in various forms (e.g., smart phones, embedded car systems, wearable computers, e-fridges). Nevertheless, the real advance enabled by such evolution does not come from any of these individual devices; it emerges from the interaction of all of them [Weiser, 1991]. Indeed, together with the evolution of computing, networking has also known a great evolution in the last fifty years. Thus, before the advent of computer networking that was initially performed over telecommunication networks, communication between computers was performed by humans, carrying information from one big calculator to another. Now, the current situation of computer networking is largely dominated by wireless networks enabling "communication on the move" [Zahariadis and Doshi, 2004]. The future of networking is further towards the so-called fourth-generation networking where all existing network technologies including wired and wireless ones are integrated

into a single *pervasive network*.

2.1 Pervasive Computing Environments

The convergence of powerful, small, affordable computing devices with a network that ties them all together, and software systems that seamlessly adapt to the surrounding environment leads to the vision of *pervasive computing*. The essence of this vision is that everywhere around us the environment is populated with computing and communication facilities gracefully integrated with human users [Satyanarayanan, 2001].

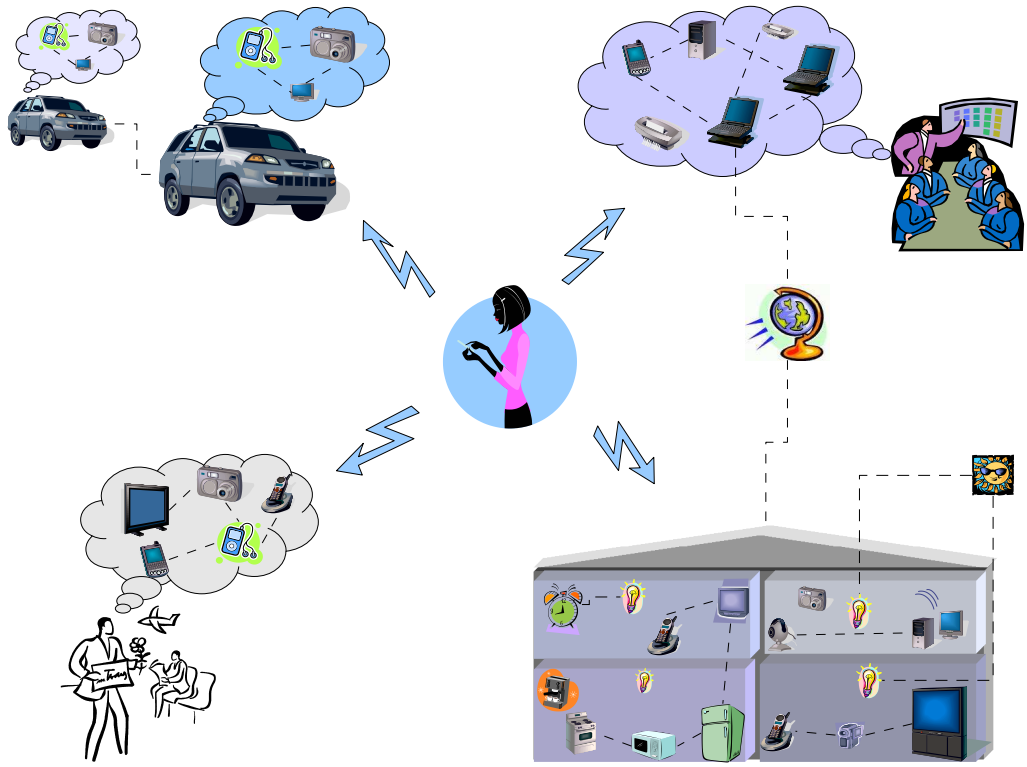


Figure 2.1: Pervasive Computing Environments

Figure 2.1 represents various pervasive computing environments (in, e.g., home, car, airport, office) populated with a number of networked devices called *pervasive devices* in the following (e.g., home appliances, computers, plasma screens, car GPS system). Such devices may range from resource rich (e.g., workstation in the home, plugged laptop in

the office) to more or less resource constrained devices (e.g., PDA or smart phone carried by the user in the airport, sensors in the home). These constraints may be in terms of CPU, memory, storage, display capabilities, battery power and bandwidth. Devices may further be stationary (e.g., a video server, a large screen) or mobile (e.g., a PDA, a car embedded system), and provide hardware and software functionalities to the pervasive environment, called *pervasive functionalities* (e.g., display functionality of a screen, video streaming functionality of a video server). Also, pervasive devices are permanently or punctually connected to the pervasive networking environment that may be constituted of heterogeneous networks including wired (e.g., WAN, LAN, ADSL Internet connexion) and/or wireless networks (e.g., PAN, Bluetooth, WiFi in ad hoc or infrastructure mode).

As depicted in Figure 2.1, a user carrying a pervasive device may move from one environment to another. Building *pervasive applications* realizing *user tasks* for mobile users by seamlessly combining the functionalities of pervasive devices and adapting the resulting combination to the specifics of each pervasive environment is one of the major challenges in achieving pervasive computing. This requires dealing with a number of challenges that are mainly due to:

1. The environment's **heterogeneity**: pervasive devices and the pervasive functionalities they provide are heterogeneous in terms of underlying technologies, and may reside in heterogeneous networks, which restricts the ability to integrate them for realizing user tasks.

2. The environment's **dynamics**: the mobility of some pervasive devices and the limited resources of others increase the dynamics of pervasive computing environments. This dynamics is perceived in terms of the number and lifetime of pervasive functionalities a user can access to at a specific time and location. In particular, new devices may appear in the environment while other devices may become out of reach due to a lack of resources (e.g., battery down), or due to the range of radio transmissions.

3. **Resource constraints** of thin devices: the limited resources of some devices that may participate in the realization of a user task have to be considered.

4. **User centrism**: in pervasive computing environments, the user is the center of attention. He/she must be served by the environment as seamlessly and as naturally as possible. User centrism calls for efficient solutions with acceptable response times enabling the interactivity with the user. It also requires the awareness on the surrounding environment, which includes the awareness of the non-functional characteristics of pervasive functionalities in order to enable the selection of functionalities that best conform to the user's needs.

Middleware, which is a software layer that stands between the networked operating system and applications and deals in a reusable way with problems like distribution, heterogeneity and mobility, frequently encountered in distributed systems [Issarny et al., 2007], appears as a major enabler for the development of pervasive applications. However, realizing user tasks for mobile users requires the middleware to deal with the previously identified challenges by providing:

1. Abstraction of the heterogeneous computing and networking environment for enabling the interoperation between pervasive devices independently from their underlying networking and computing technologies.

2. Abstraction of the pervasive functionalities provided by pervasive devices enabling the location of relevant pervasive functionalities available in the user's vicinity.

3. Middleware functionalities for enabling the dynamic publication, location and access of pervasive functionalities on the network and further the dynamic integration of these functionalities for realizing user tasks.

4. Middleware awareness of non-functional features of pervasive functionalities, which plays a decisive role in enhancing users' experience of the pervasive environment.

Among the various investigated middleware paradigms that distinguish by the coordination model they offer to applications, RPC-based middleware appears to be most appropriate for building user-centric tasks. Indeed, RPC allows invoking procedures on remote hosts enabling the user to be at the heart of the interactions with pervasive devices, while

other coordination models such as the tuple space and message-oriented models, which rely on indirect interactions through a shared memory or distributed message-queues respectively, can less naturally serve user centrism.

Among the various middleware paradigms that rely on the RPC coordination model, service-oriented middleware (SOM) is the one that best fits the requirements of pervasive computing. Indeed, compared to object-oriented (OO) and component-oriented (CO) middleware, SOM discussed in the next section, allows the development of pervasive applications in terms of loosely coupled pervasive services.

2.2 Service-Oriented Pervasive Computing

Service-Oriented Middleware (SOM) for pervasive computing is a middleware paradigm that employs the Service-Oriented Architecture (SOA) [M. P. Papazoglou, 2003] for modelling pervasive environments. Using SOA, pervasive functionalities provided by pervasive devices are abstracted as services. These services are independent software entities with well defined interfaces that may be accessed without any knowledge of their underlying technologies.

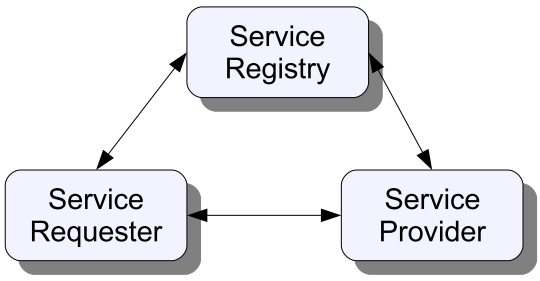


Figure 2.2: SOA Actors

The SOA architectural style is structured around the three basic actors depicted in Figure 2.2: *Service Provider* is the role assumed by a software entity offering a service, *Service Requester* is the role of a client entity seeking to consume a specific service, and *Service Registry* is the role of an entity maintaining information on available services and the way to access them. An additional role introduced as part of the extended SOA

[M. P. Papazoglou, 2003] is identified as *Service Aggregator*, which is the role of an entity that composes existing services and offers them as a new service to client applications. As depicted in Figure 2.3, service aggregator acts both as service provider by providing composite services to applications and as service requester by consuming existing networked services.

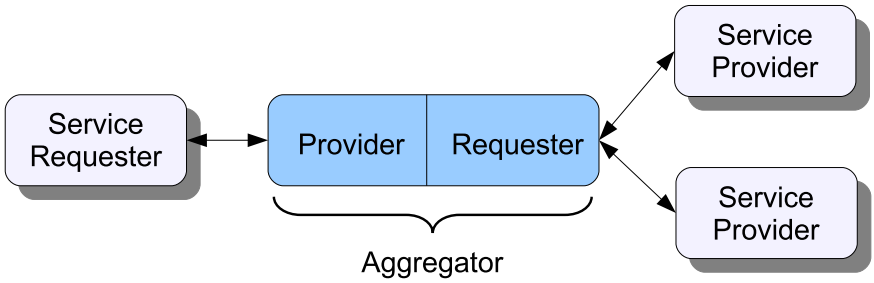


Figure 2.3: The Role of Service Aggregator

Figure 2.4, based on [M. P. Papazoglou, 2003], presents in more detail the notion of a service and illustrates the basic SOA functionalities performed by the SOA actors. For enabling the identification of provided and required services for providers and requesters, services are described in a structured way by using a *Service Description* formalism or language. There, the service *Capability*, *Interface*, *Behaviour*, *QoS* characteristics as well as the address for accessing the service may be specified. The service capability describes the functionality provided by the service, i.e., what the service does. The service interface describes the list of *Operations* by which the service realizes its capability. A service operation represents the unit of interaction with the service, it has a *Signature*, i.e., a structure in terms of data to be exchanged with the service. The service behaviour, called also *Conversation*, defines the temporal relationships and properties between the service operations necessary for a valid interaction with the service. Service QoS properties describe the non-functional characteristics of the service, such as security or transactional properties.

In Figure 2.4, SOA actors are associated with the basic SOA functionalities they perform. These functionalities are:

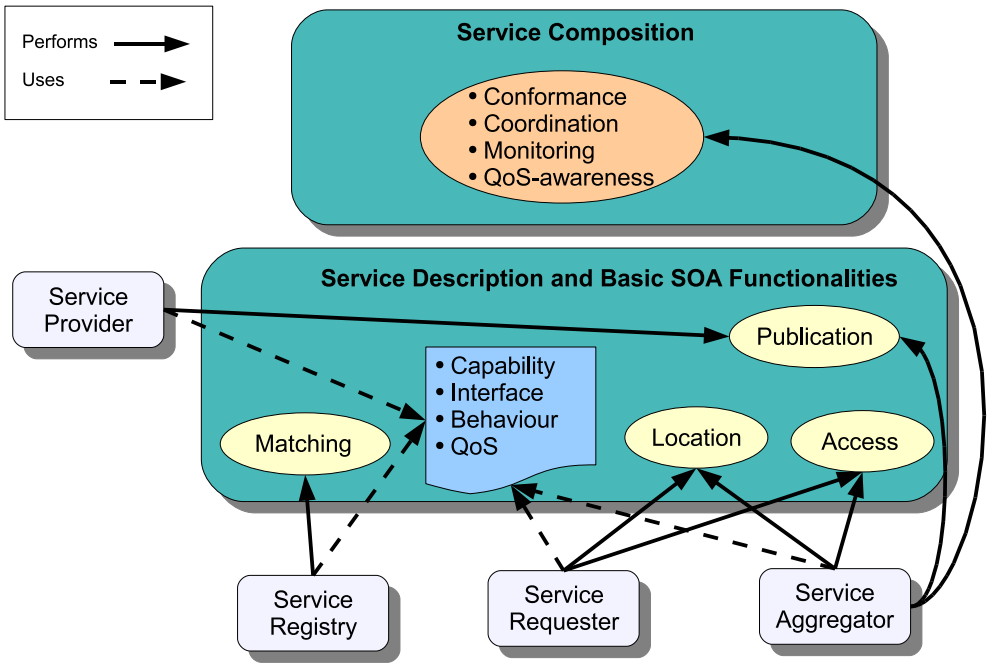


Figure 2.4: SOA Conceptual Elements

1. *Service Publication*: allows service providers to register their services in a service registry.

2. *Service Location*: allows service requesters to retrieve desired services from a service registry.

3. *Service Matching*: performed by a service registry, allows selecting among the registered services those that best conform to a service request.

4. *Service Access* allows a requester to establish a connection with a selected service, i.e., *Service Binding*, after which *Service Interaction* occurs as a set of successive invocations of service operations.

5. *Service Composition* allows the integration of multiple services into a single composite service, which may be achieved at design-time (static) or at run-time (dynamic). Service composition decomposes into four other sub-functionalities:

- *Service Conformance* ensures the integrity of a composite service by assessing the compatibility of its description with those of its constituent component services
- *Service Coordination* controls the execution of services taking part in a composition
- *Service Monitoring* allows observing the execution status of composite services to possibly trigger service adaptation
- *QoS-awareness* verifies the fulfilment of the QoS requirements of composite services based on the QoS provided by the integrated component services

Table 2.1 introduces the set of abstractions for designing pervasive computing systems using SOA. Specifically, pervasive devices that provide pervasive functionalities are abstracted as service providers that provide service capabilities. Users of the pervasive computing environment that request functionalities are viewed as service requesters. Advertising pervasive functionalities by pervasive devices and identifying relevant pervasive functionalities for the user correspond respectively to service publication and service location in the context of SOA. The realization of user tasks by composing pervasive functionalities of a pervasive computing environment translates to the dynamic composition of services. Finally, awareness of the surrounding environment to serve user-centrism includes the awareness of the QoS provided by pervasive services.

Based on SOA and more generally on the RPC coordination model, a number of middleware platforms for realizing the pervasive computing vision have been proposed in the literature as surveyed in the next section.

2.3 Middleware for Pervasive Computing Environments: State of The Art

We survey in this section research efforts investigating middleware platforms for pervasive computing. Specifically, proprietary middleware that rely on specific middleware technologies and focus on issues related with the dynamic composition of pervasive applications

Pervasive Computing	SOA
Pervasive Device	Service Provider
Pervasive Functionality	Service Capability
Pervasive Application Realizing a User Task	Composite Service
User	Service Requester
Advertising pervasive functionalities	Service Publication
Identifying relevant pervasive functionalities	Service Location
User Task Realization	Dynamic Service Composition

Table 2.1: Service-Oriented Pervasive Environments

are presented in Section 2.3.1. Then, interoperable middleware that provide solutions for dealing with middleware heterogeneity in pervasive computing environments are surveyed in Section 2.3.2. Finally, we discuss existing research efforts investigating semantic-aware middleware in Section 2.3.3 and provide an overall discussion in Section 2.3.4.

2.3.1 Proprietary Middleware

There have been a number of research projects investigating middleware platforms for pervasive computing. We survey in the following the Aura, Gaia, Oxygen, Pico and WSAMI projects, which are all focusing on enabling the dynamic composition of pervasive service capabilities.

The Aura project [Sousa and Garlan, 2002] defines an architecture that allows users to dynamically realize daily tasks modelled as abstract software applications, in a transparent way, without manually dealing with the configuration and reconfiguration issues of these applications. User tasks defined in Aura are composed of abstract services to be found in the environment. One of the main innovative features of Aura is that user tasks adapt themselves according to the resources available in each pervasive computing environment, thus taking the full advantage of the diverse capabilities of each environment. Furthermore, each environment is able to renegotiate task support with respect to the run time variation

of service capabilities and resources. The main issues addressed by the Aura system are thus related with the management of the environment's dynamics through the dynamic configuration and reconfiguration of user tasks.

The Gaia project [Roman et al., 2002, Shiva Chetan and Campbell, 2005] is a distributed middleware infrastructure that coordinates networked devices and software components in a physical space, called an active space, in order to enable the dynamic deployment and execution of software applications. In this middleware, an application is mapped to available resources of a specific active space. This mapping can be either assisted by the user or automatic. Gaia supports the dynamic reconfiguration of pervasive applications. For instance, it allows changing the composition of an application dynamically upon a user request (e.g., the user may specify a new device providing a component that should replace a component currently used). Furthermore, Gaia supports the mobility of applications between active spaces by saving the state of the application. Similarly to Aura, Gaia focuses on the dynamic aspect of pervasive environments and provides the support for dynamically mapping applications to available resources of a specific active space.

Pico (Pervasive Information Community Organization) [Kumar et al., 2003], is a middleware framework intended for time-critical applications (e.g., tele-medicine, military applications). This middleware supports the automated, continual, unobtrusive provision of services. It consists of autonomous software entities called *delegents* (or intelligent delegates) and hardware devices that provide services called *camileuns* (which stands for **c**onnected, **a**daptive, **m**obile, **i**ntelligent, **l**earned, **e**fficient, **u**biquitous **n**ode**s**). The main objective of Pico is to allow the dynamic creation of delegent communities in order to perform tasks on behalf of users. While Pico relies on different paradigms compared to Aura and Gaia (e.g., software agents), it tackles similar issues, related with the adaptation to the dynamically changing pervasive environments through the dynamic composition of pervasive service capabilities.

The WSAMI project [Issarny et al., 2005] supports the abstract specification of pervasive computing applications in the form of software architectures, together with their dynamic composition according to the environment. The proposed middleware builds on

the Web services architecture[1], whose pervasiveness enables service availability in most environments. In addition, dynamic composition of applications is dealt with in a way that enforces quality of service for deployed applications in terms of security and performance through the systematic customization of connectors that dynamically integrate relevant middleware-related services. One of the major benefits of WSAMI compared to the above middleware infrastructures is its ability to be deployed on resource-constrained devices, which enables infrastructure less, totally decentralized pervasive computing environments.

Oxygen [Walker, 2004] is an MIT project that aims at enabling pervasive, human-centred computing through a combination of system, software and networking technologies developed for the purpose of the project. In Oxygen, users naturally interact with the system using speech and vision technologies. This system relies on a variety of computational and handheld devices. Specifically, computational devices called Enviro21s (E21s) are devices embedded in home, office, and car environments and are responsible of sensing these environments. On the other hand, handheld devices, called Handy21s (H21s), are carried by users and allow them to interact with the environment and perform daily tasks. Furthermore, the project rely on specific networking and software technologies, i.e., the self-configuring networks (N21s), and the self-adaptive software (O2S). The oxygen project developed technology-specific software, hardware and networking technologies, through which rich pervasive computing environments have been built. Some of these prototyped technologies are being tested by the Oxygen industry partners. While this project developed advanced, highly adaptive, user-centric applications, their assumptions on the availability of specific technologies such as H21 devices with specific interfaces that communicate over specific networks is restrictive. The current situation of pervasive computing is much more heterogeneous than that, and we can hardly envision that H21 devices and associated software and networking technologies will be available worldwide in the near future.

While the above middleware infrastructures deal with a lot of issues related with the environment dynamics (e.g., dynamic configuration, reconfiguration of pervasive software systems), they poorly deal with the environment heterogeneity. Indeed, existing middle-

[1]Web Services: http://www.w3.org/2002/ws/

ware generally employ or specify reference protocols to discover and communicate with networked software services, thus enabling compliant software systems to interoperate. The ultimate objective of these approaches is to introduce a reference middleware for pervasive computing to be deployed everywhere. However, the emergence of such middleware platforms that have been put forward towards the realization of the pervasive computing vision including those surveyed above, has generated a new problem: *middleware heterogeneity*. Indeed, two software applications that rely on two different middleware infrastructures are unable to communicate with each other, thus calling for interoperable middleware.

2.3.2 Interoperable Middleware

In the context of service-oriented pervasive computing, middleware heterogeneity manifests itself in the two major middleware functionalities, i.e., service discovery, which includes service publication, location and matching; and service access.

To deal with middleware heterogeneity, a number of middleware platforms have investigated interoperability methods [Grace et al., 2003, Bromberg and Issarny, 2005, Raverdy et al., 2006].

ReMMoC (Reflective Middleware for Mobile Computing) [Grace et al., 2003], is a configurable and reconfigurable middleware that enables software applications to be developed independently of specific middleware technologies. Such applications are then able to discover and interoperate with a range of heterogeneous services, thanks to the ReMMoC-awareness of the middleware technologies available in the current environment. Specifically, upon the detection of the specific service discovery and access protocols employed in the current environment, ReMMoC reconfigures by loading the appropriate *component frameworks* enabling service requesters to use those protocols. This is enabled through a single common interface provided for all the supported underlying protocols.

INDISS (Interoperable Discovery System for Networked Services) [Bromberg and Issarny, 2005] introduces a transparent approach to service discovery protocol interoperability. Specifically, the interoperability layer is located on top of the network layer and directly translates protocol messages to/from the various service discovery protocols. Contrary to ReMMoC

that requires service requesters to support multiple protocols and realizes interoperability through protocol substitution, INDISS realizes interoperability transparently to service requesters and providers through protocol translation. Those continue to use their native middleware and related discovery and access protocols. Similarly to INDISS, the same authors present NEMESYS (Network Meta communication System for Middleware Interoperability) [Bromberg, 2006], which realizes interoperability between service access protocols.

Another solution to interoperability proposed in the MUSDAC (MUlti-protocol Service Discovery and ACcess) middleware [Raverdy et al., 2006], enables explicit translation of service discovery protocol messages. In this middleware, the interoperability layer is located on top of the existing service discovery protocols, and provides an explicit discovery API to service requesters. Then, incoming service service advertisements are translated to a common XML format proprietary to the MUSDAC platform. This enables MUSDAC to match service requests against service advertisements independently from their initial service description format. Further to dealing with service discovery protocols heterogeneity, MUSDAC enables service discovery over multi-network environments, which is a key requirement in pervasive computing environments.

While the above three solutions deal with middleware heterogeneity, they suffer from a common limitation, which also concerns the proprietary middleware platforms. Indeed, the openness of pervasive computing environments requires that service requesters and providers agree on both the functional and non-functional semantics of service capabilities, so that they can integrate and interact in a way that guarantees dependable service provisioning and consumption. In all the above surveyed approaches, this agreement is performed at the syntactic level, assuming that service requesters and providers use a common syntax for denoting service semantics. However, such vision, based on the strong assumption that service providers and requesters describe services with identical terms worldwide, is hardly achievable in open pervasive environments. This raises the issue of *syntactic heterogeneity* of service descriptions. A promising approach towards addressing syntactic heterogeneity relies on semantic modelling of the services' functional and non-functional features.

2.3.3 Semantic-aware Middleware

A field of research from the artificial-intelligence domain that deals with the definition of the semantics of information is called *knowledge representation*. From this field, an appropriate model to represent knowledge is *ontologies* [Singh and Huhns, 2005], which constitute a rich model for formally specifying information and a variety of structural and non-structural relationships between information. Specifically, an ontology is a formal explicit description of terms in a domain of discourse (*classes*, sometimes called *concepts*), properties of each concept describing various features and attributes of the concept (*slots*, sometimes called *roles* or *properties*), and *restrictions* on slots (*facets* (sometimes called *role restrictions*)). Most ontology models support the following relationships [Singh and Huhns, 2005]:

1. *Inheritance*, called also *subsumption* relation (*is-a*, *is-subtype-of* or *is-subclass-of*, the converse of *is-superclass-of*), is the relation between a class and one or more refined versions of it. Each subclass shares the same features of its superclass, adding its own features to it. Inheritance allows concepts to be organized in hierarchies.

2. *Aggregation* called also *Meronymy* relation (part-whole or part-of) defines how classes representing components of something are related with a class defining the entire assembly.

3. *Instantiation* is the relation that associates classes with concrete "real-life" objects, called *individuals* or *instances*.

One of the most widely used languages for specifying ontologies, which is a W3C recommendation, is the Web Ontology Language (OWL[2]). OWL has its formal foundation in Description Logics (DL) [Donini et al., 1996]; hence, the semantic specification of information using ontologies in OWL enables semantic reasoning on this information.

Semantic reasoning performed by a DL-reasoner allows inferring implicit relationships between concepts from the explicit definitions of these concepts in an ontology, which

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

is called *ontology classification*. Figure 2.5, produced with the Protégé[3] ontology editor, shows an example of an ontology classification using a DL-reasoner. The left part of the figure shows the specification of a Pizza ontology[4] (asserted hierarchy of concepts), while the right part shows the same ontology after classification (inferred hierarchy of concepts). In the figure, some concepts defined under the concept NamedPizza in the ontology, e.g., American pizza, have been classified under the concept CheesyPizza after reasoning carried out based on the description of their ingredients.

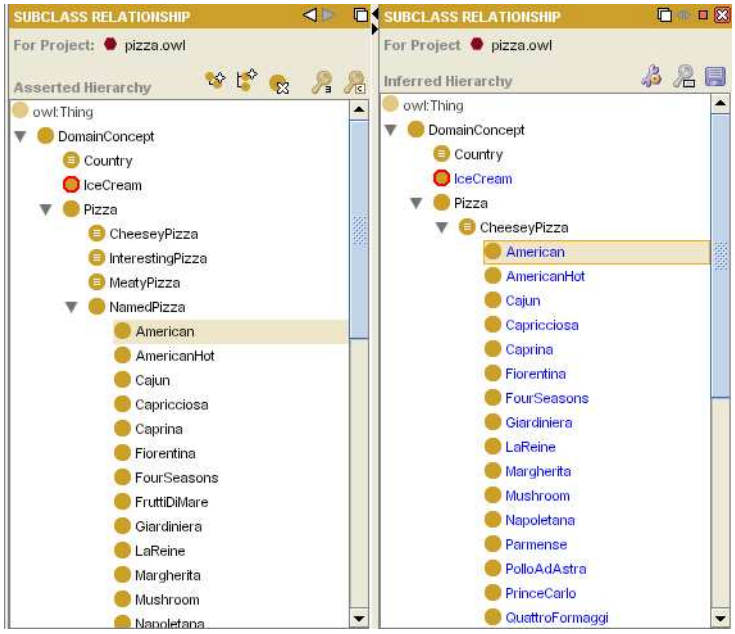


Figure 2.5: Asserted (left) and Inferred (right) Hierarchies of Concepts

Using ontologies, various elements of service descriptions can be formally specified, leading to a consistent interpretation of the information exchanged between different participants in the service-oriented pervasive computing environment. A number of research efforts have thus investigated middleware platforms that support semantic specification of services for pervasive computing [Masuoka et al., 2003, Singh et al., 2005, Chakraborty et al., 2006, Chakraborty et al., 2005]. These solutions mainly focus on providing middleware function-

[3]Protégé: http://protege.stanford.edu/
[4]Pizza Ontology: http://www.co-ode.org/ontologies/pizza/

alities enabling semantic service discovery and composition as surveyed hereafter.

The Task Computing project [Masuoka et al., 2003] is an effort for ontology-based dynamic service composition in pervasive computing environments. It relies on an existing service discovery protocol, i.e., UPnP (Universal Plug and Play[5]), enriched with semantic service descriptions given in OWL-S[6] (Ontology Web Language for Services). Each user of the pervasive computing environment carries a service composition tool on his/her device that discovers on the fly available services in the user's vicinity and suggests to the user a set of possible compositions of these services. While this approach validates the relevance of ontology languages in pervasive computing environments, it presents some limitations. For instance, suggesting to the user all the possible compositions of networked services requires that the user selects the right composition among the suggested ones, which can be inconvenient for mobile users of the pervasive computing environment, particularly, if the number of possible compositions is high. Finally, the discovery protocol employed in this approach, which has been designed for the networked home environment is not well suited for highly dynamic, large scale environments [Flores-Cortes et al., 2006].

IGPF (Integrated Global Pervasive Computing Framework) [Singh et al., 2005] introduces a semantic Web services-based middleware for pervasive computing. This middleware builds on top of the semantic Web paradigm [Berners-Lee et al., 2001] to share knowledge between the heterogeneous devices that populate pervasive computing environments. The idea behind this framework is that information about the pervasive computing environments (i.e., context information) is stored in knowledge bases on the Web. This allows different pervasive computing environments to be semantically connected and to seamlessly pass user information (e.g., files/contact information), which allows users to receive relevant services. Based on this knowledge bases, the middleware supports the dynamic composition of pervasive computing services modelled as Web services. These composite services are then shared across various pervasive computing environments via the Web. This solution suffers from the strong assumption that pervasive devices have a permanent connection to the Internet, which may not be always be the case (e.g., sponta-

[5]UPnP: http://www.upnp.org
[6]OWL-S: http://www.daml.org/services/owl-s/

neous, infrastructure less connections between mobile users).

The Ebiquity group describes a semantic service discovery and composition protocol for pervasive computing. The service discovery protocol called GSD (Group-based Service Discovery) [Chakraborty et al., 2006], groups service advertisements using an ontology of service functionalities. In this protocol, service advertisements are broadcasted to the network and cached by the networked nodes. Then, service discovery requests are selectively forwarded to some nodes of the network using group information propagated with service advertisements. Based on the GSD service discovery protocol, the authors define a service composition functionality for infrastructure-less mobile environments [Chakraborty et al., 2005]. Composition requests are sent to one of the composition managers of the environment which performs a distributed discovery of the required component services.

The above three semantic-aware middleware for pervasive computing provide base support for the semantic discovery and composition of pervasive services. However, the efficiency of the proposed solutions with respect to the resource constraints of pervasive devices is not assessed. Indeed, semantic-awareness realized through semantic reasoning on ontologies is a resource consuming process, which is not suitable to be employed in resource constrained devices without appropriate optimizations [Ben Mokhtar et al., 2006b].

2.3.4 Discussion

Due to the specifics of pervasive computing environments, the development of pervasive applications realizing user tasks for mobile users by seamlessly integrating pervasive functionalities provided by pervasive devices raises a number of middleware requirements identified in Section 2.1. After the survey of existing research efforts in the area of middleware for pervasive computing these requirements can be refined as follows:

- The support of middleware *interoperability* to enable pervasive users to discover and interact with pervasive services independently from the underlying technologies. Middleware interoperability decomposes into *service discovery protocol* interoperability and *service access protocol* interoperability.

- The support of *multi-network* management to enable pervasive users to reach pervasive services available on heterogeneous networks
- The support of *semantic-awareness* to enable the consistent interpretation of the information advertised about pervasive services and requested by users
- The support of the *dynamic publication, location, access* and *composition* of pervasive services
- Efficiency of the provided middleware functionalities should be assessed to fit the resource constraints of thin devices
- The support of QoS-awareness to enable the realization of user-centric tasks

While the essence of the above requirements is well understood, and solutions to individual requirements have been proposed that may form the foundations of a comprehensive middleware for pervasive environments, a number of problems remain. First and foremost, these issues have always been considered separately. For instance, to the best of our knowledge, middleware interoperability solutions have only addressed syntactic service discovery. At the same time, semantic-aware middleware neither manage discovery and access protocol-heterogeneity nor network-heterogeneity. Furthermore, efficiency issues, specifically those due to semantic-awareness, are poorly investigated. In the following, we outline the architecture of a semantic middleware for pervasive computing that comprehensively deals with all the above issues.

2.4 Semantic, Service-Oriented Middleware for Pervasive Computing

We sketch in this section the architecture of a semantic middleware for pervasive computing (Figure 2.6). This middleware provides pervasive applications (highest layer in the figure) with a set of semantic SOM functionalities (upper middle layer in the figure) that realize the basic SOA functionalities meeting the requirements of pervasive computing environments. These functionalities are:

1. *Semantic-, QoS-aware Service Composition*: integrates multiple semantic services into a single composite service that realizes or participates in the realization of a user task. Service composition considers both functional and QoS capabilities of the composed services to satisfy users' requirements. *Service Registry*: stores service descriptions given in a common language to which heterogeneous service descriptions are translated by the multi-protocol management functionality described below. This registry can be centralized, semi-distributed or fully distributed according to the deployment policy of the middleware. For instance, a semi-distributed deployment of our proposed middleware is described in Chapter 6. Our proposed service registry further supports the efficient semantic, QoS-aware service publication, location and matching as follows:

    - *Semantic-, QoS-aware Service Publication*: allows service providers to publish semantic-enhanced service descriptions covering both service functional and QoS properties.
    - *Semantic-, QoS-aware Service Discovery*: allows service requesters to retrieve semantic-enhanced services by specifying functional and QoS requirements.
    - *Semantic-, QoS-aware Service Matching*: allows selecting among the registered services those that best conform to a service request in terms of semantically specified service functional and QoS properties.

The semantic SOM layer is built on top of the middleware communication layer (lower middle layer in the figure), which provides two essential functionalities for dealing with middleware and network heterogeneity, i.e., the multi-protocol and multi-network management. Multi-network management enables the dissemination of service discovery and access requests in the whole environment despite the heterogeneity of the underlying networks. Multi-protocol management includes service discovery protocol interoperability and service access protocol interoperability. It allows a service requester that relies on a specific service discovery and access protocol to discover and interact with services that rely on different discovery and access protocols. Both service discovery and access protocol interoperability rely on the translation of service discovery and access messages (e.g.,

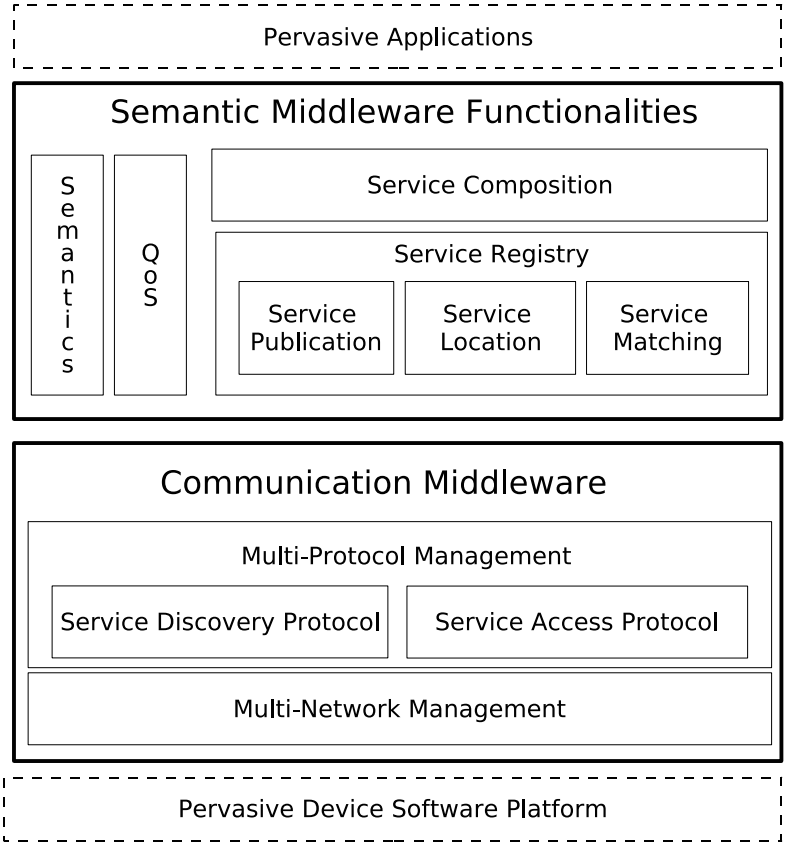


Figure 2.6: Semantic Service-Oriented Middleware for Pervasive Computing

discovery and access requests) from one protocol-specific format to another. Additionally, service discovery protocol interoperability requires the translation of service advertisements into a common service description language for enabling service matching and composition to be performed independently from the specific underlying languages. The resulting homogeneous service descriptions are stored by the service registry of our middleware.

Finally, the pervasive device software platform (lowest layer in the figure) integrates the networked operating system, device drivers and software libraries providing base system and network functionalities on which the middleware executes.

In the following chapters of this thesis we detail the different middleware functionalities that constitute our proposed middleware. Specifically, we describe in Chapter 3 a model for the semantic specification of pervasive services to be supported by our proposed

semantic service registry. We then present in Chapter 4 our efficient semantic service registry for pervasive computing environments and its provided SOM functionalities, i.e., efficient, QoS-aware, semantic service publication, location and matching. In Chapter 5, we present our QoS-aware service composition middleware functionality for the dynamic realization of user tasks. We finally present in Chapter 6 a prototype implementation and evaluation of our middleware by integrating the multi-protocol and multi-network middleware functionalities coming from the MUSDAC platform [Raverdy et al., 2006] with the previously presented SOM functionalities.

Chapter 3

Semantic Specification of Pervasive Services

The semantic specification of pervasive services is at the heart of a semantic SOM. It enables service providers to describe their capabilities and service requesters to formulate their requests. Specifically, all the SOM functionalities of our middleware rely on service descriptions. Indeed, service publication (resp. service location) uses a service description to advertise (resp. discover) a set of capabilities provided (resp. required) by a pervasive service (resp. user task). Service matching compares service descriptions provided by pervasive services and those required by user tasks. Finally, service composition integrates capabilities identified in service descriptions to realize user tasks.
According to the SOA architectural style presented in Section 2.2 (p. 13), service description should enable the specification of :

1. Provided capabilities of pervasive services and required capabilities of user tasks. The semantics underlying service capabilities should be specified using references to existing ontologies.

2. Conversations of pervasive services and user tasks for modelling their behaviour. The specification of service conversations should rely on a formal model in order to enable the automated reasoning on service behaviour. Such reasoning enables the

valid integration of services having a complex behaviour for the realization of a user task also described with a complex behaviour.

3. Non-functional properties of pervasive services and user tasks (i.e., QoS information). This specification should be extensible to support the definition of domain-specific non-functional requirements.

4. Binding information of pervasive services necessary for the service invocation, such as the service access protocol, message formats, serialization, transport and addressing information.

A number of languages have been proposed in the literature to support the semantic specification of services (e.g., OWL-S, WSMF, SWSF and SAWSDL). These languages support the requirements for semantic service specification discussed above. However, to enable multi-protocol interoperability as envisioned by our SOM, a service registry should support the publication of services described using different service description languages as well as multi-language service matching to answer heterogeneous service requests. This can be done by translating all the service descriptions at service publication (resp. location) time, into a common language, which can be one of the existing languages, so that service matching can be performed independently from the underlying service description languages. Towards this purpose, a conceptual model that homogenizes the different terminologies, raises ambiguity between contradicting elements of the different languages and provides the formal ground to perform semantic service matching is required.

After a survey of existing semantic service description languages in Section 3.1, we present in this chapter a conceptual model for the semantic specification of pervasive services in Section 3.2 and a formalization of this model in Section 3.3. The instantiation of this model through a combination of the languages BPEL4WS and SAWSDL is presented in Chapter 6.

3.1 Semantic Service Description Languages: State Of The Art

OWL-S

OWL-S (previously named DAML-S)[1], is an ontology defined using the Ontology Web Language (OWL)[2] to describe Web services capabilities. Using OWL-S, a service description is composed of three parts: the *service profile*, the *process model* and the *service grounding*. The service profile gives a high level description of a service and its provider and is generally used for service publication and discovery. It includes: (1) An informal description of the service oriented to a human user; (2) A description of the service's capabilities, in terms of Inputs, Outputs, Pre-conditions and Effects (IOPE); and (3) An extensible set of attributes describing complementary information about the service, like the service type, category, etc. The *process model* describes the service conversation as a process, while the *service grounding* specifies the information necessary for service invocation. There have been efforts for formally specifying OWL-S conversations using process algebra [Narayanan and McIlraith, 2002].

WSMF

The Web Service Modeling Framework (WSMF)[3] consists of four main elements: *Ontologies, Goals, Web services, Mediators*. Ontologies in WSMF are defined using the Web Service Modeling Language (WSML). They are used for defining the semantics underlying service descriptions. Goals describe the objectives that a service requester has, which are fulfilled through the execution of Web services. A goal contains the description of a required capability as well as a required interface. A Web service is a computational entity able to achieve users goals. It is described using a provided capability, a provided interface and non-functional properties. Capabilities are defined with a set of preconditions, assumptions, postconditions and effects. A service interface describes how a service capability can be achieved. This can be described either in terms of a set of interactions

[1]OWL-S: http://www.daml.org/services/owl-s/
[2]OWL: http://www.w3.org/TR/owl-features/
[3]WSMF: http://www.wsmo.org/

with the service, i.e., service conversation, or as an orchestration of other Web services. Finally, mediators are used to resolve mismatches between the other three elements (i.e., ontologies, goals and services). Service conversations in WSMF have also been associated with a formal semantics in [Wang et al., 2007].

SWSF

The *Semantic Web Services Framework* is an effort by the Semantic Web Services Initiative[4]. It is composed of two parts: the *Semantic Web Services Language* (SWSL) and the *Semantic Web Services Ontology* (SWSO). While, SWSL is a language for ontology specification, SWSO is the conceptual model by which services can be described (i.e., the service ontology itself). There are two versions of SWSO, the *First-Order Logic Ontology for Web Services* (FLOWS) and the *Rules Ontology for Web Services* (ROWS). Similarly to OWL-S, FLOWS (as ROWS) is composed of three ontologies: the Service Descriptors, the Process Model and the Grounding ontology. Service descriptors provide basic information about Web service capabilities and properties. The FLOWS process model ontology is an extension of the *Process Specification Language* (PSL ISO 18629) with concepts from the Web services domain to specify Web service conversations. FLOWS is associated with the Web Service Description Language (WSDL[5]) for providing Web service binding information.

SAWSDL

The Semantic Annotation for WSDL and XML Schema (SAWSDL)[6] is a W3C candidate recommendation. SAWSDL defines how to add semantic annotations to various parts of a WSDL document and its associated XML schema files. These annotations are defined by means of three SAWSDL attributes: the *modelReference* is used to link WSDL elements (e.g., operation names, input, output messages) or XML Schema elements representing data types, with concepts in existing ontologies while the *liftingSchemaMapping* and *low-*

[4]SWSF: http://www.swsi.org/
[5]WSDL: http://www.w3.org/TR/wsdl
[6]SAWSDL: http://www.w3.org/2002/ws/sawsdl/

eringSchemaMapping attributes are used to specify mappings between semantic data and XML structures. This mapping is necessary when, for instance, a service requester that relies on an XML message structure wants to invoke a service that semantically matches its requirements but relies on a different XML message structure. SAWSDL does not support the specification of service conversations. Nevertheless, it is usually combined with WS-BPEL[7], an OASIS candidate standard for the specification of business processes, for describing service conversations. Formal specification of WS-BPEL conversations has been defined using various formalisms (e.g., Finite state automata [Wombacher et al., 2004], process algebra[Cámara et al., 2006]).

Discussion

The four languages described above are compliant with our requirements for the semantic specification of pervasive services. Indeed, all of them support the semantic specification of a *service* as an entity providing a number of *capabilities*. A capability is described with a set of *inputs, outputs* and possibly *pre-conditions* and *post-conditions*. The semantics of each of these elements is defined with references to existing ontologies. Furthermore, a capability is given a set of *non-functional properties* (e.g., QoS properties) and a *conversation*. All of these languages describe a basic set of non-functional properties and support extensions for the definition of application-specific attributes. Finally, conversation specifications are given a formal semantics yet relying on different formal languages. However, the emergence of such service description languages that have been defined to address the syntactic heterogeneity of service descriptions contribute to the middleware heterogeneity problem discussed in Chapter 2. Indeed, these languages employ different terminologies to design similar service elements (Table 3.1 describes the terminology employed by each language for specifying SOA concepts introduced in Section 2.2, p. 13).

Furthermore, they rely on different formalisms to define the semantics of services conversations, which restricts the ability of integrating them towards the realization of user tasks. Finally, the way semantic annotation of service elements is performed differs from

[7]WS-BPEL: www.oasis-open.org/committees/wsbpel/

	OWL-S	WSMO	FLOWS	SAWSDL
Capability	Service Profile	Capability	Service Descriptor	Operation
Behaviour	Process Model	Interface	Process Model	Not supported
Input	Input	Assumption	Input	Input
Output	Output	Effect	Output	Output
Non-functional property	Service Parameter	non-functional property	Properties from OWL-S and WSMO	Not supported

Table 3.1: SOA Concepts Supported by Semantic Service Specification Languages

one approach to another. Specifically, when a service element (e.g., output message) is annotated with an ontology concept this may be interpreted as:

- The service provides an output of type the concept itself or *any* of its sub-concepts in the ontology hierarchy (e.g., a car selling service may provide *any* type of car with respect to the ontology hierarchy) or

- The service provides an output of type the concept itself or *some* of its sub-concepts in the ontology hierarchy (e.g., a car selling service may provide some type of cars)

Existing approaches for semantic service specification and matching assume one of the two types of annotation. For instance, Paolucci *et al.* in [Paolucci et al., 2002] supports the first type, while [M'Bareck and Tata, 2007] supports the second type of annotation.

This introduces ambiguity in service descriptions that a SOM for pervasive computing has to deal with.

Considering all these types of heterogeneity, a service requester that relies on a service description language is not able to discover and interact with service providers that use another service description language. A SOM for pervasive computing should deal with such heterogeneity to allow a service requester to discover and interact with a service provider even if these two actors are using different service description languages. Furthermore,

existing syntactic-based middleware already available in pervasive environments should also be supported by a SOM to achieve full interoperation.

Interoperability can be achieved by translating heterogeneous service advertisements into a common semantic service description language. This allows performing service matching and composition independently from the underlying service description languages. This calls for a conceptual model, which we present in this chapter, that homogenizes the terminologies between the heterogeneous service description languages, raises ambiguity underlying semantic annotation by enabling the explicit specification of annotation types and provides the formal ground for enabling the definition of service matching relations.

3.2 Semantic Service Specification Model

The UML diagram depicted in Figure 3.1 shows a graphical representation of our semantic service model. This diagram introduces the various elements of our model and the key relationships between these elements.

3.2.1 Provided and Required Capabilities

At the heart of our model is the concept of *Service*, which is defined as an aggregation of a non-empty set of independent *Capabilities* and a possibly empty set of *Non-Functional Properties*. A capability is defined as an aggregation of a potentially empty set of *Inputs*, a non-empty set of *Outputs*, an optional *Category*, an *Automaton* for describing its *Conversation* and a possibly empty set of properties. Capabilities having a conversation that involve other capabilities are said to be *composite capabilities*, whereas the others are said to be *elementary capabilities*. Elementary capabilities correspond to the unit of interaction with the service, i.e., service operations. The inputs and outputs of a service capability describe respectively the information necessary for the execution of the capability and the information resulting from the execution of the capability. A capability is also characterized by its category, which is a description of the functionality provided by the capability. Each input, output and category of a capability is defined with a *Name*,

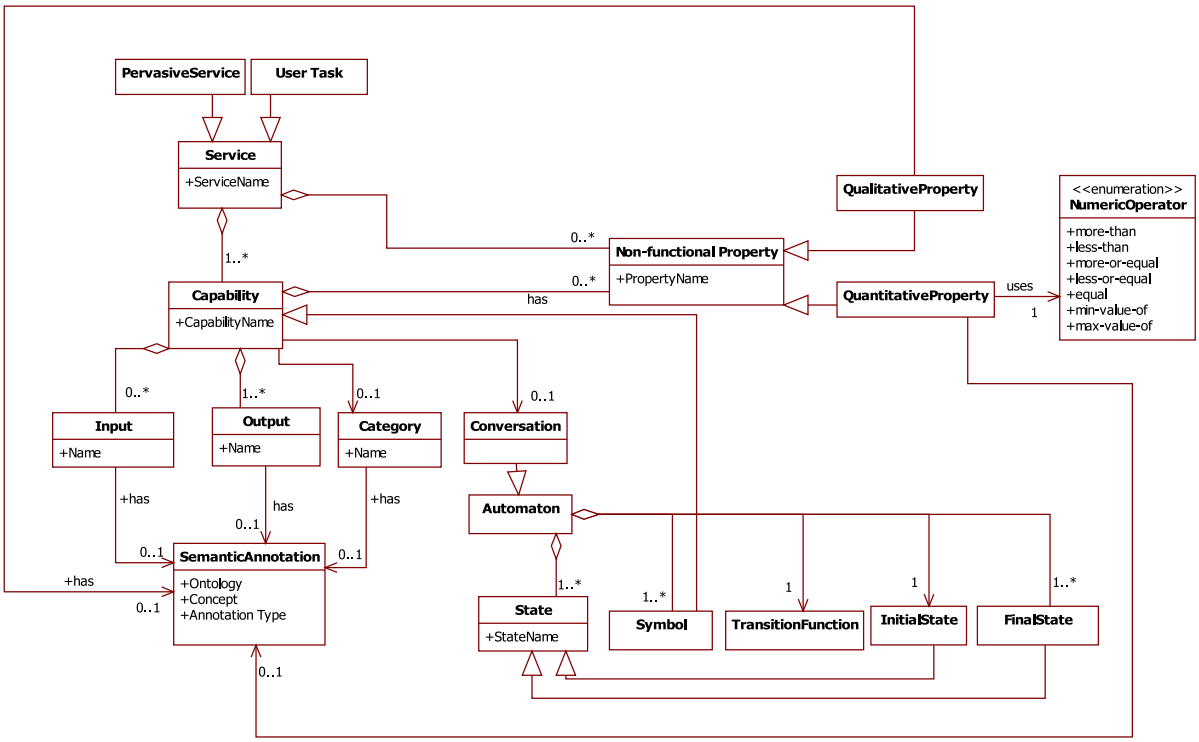


Figure 3.1: Semantic Service Specification Model

and a possible *Semantic Annotation* which is used to express the semantics underlying the input, output or the category to which it is associated. The semantic annotation of service elements is optional in order to support both semantic and syntactic-based languages. A Semantic annotation is a reference to an existing ontology concept. It is defined with the *Name* of the concept, a reference to the *Ontology* in which the concept is defined, and the *Annotation type* associated with the semantic annotation. The annotation type is used to specify what is intended by the employment of a semantic concept when annotating an entity. Two annotation types associated with a concept are supported in our model: *all-values-from* and *some-values-from* noted $\otimes$ and $\oplus$ respectively. For instance, if a provided capability has an output information annotated with a concept associated with the all-values-from annotation type, this means that the latter output can have as type the concept itself or *any* of its sub-concepts. On the contrary, if an output is annotated with a

concept having a some-values-from semantics, this means that it can have as type the concept itself or *some* of its sub-concepts. In contrast with existing approaches that assume one of the two annotation types, our model supports both types in order to allow service providers and service requesters to give more accurate annotations of their provided and required capabilities.

Figure 3.2 shows an example of a required and a provided capability inspired from the scenario introduced in Chapter 1. Both the required and the provided capabilities use (the classified fragment of) the *Resource Ontology* depicted in Figure 3.3. The provided capability, named *Airport Entertainment Server*, is offered by the airport infrastructure. It allows travellers who are waiting for their flight to listen to music or watch short movies on their mobile devices. This capability takes as input a *ResourceName* and provides users with corresponding video and sound resources. Hence, this capability has two outputs that are annotated with the concepts *Video Resource* and *Sound Resource* respectively. Semantic annotation is represented in the figure with the notation *Ontology#Concept-Annotation Type*. Both these outputs are associated with the *all-values-from* annotation type because the server does not have have any restriction regarding file types. For instance, for music resources, it can provide either mp3, ogg or midi files according to the user's preferences. Moreover, this capability is of category *Digital Server* associated with the *some-values-from* annotation type as it can act as a *Music* or a *Video server* but not as a *Game Server*. On the other hand, the required capability, that may reside in the handheld of a mobile user has the same input as the provided capability and an output of type *EntertainmentResource* associated with the type *some-values-from*. This means that the user looks for any type of resource among video, game and sound that corresponds to the topic he/she gives as input. The category of this capability, i.e., *Entertainment Server* is also given a *some-values-from* annotation type, which means that any server among video, music and game server would satisfy the request.

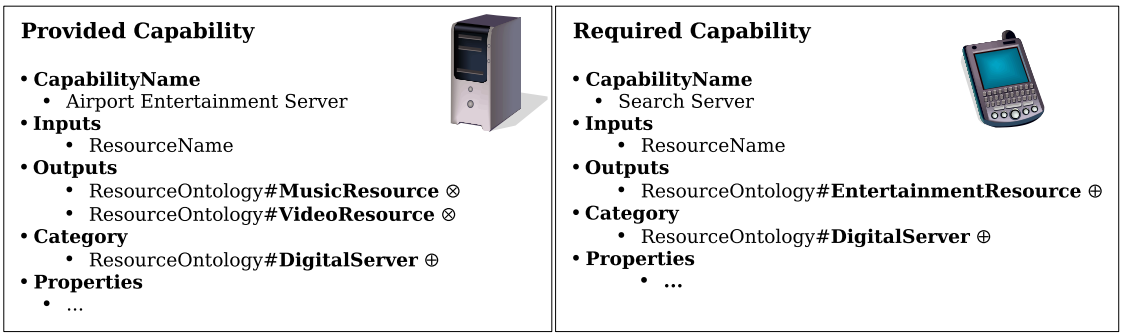


Figure 3.2: Required and Provided Capabilities

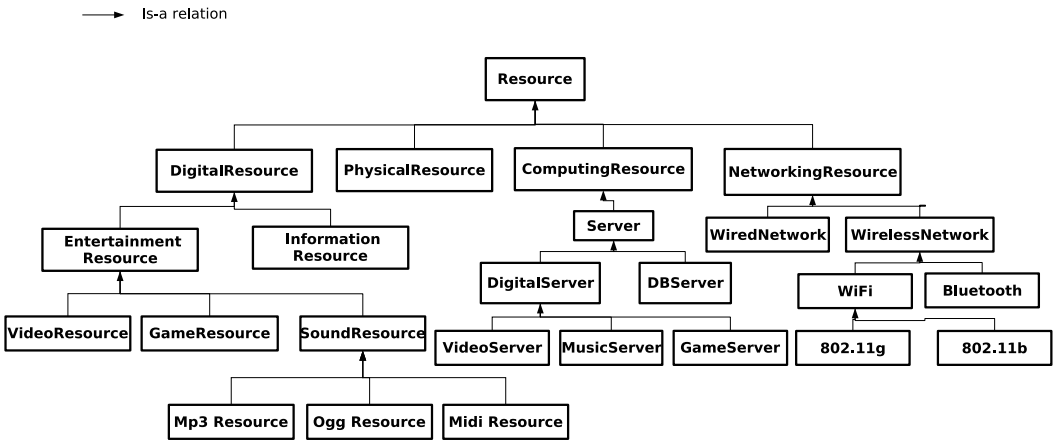


Figure 3.3: Example of an Ontology of Resources

3.2.2 Conversation Specification

The automaton, if any, associated with a provided capability, describes the conversation that the service requester has to perform in order to get the functionality advertised by the capability. In the case of required capabilities, a conversation prescribes a possible way of composing this capability out of the environment's capabilities. A service conversation is defined as a finite state automaton having a finite set of *States*, a finite set of *Symbols*, a *Transition Function*, an *InitialState* and a finite set of *FinalStates*. The automaton symbols are themselves capabilities. This means that a capability may itself be composed of other capabilities.

Figure 3.4 shows the description of the *EASY-COM* user task introduced in the scenario presented in Chapter 1. This task is modelled as a service that has two required capabilities, i.e., the *EASY-Phone* capability and the *EASY-Movie* capability. The EASY-Movie capability is represented with its associated inputs, outputs, category, properties and conversation. This conversation is modelled as an automaton where each transition label refers to another required capability. For instance, the capability *SearchServer* depicted in the beginning of the automaton refers to the capability of the same name depicted in Figure 3.2.

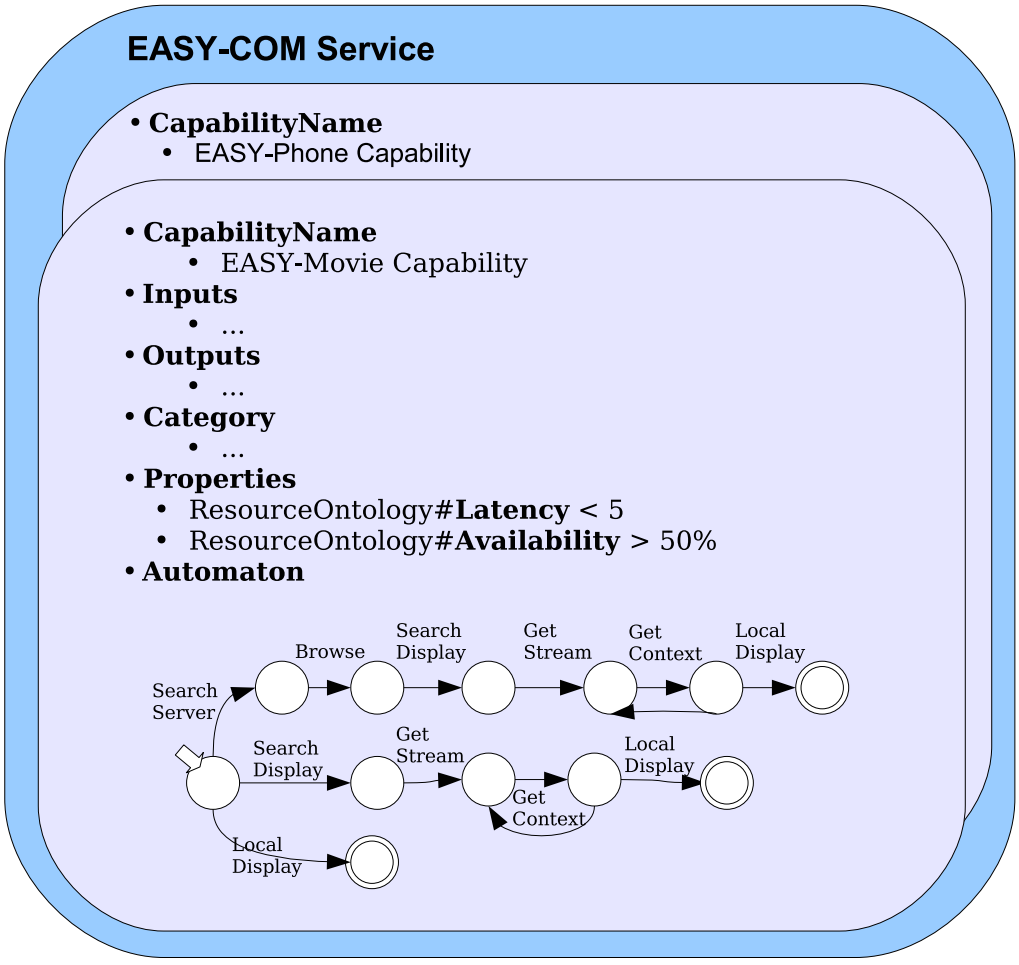


Figure 3.4: EASY-COM Service

As part of the specification of service conversations, our model supports the represen-

tation of data flow between capabilities. A data flow is specified when output information produced by a capability is consumed as input information of another capability. This can be graphically represented as part of the automaton model by introducing vertices for representing capabilities and labelled edges representing the dependencies between capabilities. Labels on edges represent the flow of data among capabilities. Figure 3.5 represents the conversation of the EASY-Movie user task integrating our graphical representation of data flow. In this figure, multiple data flow specifications have been defined between the capabilities. For instance, a data flow is defined between the *Browse* and the *DisplayStream* capabilities. This means that the *VideoStream*, which results from the selection of a movie on a video server is used as an input of the display functionality of a display device.

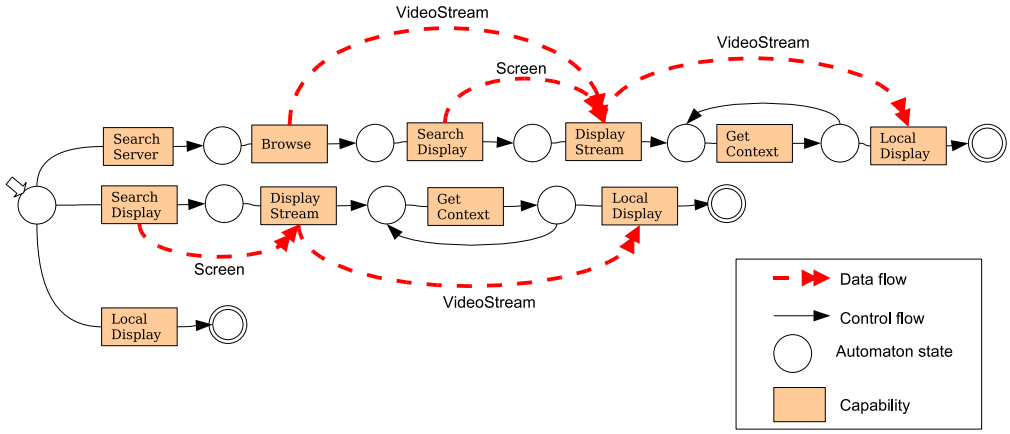


Figure 3.5: Data Flow Graphical Representation

3.2.3 Non-functional Properties

In our model, non-functional properties can be specified at two levels: at the composite capability level, i.e., *global non-functional properties* and at the elementary capability level, i.e., *local non-functional properties*. Global non-functional properties apply to the composite capability as a whole, while local non-functional properties describe features of the elementary capability itself. In both cases, a non-functional property is related to

QoS. These properties can be either *Quantitative* or *Qualitative* [Liu, 2006]. Quantitative non-functional properties, also referred to as metrics are related to quantifiable QoS attributes of the service (e.g., latency, availability). Qualitative non-functional properties, also referred to as policies, are defined using non-quantifiable QoS attributes that dictate the non-functional behaviour of the service (e.g., security, trust). In our model, these properties are defined with semantic annotations while quantitative properties are defined as numeric expressions.

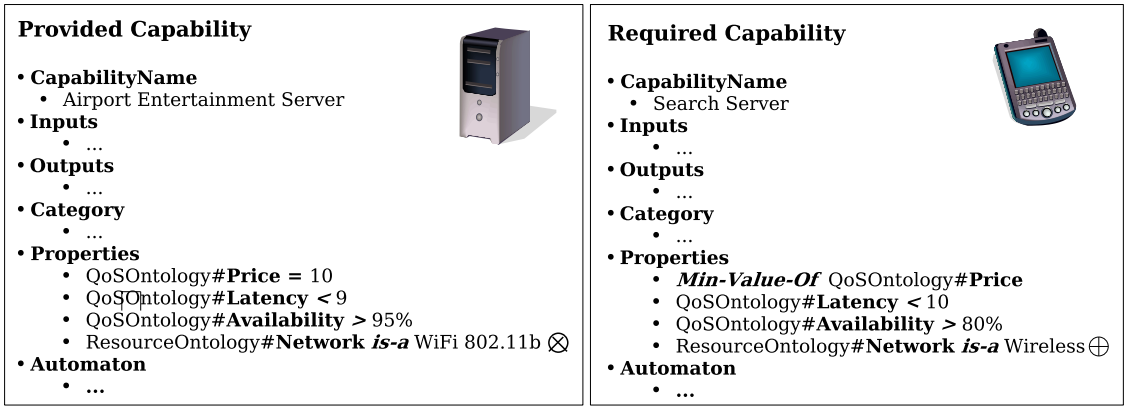


Figure 3.6: Specifying Non-Functional Properties

Figures 3.4 and 3.6 exemplify the specification of global and local non-functional properties respectively. In the first figure, the EASY-Movie composite capability has two required non-functional properties. The first property, i.e., $latency<5$ expresses the fact that the composed user task should have a global execution time less than 5 units of time, while the second property, which is related to availability, expresses the need that the composed user task should have a percentage of availability greater than 50%. These properties have to be fullfilled by a composition of provided capabilities. On the other hand, the local non-functional properties associated with the elementary capabilities *Airport Entertainment Server* and *Search Server* in Figure 3.6 express respectively properties provided and required by these capabilities and do not relate to any other capabilities. Both capabilities have qualitative and quantitative non-functional properties. The first three properties of the provided capability are quantitative properties. Their names cor-

respond to concepts representing quantitative properties taken from the QoS Ontology defined in [Liu, 2006] and depicted in Figure 3.7. In that ontology, dark colored boxes correspond to qualitative QoS properties while light colored boxes correspond to quantitative QoS properties. Quantitative properties are described using numeric expressions (e.g., $Price = 10$). Furthermore, a qualitative property is described in both the provided and the required capabilities, which is related to the networking resources being used by each of the capabilities.

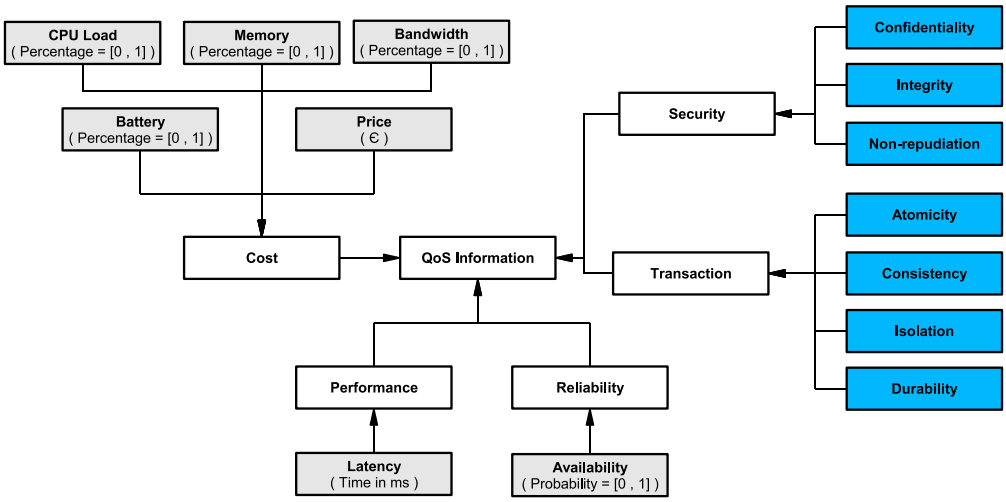


Figure 3.7: Example of QoS Ontology

3.3 Formalizing the Semantic Service Model

We introduce in this section a formalization of our semantic service model, which serves as a basis for defining our proposed conformance relations and service composition algorithms presented in Chapters 4 and 5, respectively.

Consider a finite set of services $\mathcal{S}$, a finite set of capabilities $\mathcal{C}$, a finite set of non-functional properties $\mathcal{P}$, a finite set of ontologies $\mathcal{O}$, and a finite set of concepts $\mathcal{N}$ across the set of ontologies $\mathcal{O}$. A service s in $\mathcal{S}$ is defined as a set of capabilities and non-functional properties as follows:

$(s \in \mathcal{S}) \Leftrightarrow (\exists C \subset \mathcal{C}, \exists P \subset \mathcal{P} : s =< C, P >)$

Provided and Required Capabilities

A capability c from the set of capabilities $\mathcal{C}$ is defined by a tuple $c =< I, O, cat, P, A >$ where :

- I is the set of inputs consumed by c;
- O is the set of outputs produced by c;
- cat is the category of c;
- $P \subset \mathcal{P}$ is the set of non-functional properties characterizing c and
- $A =< Q, \Sigma, \delta, st_0, F >$ is a finite state automaton describing the conversation of the capability c, as detailed below:

An input, output or category is defined as a tuple: $< Name, SemanticAnnotation >$, where $SemanticAnnotation$ is provided only for semantic enhanced services, as: $SemanticAnnotation =< o, n, at >$ where $o \in \mathcal{O}$, $n \in \mathcal{N}$ and $at \in \{\oplus, \otimes\}$ characterizes the annotation type.

The annotation type associated with a semantic concept is defined as follows. Consider the set $\{n_1, n_2, ..., n_n\}$ of all the sub-concepts of a concept n in an ontology $\mathcal{O}$ including n itself. The semantics of $n\oplus$ and $n\otimes$ is defined as follows:

$n\oplus = n_1 \vee n_2 \vee ... \vee n_n$ and

$n\otimes = n_1 \wedge n_2 \wedge ... \wedge n_n$.

Conversation Specification

The automaton $A =< Q, \Sigma, \delta, st_0, F >$ describing the conversation of a capability c is defined as follows:

- Q is a finite set of states;

- $\Sigma \subset \mathcal{C}$ is a finite set of symbols representing capabilities, i.e., the alphabet of the language the automaton accepts;
- δ is the transition function, that is $\delta : Q \times \Sigma \rightarrow Q$;
- st_0 is the start state, that is, the state in which the automaton is when no input has been processed yet (obviously, $st_0 \in Q$)
- F is a set of states of Q (i.e., $F \subset Q$), called accept states.

Van der Aaalst *et al.* in [van der Aalst et al., 2000] identified twenty control patterns for representing service conversations and for providing a comprehensive comparison of existing conversation languages with respect to these patterns. In our model, we support the set of basic control patterns identified in this work, as they are supported by most conversation specification languages, and advanced patterns can be build based on this elementary set. Figure 3.8 represent the rules that we define for mapping these basic control flow patterns into finite state automata. In this figure an elementary capability c is represented with an automaton $< Q, \Sigma, \delta, st_0, F >$, where :

- $Q = \{st_0, st_1\}$;
- $\Sigma = \{c\}$;
- $\delta(st_0, c) = st_1$;
- st_0 is the start state ;
- $F = \{st_1\}$.

A composite capability, i.e., a capability that uses one or more of the basic control flow patterns, is translated to an automaton by recursively applying the mapping rules defined in Figure 3.8 as follows: consider a set of capabilities $c_1, c_2, ..., c_n$,represented by the automata $< Q_1, \Sigma_1, \delta_1, st_{0,1}, F_1 >, < Q_2, \Sigma_2, \delta_2, st_{0,2}, F_2 >, ..., < Q_n, \Sigma_n, \delta_n, st_{0,n}, F_n >$, respectively, a composite capability c is represented by the automaton $< Q, \Sigma, \delta, st_0, F >$ according to the control pattern it uses, as follows:

- c=**Sequence**$(c_1, c_2, ..., c_n)$
  - $Q = \bigcup Q_i$;
  - $\Sigma = \bigcup \Sigma_i \bigcup \{\epsilon\}$;

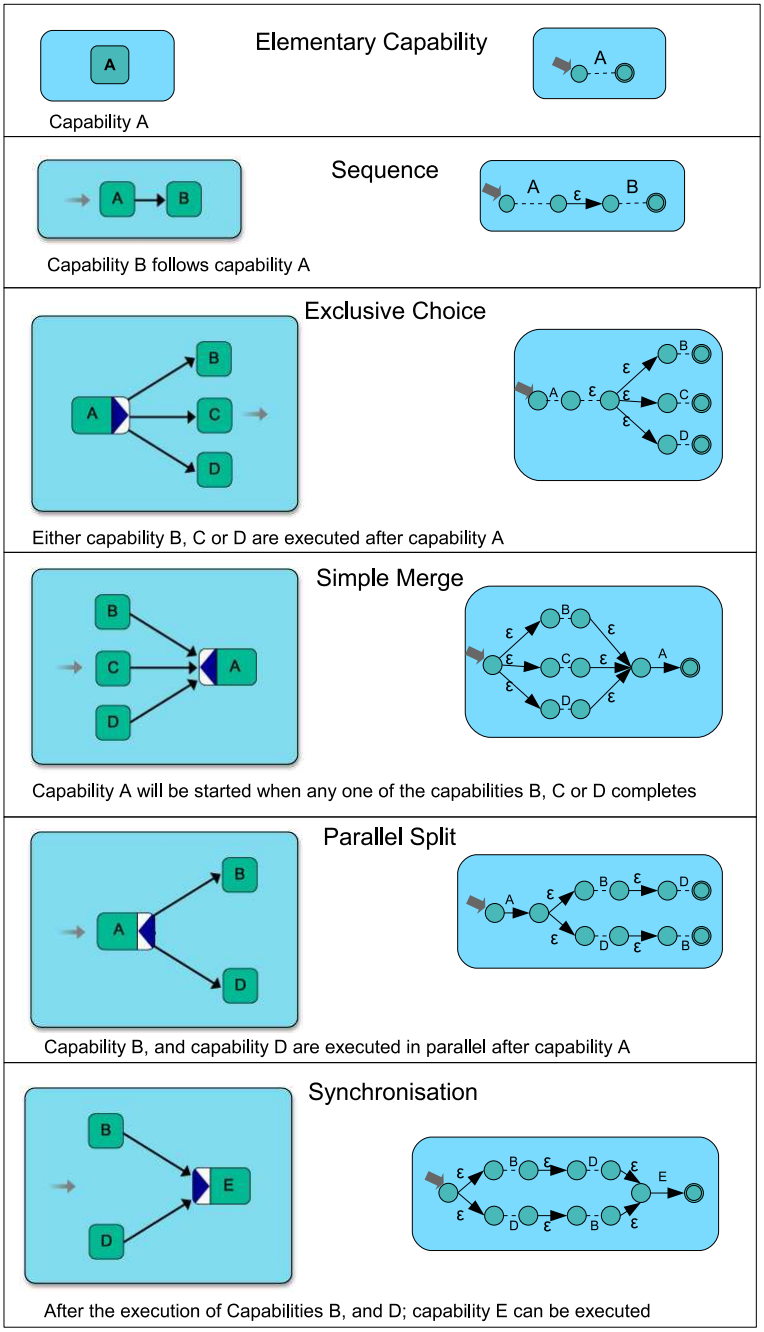


Figure 3.8: Basic Control Flow Patterns Modelled with Finite State Automata

– $\delta : \bigcup(Q_i \times \Sigma_i) \rightarrow \bigcup Q_i$
$(x, y) \mapsto \delta(x, y) = \delta_i(x, y)$ when $(x, y) \in Q_i \times \Sigma_i$ and $\delta(x, y) = st_{0,i+1}$ when $x \in F_i$

$(i \neq n)$ and $y = \epsilon$;

- $st_0 = st_{0,1}$;
- $F = F_n$.

- c=**ExcusiveChoice**$(c_1, c_2, ..., c_n)$
  - $Q = (\bigcup Q_i) \cup st_{Init}$;
  - $\Sigma = \bigcup \Sigma_i \bigcup \{\epsilon\}$;
  - $\delta : \bigcup (Q_i \times \Sigma_i) \rightarrow \bigcup Q_i$
    $(x, y) \mapsto \delta(x, y) = \delta_i(x, y)$ when $(x, y) \in Q_i \times \Sigma_i$ and $\delta(x, y) = st_{0,i}$ when $x = st_{Init}$ and $y = \epsilon$;
  - $st_0 = st_{Init}$;
  - $F = \bigcup F_i$.

- c=**SimpleMerge**$(c_1, c_2, ..., c_n)$
  - $Q = (\bigcup Q_i) \cup \{st_{Init}, st_{Final}\}$;
  - $\Sigma = \bigcup \Sigma_i \bigcup \{\epsilon\}$;
  - $\delta : \bigcup (Q_i \times \Sigma_i) \rightarrow \bigcup Q_i$
    $(x, y) \mapsto \delta(x, y) = \delta_i(x, y)$ when $(x, y) \in Q_i \times \Sigma_i$ and $\delta(x, y) = st_{0,i}$ when $x = st_{Init}$ and $y = \epsilon$ and $\delta(x, y) = st_{Final}$ when $x \in F_i$ and $y = \epsilon$
  - $st_0 = st_{Init}$;
  - $F = st_{Final}$.

- c=**ParallelSplit**(c_1, c_2): is treated as
  c=**ExclusiveChoice**(***Sequence***(c_1, c_2),***Sequence***(c_2, c_1))

- c=**Synchronisation**$(c_1, ..., c_n)$: is treated as
  c=**SimpleMerge**(***ParallelSplit***$(c_1, ..., c_n)$)

We integrate the data flow definition within our automata model using the following function:

$\Phi : \Sigma \longrightarrow 2^{\Sigma \times \mathcal{N}^2}$

$c \longmapsto \Phi(c) = \{< c_i, o_i, i_i >: i = 0..n\}$

This is interpreted as: the output o_i produced by the capability c is consumed by the capability c_i as the input i_i.

Non-Functional Properties

The set $\mathcal{P}$ of non-functional properties is defined as the union of two sets: $\mathcal{P} = \mathcal{P}_{QL} \cup \mathcal{P}_{QN}$: where $\mathcal{P}_{QL}$ is the set of qualitative properties, and $\mathcal{P}_{QN}$ is the set of quantitative properties. A qualitative property $pql \in \mathcal{P}_{QL}$ is defined as a tuple with a name and a value. The name corresponds to a concept describing qualitative non-functional properties and the value is defined as a semantic annotation. In other words: $pql =< Name, Value >, Name \in \mathcal{N}_{QL}$ where $\mathcal{N}_{QL} \subset \mathcal{N}$ is a set of concepts describing qualitative properties, and $Value \in \mathcal{N}$. A quantitative property $pqn \in \mathcal{P}_{QN}$ is defined with a couple $< Name, Value >$ where $Name \in \mathcal{N}_{QN}$ is a concept describing quantitative properties and $Value$ is a numeric expression built using the operators $=, <, \leq, >, \geq$ and values from $\mathbb{R}$.

3.4 Concluding Remarks

Semantic service specification has been a very active field of research in the last few years, which has led to the emergence of a number of semantic service specification languages. To enable the full potential of pervasive computing environments, a SOM should enable service requesters that rely on a specific service description language to discover, access and compose services that are described using different service description languages, including semantic and syntactic ones. Towards this purpose, we presented in this chapter a semantic service model enabling the specification of service functional and non-functional capabilities as well as service conversations, which serves as a basic enabler for both semantic-based and syntactic-based multi-language interoperability. Multi-language interoperability achieved by the multi-protocol management functionality of our middleware (introduced in Section 2.4) consists on translating incoming service advertisements and service requests into a language compliant with our service model as presented in Chapter 6.

Chapter 4

Efficient Semantic Service Registry for Pervasive Computing Environments

Service publication, location and matching supported by a service registry are essential functionalities of a SOM. To fit the requirements of pervasive computing environments, these middleware functionalities have to deal with a number of issues. Specifically, service matching have to consider the heterogeneity of service descriptions. Indeed, service descriptions coming from different middleware platforms may have different levels of expressiveness (e.g., rich semantic service descriptions, syntactic service descriptions). Semantic service descriptions may further be specified using different annotation types. Furthermore, preferences among non-functional properties of services have to be considered in order to provide the users with services that best fit their requirements. Finally, the efficiency of the registry has to be considered to fit resource constrained devices on which the middleware may be deployed. The efficiency of the semantic service registry depends mainly on the efficiency of the semantic matching of service capabilities. Based on the semantic service description languages, such as the ones surveyed in Chapter 3, a number of research efforts focus on matching between services to assess the suitability of advertised services against a service request. However, these solution do not address the matching of

heterogeneous service descriptions that further include both functional and non-functional properties of services. Moreover, optimizations to semantic service matching performed by service registries are realized by overloading service publication with costly computations in order to achieve efficiency at service location time.

We survey in Section 4.1 existing semantic service matching algorithms, analyse the computational cost of such matching and discuss existing optimizations to semantic service matching. We then present our registry and the SOM functionalities it realizes in Section 4.2. We finally assess the efficiency of our overall solution in Section 4.3 and present concluding remarks in Section 4.4.

4.1 Efficient Semantic Service Matching: State Of The Art

4.1.1 Matching Semantic Service Capabilities

A number of research efforts have been conducted in the area of matching semantic Web services based on the *signature*s of their provided capabilities. Signature matching deals with the identification of subsumption relationships between the concepts describing inputs and outputs of capabilities [Zaremski and Wing, 1995]. The *subsumption* relation allows relating concepts to more generic concepts in an ontology based on their formal definitions given in description logics. After subsumption reasoning on an ontology, the resulting ontology hierarchy is referred to as the *classified ontology*. This reasoning is performed by a description logics reasoner.

A base algorithm for service signature matching has been proposed by Paolucci *et al.* in [Sycara et al., 2003, Paolucci et al., 2002]. This algorithm allows matching a required capability, described as a set of *provided inputs and required outputs*, with a number of provided capabilities, described each as a set of *required inputs and provided outputs*. Inputs and outputs are semantically annotated with ontology concepts using the *all-value-from* annotation type. Specifically, the algorithm defines four levels of matching between a provided and a required ontology concept representing an input or an output. These four levels are:

- *exact*: if the concepts are equivalent or if the required concept is a direct subclass of the provided one,
- *plug in*: if the provided concept subsumes the required one and the latter is not a direct subclass of the former,
- *subsumes*: if the required concept subsumes the provided one, and
- *fail*: if there is no subsumption relation between the two concepts.

Based on these four levels of match between concepts, the matching algorithm defines a scoring function used for service ranking, ordered in the following way: exact > plug in > subsumed > fail.

Other solutions extending the above signature matching of semantic Web services have been proposed in the literature [Majithia et al., 2004, Trastour et al., 2001, Filho and van Sinderen, 2003]. However, all these algorithms neither consider the matching of service non-functional properties, nor the computational cost associated with their matching functions.

Another kind of matching, called *specification* matching has been investigated in the literature [Zaremski and Wing, 1997, Sirin et al., 2005, Sycara et al., 1999]. Specification matching deals with matching pre- and post-conditions that describe the functional semantics of services. For instance, in [Zaremski and Wing, 1997], specification matching is performed using theorem proving, i.e., inferring general subsumption relations between logical expressions that specify pre- and post-conditions of services. A more practical way to perform specification matching is to use *query containment* [Sirin et al., 2005, Sycara et al., 1999]. This is done by modelling both service advertisements and service requests as queries with a set of constraints (e.g., required inputs and outputs are modelled as restrictions on their types). Starting from the specified constraints, the possible values of both queries are evaluated, and possible inclusions between the results of the queries are inferred. Specifically, a query q_1 is contained in q_2 if all the answers of q_1 are included in the answers of q_2. Compared to signature matching, specification matching realized through query containment requires to have a central knowledge base with all the ontology instances, which is hardly achievable in open pervasive computing environments.

We thus focus in this thesis on matching based on semantic-enhanced service signatures. The key issue for efficient signature matching lies in the performance of the underlying semantic reasoning on ontologies as discussed in the following section.

4.1.2 Cost of Semantic Service Matching

From the experiments presented in [Ben Mokhtar et al., 2006b], it has been identified that the computational cost of semantic reasoning is inappropriate with respect to the interactive feature of pervasive computing environments. Indeed, assessing the conformance of a service request against a single service advertisement involving ten concepts that belong to a single ontology of a hundred concepts, generates an execution overhead in the order of four to five seconds[1]. This experiment has been realized using all the three publicly available reasoners from the scientific community, i.e., Racer[2], FaCT++[3] and Pellet[4], and results where similar for all three reasoners. This processing overhead has further to be multiplied by the number of services with which the conformance of the request is checked and the number of ontologies employed for describing each service.

In more detail, to perform semantic service matching, each pair of concepts c_i, c_j describing service elements (e.g., inputs, outputs) has to be compared. To do this, a memory model representing the ontology defining these concepts is created in order to reason over its structure. Then, these ontologies have to be *classified* in order to infer all the subsumption relations between concepts from their formal definitions. A parsing of the classified ontologies is then performed to locate the concepts c_i and c_j and assess the relationship between them among the following:

- c_1 subsumes c_2 if c_1 is an ancestor of c_2 in the classified ontology hierarchy,
- c_2 subsumes c_1 if c_2 is an ancestor of c_1 in the classified ontology hierarchy,
- empty if there is no subsumption relation between c_1 and c_2 in the classified ontology.

[1]Experiment Conditions: Notebook with a 1.6 GHz Intel Centrino processor and 512 MB of RAM
[2]Racer: http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[3]FaCT++: http://owl.man.ac.uk/factplusplus/
[4]Pellet: http://www.mindswap.org/2003/pellet/

Finally, the overall relation between the service request and the service advertisement can be assessed from the identified subsumption relations previously computed for each pair of concepts as discussed in Section 4.1.1.

In [Ben Mokhtar et al., 2006b], it is also established that ontology parsing (i.e., creation of the memory model representing the ontology) and ontology classification are the most costly phases in semantic service matching (i.e., 80% of the total execution overhead) (see Chapter 6 for a detailed evaluation). Thus, optimizations need to be investigated in order to adapt this costly process to the constraining features of pervasive computing environments.

4.1.3 Optimizations to Semantic Service Matching

Two kinds of optimizations can be investigated to reduce the cost of semantic service matching :

1. Enabling efficient subsumption assessment between ontology concepts. This can be achieved by performing ontology classification offline and investigating mechanisms for rapidly inferring subsumption between concepts at runtime.

2. Organizing semantic service advertisements in service registries, so that the number of matchings performed for resolving a service request is reduced.

Optimizing subsumption assessment between ontology concepts is similar to optimizing subsumption assessment between classes in object-oriented programming languages. In this area, algorithms for encoding multiple-inheritance class hierarchies have been investigated (e.g., [Caseau, 1993, Krall et al., 1997, Ait-Kaci et al., 1989, van Bommel and Beck, 1999]). The rationale behind the encoding of class hierarchies is to assign a numeric code to each class in order to assess the relationship between two classes by numerically comparing their codes instead of browsing the whole hierarchy. However, employing existing algorithms for encoding ontologies also needs to deal with issues typical to knowledge representation, such as support for conflict-free encoding for large ontologies while achieving efficient matching.

For organizing semantic service advertisements in service registries, solutions may be sought in service classifications. The OWL-S service description language provides the

means for defining hierarchies of service descriptions called *profile hierarchies*[5]. These hierarchies are similar to the object-oriented inheritance hierarchies. For instance, when a new service profile is defined, it may be specified as a subclass of an existing profile class. This allows the new service to inherit all the properties of all the classes specified in its super-hierarchy of classes. While this approach allows the classification of service profiles according to the classes from which they inherit, it does not allow considering possible relations between service capabilities that do not have a common set of properties but still provide similar functional features.

Service classification can also be based on the service category using existing taxonomies such as NAICS[6] or UNSPSC[7]. However, service categories alone do not give enough information about the service functionality.

Other solutions that combine both encoding and registry organization techniques have been investigated. In this area, [Constantinescu and Faltings, 2003] propose to numerically encode service descriptions and use the Generalized Search Tree (GiST) algorithm proposed by Hellerstein in [Hellerstein et al., 1995] for creating and maintaining the registry of numerically encoded services. Combining both encoding and indexing techniques allows performing efficient service location, in the order of milliseconds for trees of 10000 entries. However, insertion within trees of this size to realize service publication, is still a heavy process that takes approximately 3 seconds.

In [Srinivasan et al., 2004], the authors propose an approach to optimizing service location in a UDDI registry augmented with OWL-S for the description of semantic Web services. In this approach, the authors propose to exploit the service publication phase to perform semantic reasoning and pre-compute information that will help to efficiently answer service requests. Performance evaluation of this approach shows that the service publication phase employing this algorithm takes around seven times the time taken by UDDI to publish a service. On the other hand, the time to process a service request is in the order of milliseconds.

[5]OWL-S Profile Hierarchies: http://www.daml.org/services/owl-s/1.1/ProfileHierarchy.html
[6]NAICS taxonomy: http://www.census.gov/epcd/www/naics.html
[7]UNSPSC taxonomy: http://www.unspsc.org/

While the two above approaches opt for overloading the service publication phase with costly computations in order to later achieve efficiency upon resolving service requests, we aim at achieving both lightweight service publication and location, as both operations need to be performed on resource constrained-devices.

4.1.4 Discussion

The survey of existing research efforts towards semantic service matching demonstrates that the proposed matching algorithms consider service descriptions coming from a single service description language. They focus on matching service functional properties and do not enable the evaluation of the degree of match between services with respect to both functional and non-functional properties. Furthermore, the efficiency of the proposed semantic service matching algorithms is not assessed. Finally, solutions to optimized semantic service discovery achieve efficiency of semantic service location by overloading service publication. Thus, the quest for an efficient service registry for pervasive computing environments is still open. This registry should:

- Enable flexible service matching that supports both semantic and syntactic service descriptions for enabling multi-language interoperability
- Support semantic descriptions with different annotation types
- Assesses the conformance and evaluate the degree of conformance between service capabilities based on both service functional and non-functional properties
- Support an appropriate ontology encoding mechanism to perform efficient semantic service matching
- Enable organizing of service descriptions to support both efficient service publication and location

4.2 Efficient Semantic Service Registry

Figure 4.1 introduces the architecture of our efficient semantic service registry for pervasive computing. This registry allows heterogeneous service capabilities to be registered and retrieved by translating their corresponding descriptions to the model introduced in the previous chapter.

This registry is composed of three of the functionalities of our SOM. Specifically, service matching, presented in Section 4.2.1 presents a set of conformance relations to assess the suitability of a service advertisement with a service request. Efficient semantic service matching presented in Section 4.2.2 allows efficiently assessing the conformance between two service descriptions through appropriate ontology encoding algorithms. Service publication, presented in Section 4.2.4, is used to efficiently classify service advertisements in the registry. Finally, service location, presented in Section 4.2.5 is used to efficiently retrieve a ranked list of service descriptions that conform to a service request. In addition, an index, presented in Section 4.2.3, in maintained to efficiently access to the classified service advertisements.

We present in the following sections each of these SOM functionalities.

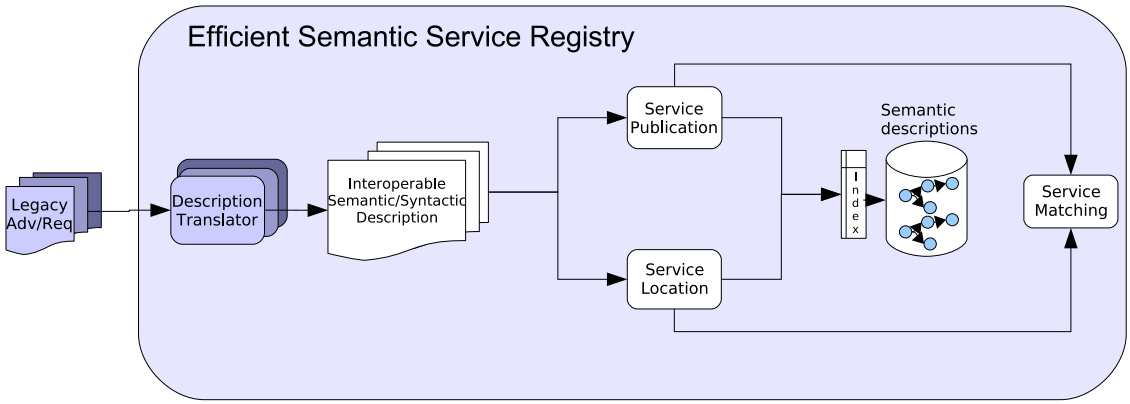


Figure 4.1: Registry Architecture Overview

4.2.1 Service Matching

Based on our semantic service model presented in Chapter 3, we present in this section a set of conformance relations for matching the description of a required service with the description of a provided service. The conformance relations presented in this section deal with both functional and non-functional capabilities of services. They are based on the relation **ConceptMatch**(), which is used for matching two concepts n_1, n_2 from the set of concepts of an ontology O associated with their respective annotation type, either $\oplus$ or $\otimes$. Based on this relation we present the relations **ElementMatch**() and **FunctionalCapabilityMatch**() for matching two capability elements, and two service capabilities (respectively). We finally, present a set of functions to evaluate the semantic distance between capabilities.

Matching Two Semantic Concepts

. The relation **ConceptMatch**(n_1,n_2), where n_1 is a provided concept and n_2 is a required concept, decomposes in three cases with respect to the annotation type associated with the concepts n_1 and n_2 as follows:

$$\begin{aligned}\textbf{ConceptMatch}(n_1^{\otimes},n_2^{\otimes}) \Leftrightarrow\ & n_1^{\otimes} = (c_1 \wedge c_2 \wedge c_3 \wedge ...) \wedge \\ & n_2^{\otimes} = (c'_1 \wedge c'_2 \wedge c'_3 \wedge ...) \wedge \\ & \forall c'_j, \exists c_i : c'_j = c_i \\ \\ \Leftrightarrow\ & \textbf{Subsume}(n_1,n_2)\end{aligned}$$

$$\begin{aligned}\textbf{ConceptMatch}(n_1^{\otimes},n_2^{\oplus}) \Leftrightarrow\ & n_1^{\otimes} = (c_1 \wedge c_2 \wedge c_3 \wedge ...) \wedge \\ & n_2^{\oplus} = (c'_1 \vee c'_2 \vee c'_3 \vee ...) \wedge \\ & \exists c_i, \exists c'_j : c_i = c'_j \\ \\ \Leftrightarrow\ & n_1^{\otimes} = (c_1 \wedge c_2 \wedge c_3 \wedge ...) \wedge \\ & n_2^{\oplus} = (c'_1 \vee c'_2 \vee c'_3 \vee ...) \wedge\end{aligned}$$

$$\{c_i\} \cap \{c'_j\} \neq \emptyset$$

$$\Leftrightarrow \textbf{Subsume}(n_1,n_2) \vee$$
$$\textbf{Subsume}(n_2,n_1) \vee$$
$$\exists c \in O : \textbf{Subsume}(n_1,c) \wedge \textbf{Subsume}(n_2,c)$$

$$\textbf{ConceptMatch}(n_1^{\oplus},n_2^{\oplus}) \Leftrightarrow n_1^{\oplus} = (c_1 \vee c_2 \vee c_3 \vee ...) \wedge$$
$$n_2^{\oplus} = (c'_1 \vee c'_2 \vee c'_3 \vee ...) \wedge$$
$$\forall c_i, \exists c'_j : c_i = c'_j$$

$$\Leftrightarrow \textbf{Subsume}(n_2,n_1)$$

Where the relation **Subsume**() between two concepts n_1 and n_2 of an ontology O holds if and only if the concept n_1 subsumes the concept n_2 in O, i.e., n_1 is more generic than or is equivalent to n_2 in O after ontology classification.

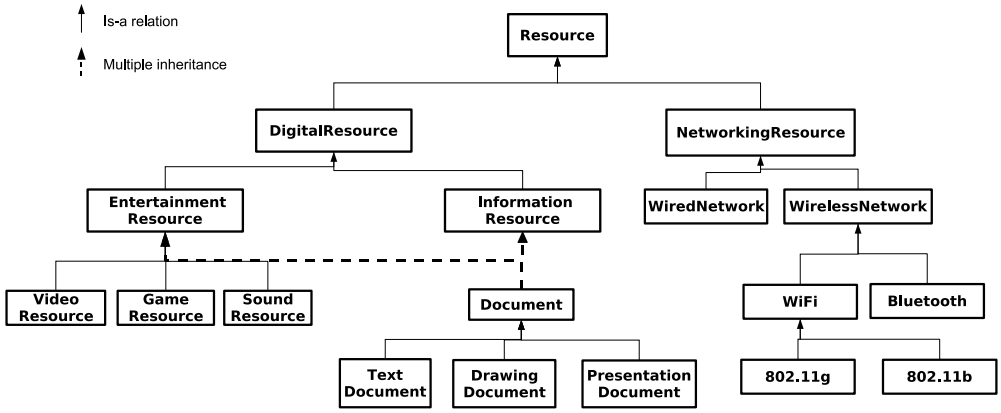


Figure 4.2: Ontology Example

The first case describes the situation where both the provided and the required concepts are associated with the all-values-from annotation type. This means that either concept

can be replaced by a conjunctive clause including the concept itself and all the concepts from its respective sub-hierarchy. Hence, the matching between these two concepts holds if and only if n_1 subsumes n_2. For instance, according to the ontology of resources depicted in Figure 4.2, **ConceptMatch**($WirelessNetwork^{\otimes}$,$WiFi^{\otimes}$) holds because:

$(WirelessNetwork^{\otimes} \equiv WirelessNetwork \wedge WiFi \wedge Bluetooth \wedge 802.11g \wedge 802.11b) \Rightarrow (WiFi^{\otimes} \equiv WiFi \wedge 802.11g \wedge 802.11b)$

The second case describes the situation where a required concept is associated with a some-values-from annotation type while the provided concept is associated with a all-values-from annotation type. In this case, as the required concept translates into a disjunction of its sub-concepts it can be satisfied if there is an intersection between the sub-concepts of the former and the sub-concepts of the latter. For instance, both **ConceptMatch**($WirelessNetwork^{\otimes}$,$WiFi^{\oplus}$) and **ConceptMatch**($802.11g^{\otimes}$,$WiFi^{\oplus}$) hold because:

$(WirelessNetwork^{\otimes} \equiv WirelessNetwork \wedge WiFi \wedge Bluetooth \wedge 802.11g \wedge 802.11b) \Rightarrow (WiFi^{\oplus} \equiv WiFi \vee 802.11g \vee 802.11b)$

and

$(802.11g^{\otimes} \equiv 802.11g) \Rightarrow (WiFi^{\oplus} \equiv WiFi \vee 802.11g \vee 802.11b)$

Additionally, in this case a matching may hold also if there no subsumption between the provided and the required concept but there is an intersection between their sub-concepts. This is due to the multiple-inheritance structure of ontologies. For instance, a matching holds between the concepts: $InformationResource^{\otimes}$ and $EntertainmentResource^{\oplus}$ in Figure 4.2, as there is an intersection between the sub-hierarchies of these two concepts.

Finally, the last case represents the situation where both the provided and the required concepts are associated with the annotation type some-values-from. In this case the matching holds only if the required concept subsumes the provided one. In all the other cases, the matching can not be assessed. For instance, **ConceptMatch**($WiFi^{\oplus}$,$WirelessNetwork^{\oplus}$) holds because:

$(WiFi^{\oplus} \equiv WiFi \vee 802.11g \vee 802.11b) \Rightarrow (WirelessNetwork^{\oplus} \equiv WirelessNetwork \vee WiFi \vee Bluetooth \vee 802.11g \vee 802.11b)$

Notice that our matching relation does not consider the case where the provided concept has a some-values-from annotation type while the required one has an all-values-from annotation type, i.e., **ConceptMatch**($n_1^{\oplus}$,$n_2^{\otimes}$). In this case, whatever is the **Subsume**() relation between the two concepts, we can not insure that the matching holds. The only assertion that we can make is that if the relation **Subsumption**(n_1,n_2) does not hold this implies that the relation **ConceptMatch**(n_1,n_2) does not hold neither. In other words:

$\neg$ **Subsume**(n_1,n_2) $\Rightarrow$ $\neg$ **ConceptMatch**($n_1^{\oplus}$,$n_2^{\otimes}$)

Indeed, if n_1 does not subsume n_2, i.e., n_2 is more generic than n_1 or n_2 is not in the same hierarchy as n_1 at all, the annotation type of n_1 translates into a disjunction of concepts that contains at most a sub-part of the sought concept n_2. This does not satisfy n_2 as its annotation type translates into a concept conjunction of all its sub-hierarchy of concepts and not only part of them. Furthermore, **ConceptMatch**() can not be asserted if the relation **Subsumption**(n_1,n_2) holds, but in this case, contrary to the previous one, there is still a chance for the provided concept to satisfy the required concept at runtime. If we consider the previous example, neither **ConceptMatch**($WirelessNetwork^{\oplus}$,$WiFi^{\otimes}$) nor **ConceptMatch**($802.11g^{\oplus}$,$WiFi^{\otimes}$) holds because:

$(WirelessNetwork^{\oplus} \equiv WirelessNetwork \vee WiFi \vee Bluetooth \vee 802.11g \vee 802.11b) \not\Rightarrow (WiFi^{\otimes} \equiv WiFi \wedge 802.11g \wedge 802.11b)$

and

$(802.11g^{\oplus} \equiv 802.11g) \not\Rightarrow (WiFi^{\otimes} \equiv WiFi \wedge 802.11g \wedge 802.11b)$

While in the second case it is obvious that the provided concept does not satisfy the required one, we can not exclude a possible matching of the provided concept with the required one in the first case at runtime. Thus, as we aim at automated semantic service matching, we opt for ignoring this possibility by considering that if a required concept is associated with an all-value-from annotation type while the provided concept is associated

with a some-values-from annotation type the matching fails.

Matching Two Capability Elements

Elements describing service capabilities, i.e., input, output, category, can be associated with a semantic annotation or be syntactically defined with their names (if the semantic annotation field is empty, with respect to our model of Figure 3.1). In the case of two semantic elements, matching is performed using the **ConceptMatch**() relation, while it is realized through a syntactic comparison of element names if one of the two compared elements (or both of them) are syntactically described. More formally, the relation **ElementMatch**() is defined as follows:

if $e_1.SemanticAnnotation \neq \emptyset$ and $e_2.SemanticAnnotation \neq \emptyset$:
ElementMatch(e_1,e_2) $\Leftrightarrow$ **ConceptMatch**($e_1.SemanticAnnotation$,$e_2.SemanticAnnotation$)
else
ElementMatch(e_1,e_2) $\Leftrightarrow$ $e_1.Name = e_2.Name$

Matching Two Service Capabilities

Using the relation **ElementMatch**(), we define the relation **FunctionalCapabilityMatch**() for matching functional properties of a provided capability $c_1 =< I_1, O_1, cat_1, P_1, A_1 >$ with a required capability $c_2 =< I_2, O_2, cat_2, P_2, A_2 >$ in $\mathcal{C}$ as follows:

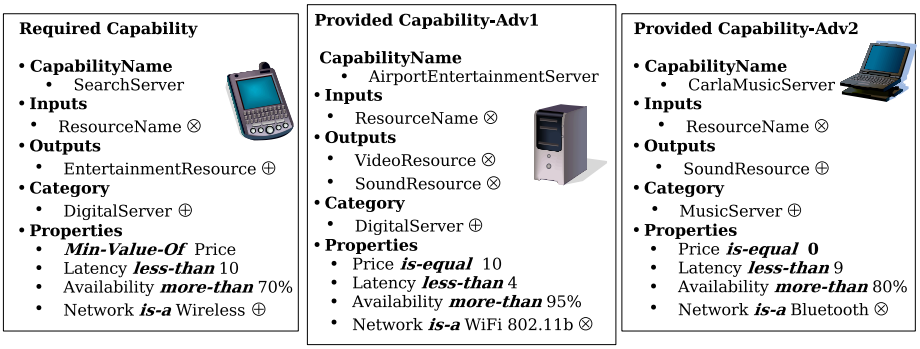
FunctionalCapabilityMatch(c_1,c_2) =
$\forall\ in_2 \in I_2, \exists\ in_1 \in I_1$: **ElementMatch**($in_1$, in_2) and
$\forall\ out_2 \in O_2, \exists\ out_1 \in O_1$: **ElementMatch**($out_1$, out_2) and
ElementMatch(cat_1, cat_2)

Matching non functional properties of capabilities is performed using the relation **PropertiesCapabilityMatch**() defined as follows:

PropertiesCapabilityMatch(c_1,c_2) =
$\forall$ pql $\in P_2.P_{QL}$, $\exists$ pql' $\in P_1.P_{QL}$: **ConceptMatch**(*pql'*, *pql*) and

$\forall$ pqn $\in P_2.P_{QN}$, $\exists$ pqn' $\in P_1.P_{QN}$: **NumericExpressionMatch**(*pqn', pqn*)

In this relation, qualitative non-functional properties are compared using the **ConceptMatch**() relation while quantitative properties are compared with the **NumericExpressionMatch**() relation. This relation is used to compare two numeric expressions using the operators $=, <, \leq, >, \geq$ and values from $\mathbb{R}$. This relation holds between two numeric expressions pqn_1 and pqn_2, if the values that satisfy pqn_1 are a subset of the values that satisfy pqn_2. For instance, **NumericExpressionMatch**($Latency = 3.5$,$Latency < 5$) holds because the value 3.5 that satisfies the first expression is included in the interval $[0, 5[$ that satisfies the second expression.



	Req	Adv1	Adv2	FunctionalDoM(Adv1,Req)	FunctionalDoM(Adv2,Req)
In	ResourceName ⊗	ResourceName⊗	ResourceName⊗	0	0
Out	Entertainment ressource ⊕	VideoResource ⊗ SoundResource ⊗	SoundResource ⊕	Min (τ_2, τ_2)	τ_3
Category	Digital Server ⊕	DigitalServer ⊕	Music Server ⊕	0	τ_3
				FunctionalDoM(Adv1,Req) $=\tau_2$	**FunctionalDoM**(Adv2,Req) $=2*\tau_3$

	Req	Adv1	Adv2	Adv1'	Adv2'	Adv'1	Adv'2
Price	Min	10	0	10	0	0,75	0,25
Latency	<10	<4	<9	4	9	0,25	0,75
Availability	>70%	>95%	>80%	95,00%	80,00%	0,25	0,75
Network	Wireless⊕	WiFi802.11b ⊗	Bluetooth ⊗	τ_2	$2*\tau_2$	0,25	0,75
						PropertiesDoM(Adv1,Req) $=0,75*\omega_1+0,25*\omega_2+0,25*\omega_3+0,25*\omega_4$	**PropertiesDoM**(Adv2,Req) $=0,25*\omega_1+0,75*\omega_2+0,75*\omega_3+0,75*\omega_4$

Figure 4.3: Matching and Evaluating the Semantic Distance Between Capabilities

Matching service conversations is performed by the service composition functionality of our middleware as presented in Chapter 5.

A complete example of matching capabilities is depicted in Figure 4.3. In this example,

two provided capabilities (Adv_1 and Adv_2) are matched with the required capability *Req*. The required capability is specified on the device of Rozalie who is looking for entertainment capabilities in the various pervasive environments that she crosses during her travel. In particular, Rozalie is looking for digital servers, which she can access by giving the title of an entertainment resource and getting the corresponding resource. According to the ontology of Figure 4.2 employed by Rozalie, this resource could be either a music, video or gaming resource. The required capability further identifies some required properties regarding latency and availability, as well as network connectivity and price. In this example, two entertainment capabilities are available in the environment. Specifically, the previously introduced *Airport Entertainment Server* (Adv_1 in the figure) and the *Carla Music Server* (Adv_2 in the figure). The first capability is offered by the airport networking infrastructure, thus providing strong QoS properties (e.g., high availability, low latency). However this capability is not free-of-charge. On the other hand, another traveller in the airport, namely Carla, allows other users to use her music resources for free, but without good QoS guarantees. Both provided capabilities match the required capability. Indeed, the input of the required capability, i.e., *ResourceName*$\otimes$, matches the inputs required by both provided capabilities, i.e., *ResourceName*$\otimes$ as they reference the same concept in the *Resource Ontology* and they all employ the same annotation type. Furthermore, the outputs and categories of both provided capabilities match the output and category of the required capability. For instance, both of the two outputs of the capability Adv_1 match the output of the required capability because:

ConceptMatch($VideoResource^{\otimes}$,$EntertainmentResource^{\oplus}$) holds and
ConceptMatch($SoundResource^{\otimes}$,$EntertainmentResource^{\oplus}$) holds
These two matchings hold because both:

ConceptMatch($SoundResource^{\otimes}$,$EntertainmentResource^{\oplus}$)
and
ConceptMatch($VideoResource^{\otimes}$,$EntertainmentResource^{\oplus}$) holds

Indeed,

ConceptMatch($SoundResource^{\otimes}$,$EntertainmentResource^{\oplus}$) holds because:

$SoundResource.AnnotationType = \otimes$ and

$EntertainmentResource.AnnotationType = \oplus$ and

Subsume($EntertainmentResource$,$SoundResource$).

Similarly, in the second case, we have:

ConceptMatch($VideoResource^{\otimes}$,$EntertainmentResource^{\oplus}$) holds because:

$VideoResource.AnnotationType = \otimes$ and

$EntertainmentResource.AnnotationType = \oplus$ and

Subsume($EntertainmentResource$,$VideoResource$)

On the other hand the output of the capability Adv_2 also matches the output of the required capability. Indeed,

ConceptMatch($SoundResource^{\oplus}$,$EntertainmentResource^{\oplus}$) holds because:

$SoundResource.AnnotationType = \oplus$ and

$EntertainmentResource.AnnotationType = \oplus$ and

Subsume($EntertainmentResource$,$SoundResource$)

Finally, matching the categories provided by the capabilities Adv_1 and Adv_2 against the category of the required capability can be performed in a similar way as shown for output matching.

Regarding non-functional properties, the first three non-functional properties of the required capability are quantitative non-functional properties. They, are all satisfied by the properties of Adv_1 and Adv_2. For instance, the property $Latency < 10$ of the required capability is satisfied by both properties $Latency < 4$ and $Latency < 9$ of Adv_1 and Adv_2 respectively. Finally, the last non-functional property of the required property, which is a

qualitative non-functional property, is also matched by both Adv_1 and Adv_2. Indeed,

ConceptMatch($WiFi802.11b^{\otimes}$,$Wireless^{\oplus}$) holds because:

$WiFi802.11b.AnnotationType = \otimes$ and

$Wireless.AnnotationType = \oplus$ and

Subsume($Wireless$,$WiFi802.11b$)

On the other hand,

ConceptMatch($Bluetooth^{\otimes}$,$Wireless^{\oplus}$) holds because:

$Bluetooth.AnnotationType = \otimes$ and

$Wireless.AnnotationType = \oplus$ and

Subsume($Wireless$,$Bluetooth$)

Semantic Service Distance

When a match is assessed between two capabilities c_1 and c_2 in $\mathcal{C}$ using the relations defined in the previous sections, we use the function **CapabilityDoM**(c_1, c_2) (where DoM stands for Degree of Match) to estimate the semantic distance between these capabilities. The semantic distance allows a service registry to select service capabilities that best conform to a service request.

The **CapabilityDoM**() function is based on the two functions **FunctionalDoM**() and **PropertiesDoM**() that define the degree of match between functional and non-functional properties of capabilities respectively. More precisely:

CapabilityDoM(c_1, c_2)=λ_1**FunctionalDoM**(c_1, c_2) + λ_2 **PropertiesDoM**(c_1, c_2)

where the weights $\lambda_1 > 0$ and $\lambda_2 > 0$ allow specifying the preference between functional and non-functional properties. For instance, a user may prefer a service that does not exactly conform to its required functional properties but adequately fulfils its required

QoS properties (e.g., security level).

Functional Degree of Match Between Capabilities

The degree of match between the functional properties of capabilities is evaluated by the aggregation of the degree of match between pairs of inputs, outputs an category elements identified after the assessment of the **ConceptMatch**() relation, as follows:

$$\begin{aligned}&\textbf{FunctionalDoM}(c_1, c_2) = \text{Min}(\\ &\quad \textstyle\sum_{i=1}^{|I_1|} \textbf{ConceptDoM}(I_2.in'_i, I_1.in_i) +\\ &\quad \textstyle\sum_{i=1}^{|O_2|} \textbf{ConceptDoM}(O_1.out_i, O_2.out'_i) +\\ &\quad \textbf{ConceptDoM}(cat_1, cat_2))\end{aligned}$$

Where **ConceptDoM**() is a function that evaluates the degree of match between two semantic concepts in $\mathcal{N}$. Syntactic elements of service capabilities are not considered in the evaluation of the degree of match between capabilities because they are :

- Either syntactically identical, and we assume that they should be also semantically identical, thus the semantic distance between them is equal to 0, which does not affect the overall semantic distance
- Or they are syntactically different, where we assume that they are also semantically different.

This degree of match between semantic concepts depends on the annotation type associated with these concepts as well as on their closeness in the classified ontology.

Specifically, for evaluating the degree of match between a provided concept n_1 and a required concept n_2 we distinguish five cases:

1. Both the provided and the required concepts are associated with an all-values-from annotation type and the provided concept subsumes the required concept,

2. The required concept is associated with a some-values-from annotation type, the provided concept is associated with an all-values-from annotation type and it subsumes the required concept,

3. The required concept is associated with a some-values-from annotation type, the provided concept is associated with an all-values-from annotation type and it is subsumed by the required concept,

4. The required concept is associated with a some-values-from annotation type, the provided concept is associated with an all-values-from annotation type and it is neither subsumed by, nor it subsumes the required concept,

5. The required concept is associated with a some-values-from annotation type, the provided concept is associated with a some-values-from annotation type and it is subsumed by the required concept.

In the above cases, the first and the second cases are preferred over the third and fourth cases, which are preferred over the last case. The preferences between these different cases is translated in the **ConceptDoM**() function by the use of coefficients τ_1, τ_2 and τ_3, where $\tau_1 < \tau_2 < \tau_3$. Figure 4.4 illustrates the different cases for calculating the degree of match between concepts employing the ontology of Figure 4.5. In the figure, coloured areas specify concepts associated with the all-values-from annotation type, while hatched areas specify concepts associated with the some-values-from annotation type. The intersection between the two areas corresponds to the set of provided concepts that satisfy a request. The first and the second cases are associated with the coefficient τ_1 because in both cases the provided concepts satisfy all the required concepts. The third and the fourth cases are associated with the coefficient τ_2 because the required concepts are satisfied by an identified subset of the provided concepts. Finally, the last case is associated with the coefficient τ_3 because the matching holds but the subset of the provided concepts that satisfy the required concepts can not be identified at service matching time.

In addition, for each situation, provided concepts that are closest to the required one in the classified ontology are preferred among the others.

More formally the **ConceptDoM**() function is defined as follows:

- If n_2.AnnotationType = $\otimes$ and n_1.AnnotationType=$\otimes$ and **Subsume**(n_1,n_2) $\Rightarrow$ **ConceptDoM**(n_1, n_2) = $\tau_1 \times |n_1.Level - n_2.Level|$
- If n_2.AnnotationType = $\oplus$ and n_1.AnnotationType=$\otimes$ and **Subsume**(n_1,n_2) $\Rightarrow$ **ConceptDoM**(n_1, n_2) = $\tau_1 \times |n_1.Level - n_2.Level|$
- If n_2.AnnotationType = $\oplus$ and n_1.AnnotationType=$\otimes$ and **Subsume**(n_2,n_1) $\Rightarrow$ **ConceptDoM**(n_1, n_2) = $\tau_2 \times |n_1.Level - n_2.Level|$
- If n_2.AnnotationType = $\oplus$ and n_1.AnnotationType=$\otimes$ and $\exists c$: **Subsume**(n_1,c) and **Subsume**(n_2,c) $\Rightarrow$ **ConceptDoM**(n_1, n_2) = $\tau_2 \times |c.Level - n_2.Level|$
- If n_2.AnnotationType = $\oplus$ and n_1.AnnotationType=$\oplus$ and **Subsume**(n_2,n_1) $\Rightarrow$ **ConceptDoM**(n_1, n_2) = $\tau_3 \times |n_1.Level - n_2.Level|$

where $n_i.Level$ specifies the level of the concept n_i in the classified ontology hierarchy.

A complete example of evaluating the semantic distance between capabilities is depicted in Figure 4.3. While provided capabilities Adv_1 and Adv_2 both match the required capability in terms of functional and non-functional properties, they do not have the same semantic distance with respect to the required capability, i.e., their semantic distance is τ_2 and $2 * \tau_3$ respectively. Functional degree of match between the capability Adv_1 and the required capability is calculated as follows:

FunctionalDoM(Adv_1,Req)=
ConceptDoM($ResourceName^{\otimes}$,$ResourceName^{\otimes}$) +
Min(**ConceptDoM**($VideoResource^{\otimes}$,$EntertainmentResource^{\oplus}$),
ConceptDoM($SoundResource^{\otimes}$,$EntertainmentResource^{\oplus}$)+
ConceptDoM($DigitalServer^{\oplus}$,$DigitalServer^{\oplus}$)

ConceptMatch($ResourceName^{\otimes}$,$ResourceName^{\otimes}$)=
$\tau_1 \times |ResourceName.Level - ResourceName.Level|$=0

ConceptDoM($VideoResource^{\otimes}$,$EntertainmentResource^{\oplus}$) =
$\tau_2 \times |VideoResource.Level - EntertainmentResource.Level|$=

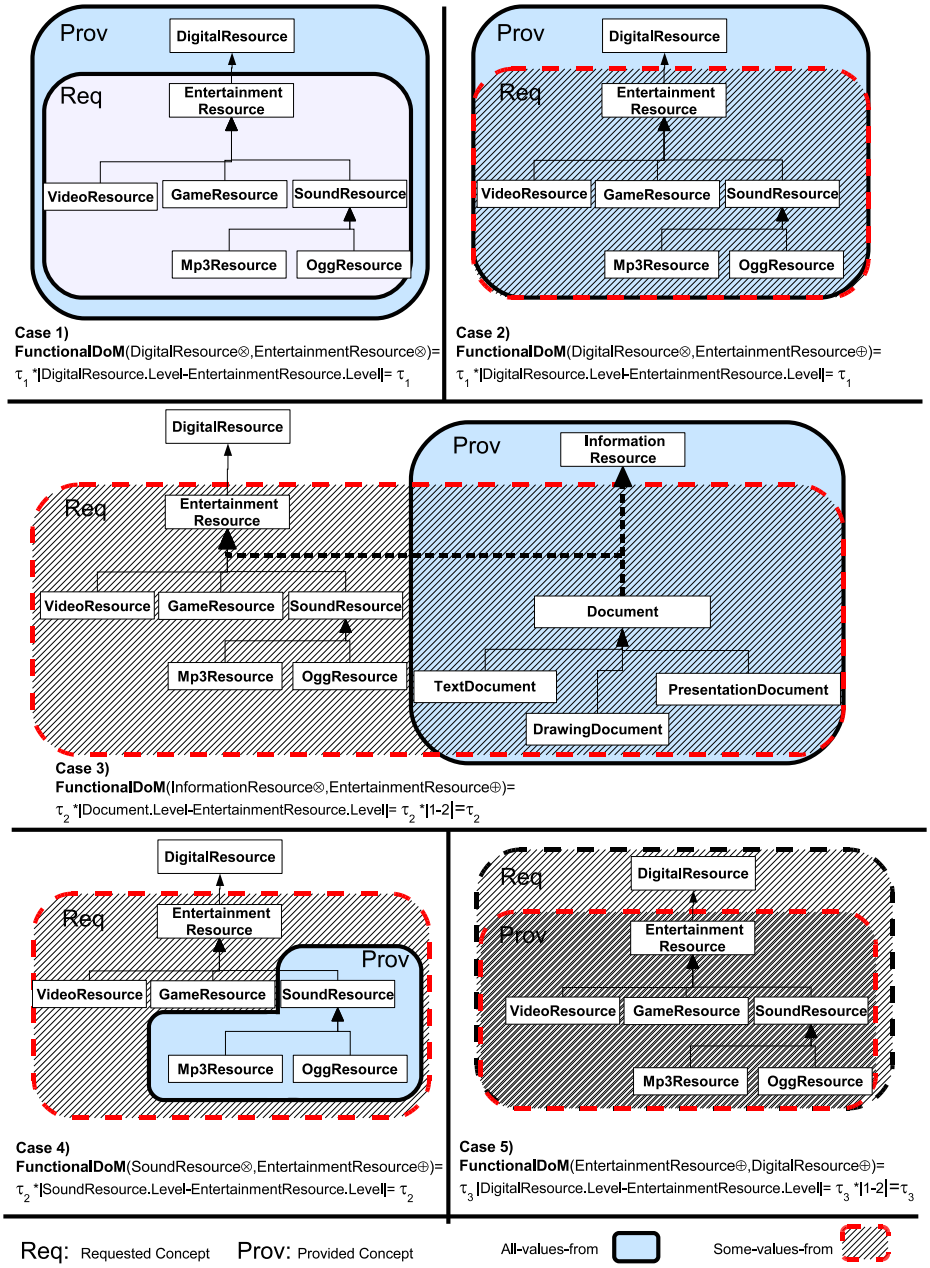


Figure 4.4: Degree of Match Between Concepts

τ_2

$\mathbf{ConceptDoM}(SoundResource^{\otimes}, EntertainmentResource^{\oplus}) =$

$\tau_2 \times |SoundResource.Level - EntertainmentResource.Level|=$
τ_2

Thus:
Min(**ConceptDoM**($VideoResource^{\otimes}$,$EntertainmentResource^{\oplus}$),
ConceptDoM($SoundResource^{\otimes}$,$EntertainmentResource^{\oplus}$))=$\tau_2$

Finally:
ConceptDoM($DigitalServer^{\oplus}$,$DigitalServer^{\oplus}$) =0

Hence,

FunctionalDoM(Adv_1,Req)= $0 + \tau_2 + 0 = \tau_2$

Non-Functional Degree of Match Between Capabilities

The degree of match between non-functional properties of capabilities is evaluated using the function **PropertiesDoM**() as follows:

$$\textbf{PropertiesDoM}(Adv,\ Req) = \sum_{i=1}^{n} w_i * p_i \tag{4.1}$$

where, n is the number of non-functional properties of Req, w_i is the relative importance of the considered property, i.e., the lower the weight w_i assigned to the property p_i is, compared to the weights assigned to the other properties, the more p_i is preferred in relation to other properties. This allows a service requester to specify priorities between non-functional properties. For instance, a service requester may prefer using a service that ensures a higher security level even if this service has higher latency than other services. In this case, the weight given to the property *Security* should be lower than the weight given to the property *Latency*.

Since properties are heterogeneous − i.e., some are qualitative, some are quantitative and further expressed in different units − data normalization is needed in order to evaluate

the **PropertiesDoM**(). The first normalization that we introduce is assigning numeric values to qualitative properties such that they can participate in the **PropertiesDoM**() function. These values are given by the function **ConceptDoM**() defined earlier. This allows evaluating a provided qualitative property with respect to a required property. Indeed, the smaller the **ConceptDoM**() between a provided qualitative property and a required one is, the better. The second normalization that we apply is the standard deviation normalization on the various properties as in [Liu, 2006]. This normalization is performed as follows:

Properties that are stronger with greater values (e.g., availability) are normalized according to the following equation:

$$p'(adv_i) = \begin{cases} 0 & \text{if } (p(adv_i) - m(p) > 2 * \Delta(p)) \\ 1 & \text{if } (p(adv_i) - m(p) < -2 * \Delta(p)) \\ 0.5 - \frac{p(adv_i) - m(p)}{4 * \Delta(p)} & \text{otherwise} \end{cases} \tag{4.2}$$

While properties that are stronger with smaller values (e.g., latency, normalized qualitative properties), are normalized according to the following equation (so that smaller values contribute more to the **PropertiesDoM**() function):

$$p'(adv_i) = \begin{cases} 1 & \text{if } (p(adv_i) - m(p) > 2 * \Delta(p)) \\ 0 & \text{if } (p(adv_i) - m(p) < -2 * \Delta(p)) \\ \frac{p(adv_i) - m(p)}{4 * \Delta(p)} + 0.5 & \text{otherwise} \end{cases} \tag{4.3}$$

where $p(adv_i)$ is the value of property p for the provided capability adv_i, and $m(p)$ and $\Delta(p)$ are the mean value and standard deviation for the property p, respectively.

Figure 4.3 describes an example of evaluating the degree of match between non-functional properties of capabilities. First, as these properties are heterogeneous, qualitative properties are normalized to numeric values using their **ConceptDoM**(). Results are given in columns *Adv1'* and *Adv2'* of the table for the capabilities Adv_1 and Adv_2 respectively. Using these values, we normalize all the properties using the standard deviation normalization. Results are given in columns Adv_1'' and Adv_2''. Having all the values normalized, it is easy to evaluate the **PropertiesDoM**() for each provided capability as follows:

$$\textbf{PropertiesDoM}(Adv_1,\ Req) = 0.75 * w_1 + 0.25 * w_2 + 0.25 * w_3 + 0.25 * w_4,$$
$$\textbf{PropertiesDoM}(Adv_2,\ Req) = 0.25 * w_1 + 0.75 * w_2 + 0.75 * w_3 + 0.75 * w_4$$

where w_1, w_2, w_3 and w_4 are the weights attributed to each of the properties *Price*, *Latency*, *Availability* and $Network$, respectively. Assuming that they all have the same relative importance, i.e., $w_i = 1$, the first capability has a better semantic distance in terms of non-functional properties, i.e., **PropertiesDoM**(Adv_1, *Req*)= 1.5 < **PropertiesDoM**(Adv_2, *Req*)= 2.5.

The set of conformance relations presented in this section are used by our semantic service registry to efficiently publish and locate provided and required service capabilities, respectively. Thus, these relations have to be performed efficiently to fit the requirements of resource constrained devices on which our registry may be deployed as presented in the following section.

4.2.2 Efficient Semantic Service Matching

In order to assess the conformance between two capabilities our registry has to perform semantic reasoning on ontologies. Indeed, the **FunctionalCapabilityMatch**() relation uses the relation **Subsumes**() to assess the subsumption relation between two concepts. However, assessing subsumption between concepts is a costly operation that cannot be employed on resource constrained devices without appropriate optimizations.

In order to deal with this issue our registry employs two complementary mechanisms :

1. Ontologies are classified offline, i.e., not at service matching time, and the resulting classified ontologies are encoded using an ontology encoding algorithm.

2. Each concept used to annotate an element in a service or a request description, (i.e., input, output, category and non-functional properties) is given with the triple <Ontology,Code,Version> where *Ontology* is a unique identifier of the ontology, *Code* is the code corresponding to the entity being annotated and *Version* is the version of

the code. The information regarding the version of the code is used to ensure consistency of codes in the face of the dynamics and evolution of ontologies. The resulting service or request description that contains the triples <Ontology,Code,Version> for each semantic annotation is said to be *pre-encoded*.

In using these measures, we assume that service advertisements and requests are pre-encoded when a service location request is processed and specifically when the matching between capabilities is performed. This assumption can be supported by various scenarios. For instance, the service developer may use a tool for semantically annotating the service description and for automatically encoding the employed semantic concepts. Specifically, such tool should maintain a local repository of ontologies and as soon as ontologies are added to the local repository, the tool classifies and encodes them following an appropriate ontology encoding algorithm. As ontology classification and encoding need to be performed only once, it is not necessary to repeat these actions at service matching time. Further, each time a service developer selects a concept from an ontology to annotate a service element in a service description, the code corresponding to that element can be automatically inserted into the service description. In Chapter 6, we present such tool for generating pre-encoded semantic service descriptions.

If the service developer does not use such tool to annotate services, and consequently the resulting service descriptions and service requests are not pre-encoded, other scenarios are still conceivable. For instance, a service for encoding service descriptions may be provided in the pervasive computing environment. This service takes a non-encoded semantic service description and generates a description where each semantic concept is associated with its corresponding code in the classified ontology. This scenario is enabled thanks to the fact that the ontology encoding process does not need to be performed in a centralized way by a single entity. Indeed, one of the major requirements for the ontology encoding algorithm (discussed below) is its execution determinism, i.e., taking an ontology O as input, for any execution of the algorithm, the generated encoded ontology given as output, should always be the same.

Finally, if a service encoding facility is not available in the environment, the service

registry may implement the encoding algorithm and encode itself service advertisements as soon as a provider advertises a service. While this last solution implies an additional overhead when inserting a service in the service registry service location can still be efficient if service requests are pre-encoded.

Encoding Classified Ontologies

As mentioned already in Section 4.1, encoding classified ontologies can be done using algorithms developed for other related problem areas, such as the encoding of class hierarchies in object-oriented programming languages.

Nevertheless, the encoding algorithm used to encode the classified ontology should have the following properties:

1. The encoding algorithm should be deterministic, i.e., for any execution of the algorithm on a classified ontology, the algorithm should always give the same code to each concept. This allows ontology encoding to be performed in a distributed way.

2. There should be a function to infer if the matching holds between two concepts by only comparing the concepts' codes without looking neither at the original ontology nor at the classified ontology. Specifically, due to the multiple inheritance structure of ontologies and for being able to deal with our introduced annotation types, this function should be able to identify:

   - Subsumption between two concepts by comparing their codes,
   - Intersection between the sub-hierarchies of two concepts by comparing their codes.

3. The encoding algorithm should support conflict-free encoding for large ontologies

For instance, the encoding that consists of using a n $\times$ n binary matrix (where n is the number of concepts in the original ontology) with a 1 on position (i,j) if the concept i is an ancestor of the concept j in the ontology is not an appropriate encoding algorithm. Indeed, it does not fulfil the second requirement from the above requirements, since to infer

the subsumption between two concepts we need to have the whole matrix representing the encoded ontology instead of having just the codes associated with the corresponding concepts. Three other encoding techniques have been investigated in the literature and surveyed in [Ben Mokhtar et al., 2007b, Preuveneers and Berbers, 2006]:

- **Bit-vector based encoding** (e.g., [Caseau, 1993, Krall et al., 1997, Ait-Kaci et al., 1989, van Bommel and Beck, 1999]). These solutions aim at assigning a bit-vector to each concept of the hierarchy, minimizing the number of bits being employed. However these solutions require the re-encoding of conflicting codes whenever a subsumption check results in a false positive. Thus, these solutions are inappropriate for encoding ontologies as they restrict ontology evolution.

- **Interval based encoding** (e.g., [Zibin and Gil, 2001, Constantinescu and Faltings, 2003, Agrawal et al., 1989]). In this category of encoding algorithms, a concept in an ontology is associated with an interval. This interval is then divided into sub-intervals to encode its child concepts. Using this encoding technique, the subsumption assessment between concepts translates to an interval inclusion assessment, which can be performed efficiently (numeric comparison of the higher and lower interval limits) without looking at the whole encoded hierarchy. Furthermore, this encoding technique supports conflict free encoding. Indeed, instead of dividing an interval into subintervals of the same size, which would lead to limited scalability of the encoding algorithm, Constantinescu *et al.* define a linear inverse exponential function, $linKinvexpP(x) = \frac{1}{p^{int(\frac{x}{k})}} + (x \bmod k) * \frac{1}{k} * \frac{1}{p^{int(\frac{x}{k})}}$, where p and k are two parameters to be fixed. Using this function, each time a child concept is added into the ontology its interval is smaller than its brother concepts, which enables a better scalability with respect to ontology evolution. Finally, this solution supports conflict-free encoding, as intervals encoding brother concepts are completely independent from each other.

- **Prime number based encoding** (e.g., [Preuveneers and Berbers, 2006]). This solution uses prime numbers to encode classified ontologies by assigning to each concept a code calculated by the multiplication of its parent's code and a new prime

number that have not been associated to any other node in the hierarchy. The function used to control subsumption between two concepts consists in performing an integer division of the greater code by the smaller one. If the remainder of the division is equal to zero, the concept with the smallest code subsumes the other concept. This algorithm also supports conflict free encoding due to the unlimited number of prime numbers.

Both the second and the third encoding algorithms satisfy all the requirements identified above except the identification of the intersection between the sub-hierarchies of two concepts. This is due to the fact that both these encoding algorithms start by encoding the ontology hierarchies from root nodes, which allows to identify the common ancestors of two nodes but not the common sub-hierarchies. A solution to this is to use the same encoding principles and encode hierarchies by starting from their leaf nodes. This allows a node to have the knowledge of its sub-hierarchy.

Figure 4.5 shows an example of encoding the classified hierarchy of the resource ontology using prime number-based encoding and starting from leaf nodes. Using this encoding algorithm, matching two concepts is carried out as follows:

1. Subsumption between two concepts represented by their respective codes is checked by performing the division of the greater code by the smaller one. If the remainder of the division is 0, the concept having the greater code subsumes the second one. Else, there is no subsumption relation between the two concepts.

2. The existence of a common sub-hierarchy between two concepts is assessed when a common divisor between the codes of the two concepts is found. This can done by performing successive divisions of the two codes by prime numbers until a common divisor is found. Yet, the maximum number of divisions to be performed for finding a common divisor is limited by the number of leaf nodes of the classified ontology hierarchy.

We recall here that among the five cases of matching semantic concepts identified in Section 4.2.1 according to the annotation types associated with the concepts, there are four

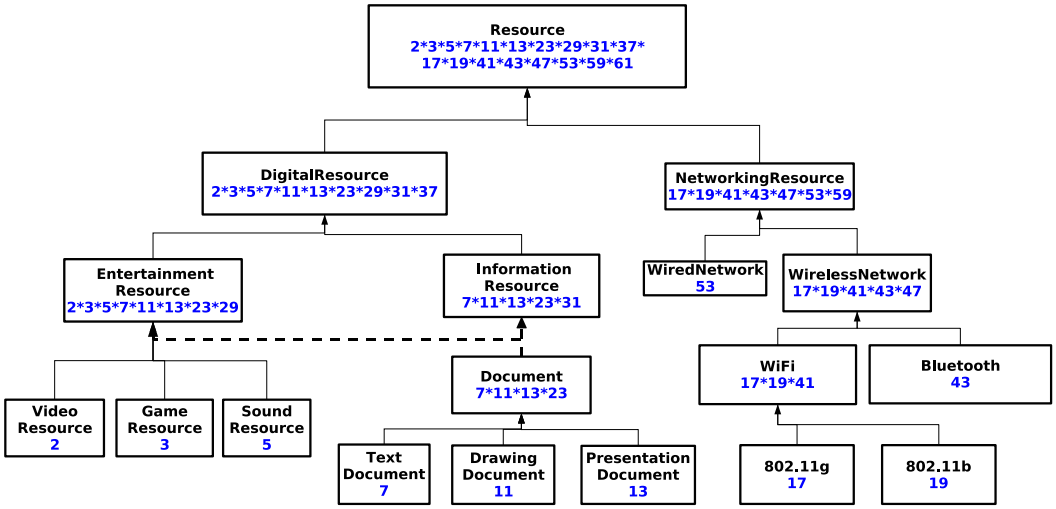


Figure 4.5: Encoded Resource Ontology

cases where the matching is assessed through subsumption check and only a single case where matching is assessed through intersection check between concept sub-hierarchies. Hence, using this encoding technique, the costly semantic reasoning on ontologies translates to a numeric comparison of codes performed through a single, or at most, a number of arithmetic divisions.

A number of heuristics for minimizing code lengths have been presented in [Preuveneers and Berbers, 2006]. An evaluation of this encoding algorithm in terms of the generated code lengths is further presented in Chapter 6.

4.2.3 Registry Service Index

In our registry, service advertisements are classified into graphs of "similar" capabilities for efficiently inserting and retrieving capabilities into/from the registry. Then, publication of a service and location of a service require to perform capability matching with nodes of those graphs. In order to reduce the number of graphs with which capability matching is performed, the registry maintains an index table. This table gives for each ontology used in the registry, the set of graphs that contain capabilities that reference this ontology. This table allows preselecting a set of graphs that are more likely to contain the capability

to be retrieved from (resp. to be inserted in) the registry, i.e., the graphs that use the same ontologies as the latter capability. Specifically, in order to locate which graphs use the same ontologies as a required (resp. provided) capability, we retrieve from the index table the sets of graphs referencing each ontology used by the required (resp. provided) capability. The intersection between all these sets of graphs gives the graphs that use at least all the ontologies referenced by the required (resp. provided) capability.

Figure 4.6 gives an example of the index table maintained by our registry. This figure illustrates the fact that a graph may use different ontologies (e.g., Graph 4 uses both the *Food* and *Wine* ontologies). Furthermore, different graphs may use the same ontologies (e.g., both Graph 2 and Graph 3 use the *Accommodation* ontology). If a required capability uses the *Accommodation* and *Business* ontologies, we perform the intersection between the set of graphs using the *Accommodation* ontology and those using the *Business* ontology, and the result of this intersection is a set containing Graph 2, i.e., $\{Graph2, Graph3\} \cap \{Graph1, Graph2\} = \{Graph2\}$. This means that Graph 2 is likely to contain capabilities that match the required capability and that all the other graphs do not contain any capability that match the sought capability.

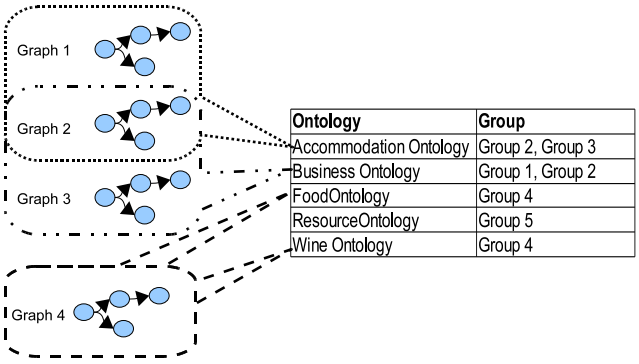


Figure 4.6: Indexing Graphs of Related Capabilities

4.2.4 Service Publication

In order to organize the service registry, service publication constructs directed acyclic graphs (DAGs) of capabilities provided by the advertised services. More formally, a graph

G of capabilities is defined by a set of *Nodes* constituting the graph and referred to by the notation **Nodes**(G), and a function **Successors**() that gives for each node of G the set of its direct successors in the graph. A node $N \in$ **Nodes**(G) contains a set of service capabilities, referred to by the notation **Capabilities**(N). The function **Successors**() is defined as follows:

$$\textbf{Successors}: \textbf{Nodes}(G) \longrightarrow 2^{\textbf{Nodes}(G)}$$
$$n \in \textbf{Nodes}(G) \longmapsto \textbf{Successors}(n) \subset \textbf{Nodes}(G)$$

Based on the function **Successors**(), we define the function **Predecessors**() as follows:

let $< x, y > \in \textbf{Nodes}(G)^2$ and two nodes in the graph G:
$y \in \textbf{Predecessors}(x) \Leftrightarrow x \in \textbf{Successors}(y)$

Using these two functions, we define the set **Roots**(G) $\subset$ **Nodes**(G), which is the set of nodes from G that do not have predecessors in G. Respectively, we define the set **Leaves**(G) $\subset$ **Nodes**(G), which is the set of nodes from G that do not have successors in G. More formally:

$x \in \textbf{Roots}(G) \Leftrightarrow \textbf{Predecessors}(x) = \emptyset$
$x \in \textbf{Leaves}(G) \Leftrightarrow \textbf{Successors}(x) = \emptyset$

The set of nodes that contains all the successors of a node N of a graph G and recursively all their successors is named the subgraph of N and is noted **SubGraph**(N). More formally this set is defined as follows:

if $N \in \textbf{Leaves}(G) \Rightarrow \textbf{SubGraph}(N) = \emptyset$
else $\forall x \in \textbf{Successors}(N) : \textbf{SubGraph}(N) = \textbf{SubGraph}(N) \cup \textbf{SubGraph}(x)$

Similarly, we define the set of all the predecessors of a node N and recursively their predecessors in a graph G by the set parent graph noted **ParentGraph**(N). More formally,

this set is defined as follows:

if $N \in \mathbf{Roots}(G) \Rightarrow \mathbf{ParentGraph}(N) = \emptyset$
else $\forall x \in \mathbf{Predecessors}(N) : \mathbf{ParentGraph}(N) = \mathbf{ParentGraph}(N) \cup \mathbf{ParentGraph}(x)$

The relation used to build the graphs is the **FunctionalCapabilityMatch**() relation. Grouping of capabilities in a graph is based on the two following principles:

1. If **FunctionalCapabilityMatch**(c_1,c_2) holds and **FunctionalCapabilityMatch**(c_2,c_1) holds between two capabilities c_1 and c_2, then c_1 and c_2 will be stored in a single node N of the graph G as these capabilities are equivalent to each other in terms of functional properties. More formally:

$$(\{c_1, c_2\} \in \mathbf{Capabilities}(N)^2) \Leftrightarrow (\mathbf{FunctionalCapabilityMatch}(c_1,c_2) \wedge \mathbf{FunctionalCapabilityMatch}(c_2,c_1)) \Leftrightarrow (\mathbf{FunctionalDoM}(c_1,c_2) = \mathbf{FunctionalDoM}(c_2,c_1) = 0)$$

2. If **FunctionalCapabilityMatch**(c_1,c_2) holds and **FunctionalCapabilityMatch**(c_2,c_1) does not hold, the capability c_1 will be stored in a node N_1 and the capability c_2 will be stored in a different node N_2 such that there exist a directed path from N_1 to N_2 in the graph. More formally:

$$(N_1 \in \mathbf{ParentGraph}(N_2)) \Leftrightarrow (\mathbf{FunctionalCapabilityMatch}(c_1,c_2) \wedge \neg\ \mathbf{FunctionalCapabilityMatch}(c_2,c_1))$$

As capabilities contained in the same node N of a graph are semantically equivalent, the comparison of a new capability with capabilities of a node using the **FunctionalCapabilityMatch**() relation can be done by comparing the new capability with any of the

capabilities of that node. We note by **Capability**(N) one of the capabilities contained in the node N.

Using this grouping principles, the most *generic* capabilities of a particular domain will be stored in root nodes of graphs, while the most *specific* capabilities will be stored in leaf nodes of graphs. The set of root nodes of a graph G_i, i.e., **Roots**(G_i), contains capabilities that are said to be more generic than other capabilities of the graph because they provide generic outputs that may match a larger number of concepts compared with the capabilities contained in the rest of the graph and require inputs that may be matched with a larger number of concepts compared with the capabilities contained in the rest of the graph. On the contrary, leaf nodes of a graph G_i, i.e., **Leaves**(G_i), contain capabilities that are said to be more specific than the other capabilities of the graph because they provide specific outputs that may be harder to match compared with the capabilities contained in the rest of the graph and require inputs that may be harder to match with inputs provided by required capabilities compared to the capabilities contained in the same graph.

When a new service is registered with the registry, the set of capabilities that it provides are classified among the existing graphs. The algorithm of classifying new capabilities is based on the following two properties:

Prop 1 : $\neg$ **FunctionalCapabilityMatch**(***Capability***$(Root_i)$, Adv): $\text{Root}_i \in$ **Roots**(G) $\Rightarrow$
$\forall$ N $\in$ **SubGraph**($Root_i$): $\neg$ **FunctionalCapabilityMatch**(***Capability***(N), Adv)

Prop 2 : $\neg$ **FunctionalCapabilityMatch**(Adv, ***Capability***$(Leaf_i)$): $\text{Leaf}_i \in$ **Leaves**(G) $\Rightarrow$
$\forall$ N $\in$ **ParentGraph**($Leaf_i$): $\neg$ **FunctionalCapabilityMatch**(Adv, ***Capability***(N))

The proofs of these properties are given in Appendix A. These properties are used to check whether a provided capability Adv will be inserted in a graph G_i without having to assess the **FunctionalCapabilityMatch**() relation with all the capabilities of that graph. Specifically, using Property [Prop 1], if the matching holds between a capability Adv and a capability of a root node of a graph G_i, we can infer that this capability will be inserted in a node of the graph that has a predecessor in G_i. Indeed, Property [Prop 1] expresses that

if the **FunctionalCapabilityMatch**() relation between a capability of a root node $Root_i$ of a graph G_i and the provided capability does not hold then, this relation will neither hold between Adv and any capability contained in nodes from the sub-graph of $Root_i$. Thus, thanks to this property, it is not necessary to assess the **FunctionalCapabilityMatch**() relation between all the capabilities of the graph and Adv.

Respectively, using Property [Prop 2], if the matching holds between a capability Adv and a leaf node of a graph G_i, we can infer that this capability will be inserted in a node of the graph that has a successor in G_i. Indeed, if a matching holds between Adv and a capability c contained in a leaf node $Leaf_i$ of G_i, this means that Adv is more generic than c. Thus, the node that will contain Adv will be in the parent graph of $Leaf_i$ in G_i. On the contrary, if the matching between Adv and c does not hold, using Property [Prop 2] we can infer that Adv will not be the predecessor of any of the successors of the node $Leaf_i$ in the graph G_i, without assessing the relation **FunctionalCapabilityMatch**() with other capabilities contained in the parent graph of $Leaf_i$.

The detailed algorithm for inserting a service advertisement in the registry performed by the service publication functionality is given in Algorithm 1:

Figure 4.7 gives an example of inserting a capability into a graph. This graph contains capabilities that use the resource ontology depicted in Figure 4.2 (p. 60). The most generic capability, i.e., the *Airport Digital Streaming Capability*, is contained in the root node of the graph, while the most specific capabilities are contained in leaf nodes, e.g., *My MP3 Server* capability. In this example, the capability to be inserted into the registry is the *Stephan Music Server Capability* or *StephanC* for short. The graph depicted in the figure is the only graph that has been pre-selected, because it is the only graph that is indexed by the *Resource* ontology. Using the algorithm defined above the first step for inserting the new capability (after graph selection) is to check whether the capability *StephanC* will have a predecessor in the graph by evaluating the **FunctionalCapabilityMatch**() between a capability from the unique root node of the graph, i.e., *Airport Digital Streaming Server* capability or *AirportC* for short, and *StephanC*. As the matching holds, *StephanC* will have

Algorithm 1 InsertService(in: serviceDescription, $G_{1..m}$, **out:** $G'_{1..k}$)

```
1: for each Capability c_i in serviceDescription do
2:   for each Graph G_i using the same ontologies as c_i do
3:     //Find Predecessors of c_i in G_i
4:     for each Node Root_i in Roots(G_i) do
5:       if FunctionalCapabilityMatch(Capability(Root_i), c_i) then
6:         Check with N_i ∈ SubGraph(Root_i)
7:         until ¬ FunctionalCapabilityMatch(Successor(N_i), c_i)
8:         Draw an edge from N_i to c_i
9:       end if
10:    end for
11:    //Find Successors of c_i in G_i
12:    for each Leaf_i in Leaves(G_i) do
13:      if FunctionalCapabilityMatch(c_i, Capability(Leaf_i)) then
14:        Check with N_i ∈ ParentGraph(Leaf_i)
15:        until ¬ FunctionalCapabilityMatch(c_i, Predecessor(N_i))
16:        Draw an edge from c_i to N_i
17:      end if
18:    end for
19:  end for
20: end for
```

a predecessor in the graph. The next step is to match with the successors of *AirportC*, i.e., *CarlaC* and *RozalieC* to find that predecessor. However, as the matching with both these capabilities does not hold, it is not necessary to go further in the successors of these nodes; *AirportC* should be the predecessor of *StephanC*. The second part of the algorithm consists of finding successors of *StephanC* in the graph by matching with leaf nodes of the graph, i.e., *MyMp3C*, *OggStreamingC* and *CarlaC*. As the matching fails with *CarlaC* this means that neither this capability, nor any of its predecessors will be a successor of *StephanC*. On the contrary, as the matching holds with both *MyMp3C*, *OggStreamingC*, we continue the matching with the predecessors of this node until the matching fails, i.e., with the *AirportC*. Thus the successor of *StephanC* should be the node containing the *RozalieC* capability. Finally, the edge between the node containing the *AirportC* capability and the node containing the *RozalieC* capability is removed.

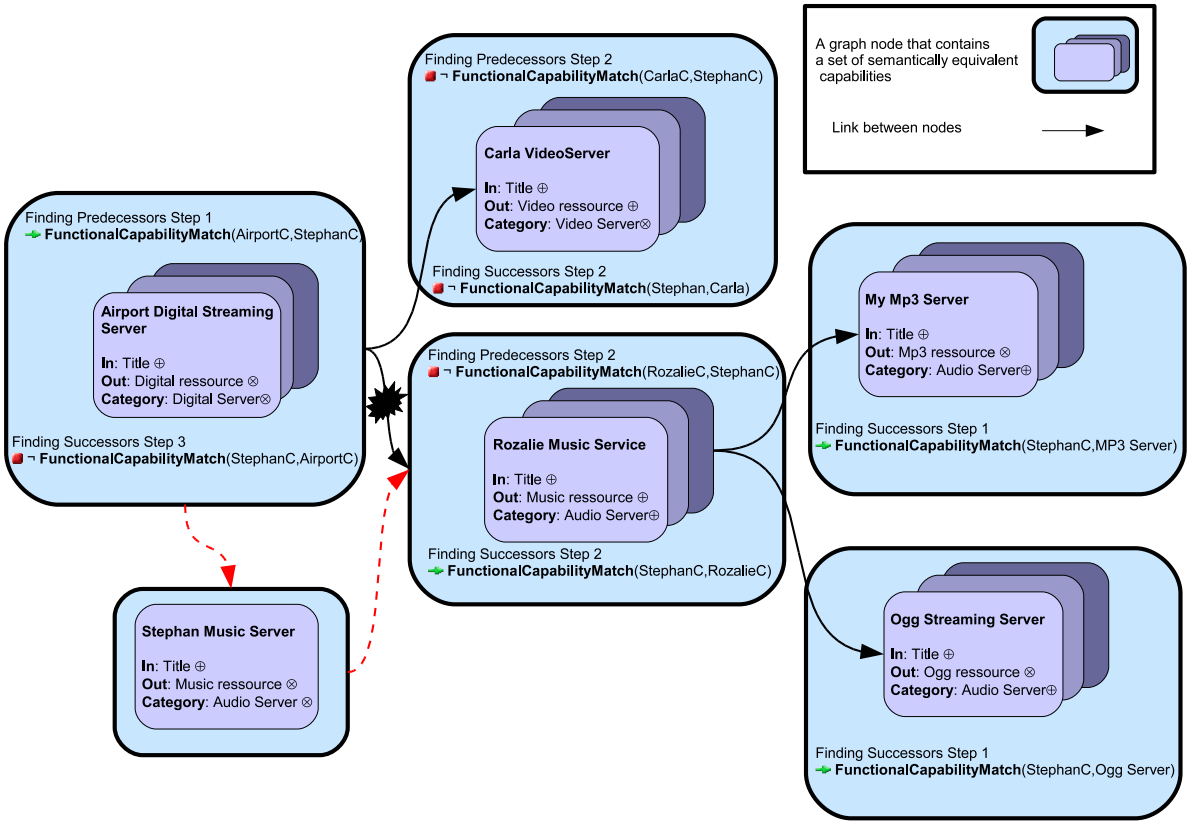


Figure 4.7: Service Publication Example

4.2.5 Service Location

Service location is responsible for efficiently answering service requests. For each capability described in the service request, service location first preselects the graphs that are most likely to match that capability, i.e., the graphs that use the same ontologies as the required capability (See Section 4.2.3). Then among these graphs the service location functionality filters out the graphs that will not contain the sought capability by using Property [Prop 1] defined above. Specifically, those graphs for which **FunctionalCapabilityMatch**() does not hold between capabilities contained in their root nodes and the required capability will be filtered out. This allows efficiently locating the graph that contains capabilities that match the required capability. Indeed, if **FunctionalCapabilityMatch**() holds between a capability contained in a root node of a graph and the required capability, the capability

Algorithm 2 MatchService(in: serviceDescription, $G_{1..m}$, **(out:** capabilitySet)

```
1: for each c_i in serviceDescription do
2:   for each G_i using the same ontologies as c_i do
3:     while not c_i matched do
4:       for each Root_k in Roots(G_i) do
5:         if FunctionalCapabilityMatch(Root_i, c_i) then
6:           Add M to capabilitySet with M in SubGraph(Root_i)
7:         end if
8:       end for
9:     end while
10:  end for
11:  Select from capabilitySet the capability M such that:
12:  FunctionalDoM(M, c_i) is minimal and
13:  PropertiesDoM(M, c_i) is minimal
14: end for
```

that best matches the request in terms of functional and non-functional properties is contained in the sub graph of that node. To locate that capability, we first estimate the **FunctionalCapabilityMatch**() between nodes in the subgraph of the latter root node and the required capability. Once that node is located, the selection between its capabilities is based on non-functional properties by using the **PropertiesDoM**() relation (Section 4.2.1).

The algorithm performed by the service location functionality is described in Algorithm 2:

In this solution, capabilities that best match the required capability in terms of functional properties are selected. Then, among these semantically equivalent capabilities the one that best conforms to the request in terms of non-functional properties is chosen. A more flexible but more costly way for selecting the capability that best conforms to the request, where the preference between functional and non-functional features is specified by the application, could be performed by using the customizable degree of match function as defined in Section 4.2.1:

$$\textbf{CapabilityDoM}(c_1,c_2) = \lambda_1 \times \textbf{FunctionalDoM}() + \lambda_2 \times \textbf{PropertiesDoM}()$$

where λ_1 and λ_2 specify the preference among functional and non-functional properties

of a capability. In this case, this relation has to be performed for all the capabilities of the subgraph of the root node for which the matching holds with the required capability.

4.3 Assessing the Efficiency of the Semantic Service Registry

As the main function of both service publication and location algorithms is parsing a set DAGs and performing matches on the capabilities of the visited nodes its complexity can be approximated with the complexity of elementary graph algorithms (e.g., the breadth-first search algorithm whose complexity is linear in the size of the graph). Furthermore, the processing done in each node for matching capabilities is composed of a set of divisions for assessing subsumption between concepts and is thus linear in the number of concepts being compared (i.e., inputs, outputs, category of provided and required capabilities).

Thus, compared to existing research efforts that investigate efficient semantic service discovery ([Constantinescu and Faltings, 2003], [Srinivasan et al., 2004]) our solution performs better as we achieve efficiency for both service publication and service location while existing solutions overload service publication to achieve efficiency at service location time.

We complement this theoretical assessment with a practical assessment through the performance evaluation of our efficient semantic service registry in Chapter 6.

4.4 Concluding Remarks

We presented in this chapter an efficient semantic service registry for pervasive computing environments. This registry supports the publication, location and matching of heterogeneous service descriptions enabling multi-language interoperability. This registry supports a set of conformance relations for matching both syntactic and rich semantic service descriptions as well as their heterogeneous non-functional properties. These conformance relations also identify the degree of conformance between service descriptions, and rate services with respect to their suitability for a specific service request, so that selection can be made among them.

A theoretical assessment of the service matching, publication and location algorithms

validates the efficiency of our registry. Indeed, thanks to an appropriate ontology encoding algorithm, which translates the costly semantic reasoning on ontologies to a numeric comparison of codes and to the organizing of semantic service descriptions, our service registry achieves efficient both service publication and location contrary to existing efficient semantic registries that opt for overloading the service publication phase to achieve efficiency at service location. A practical assessment through the performance evaluation of the various algorithms performed by our registry is further presented in Chapter 6. Our semantic service registry can be centralized, semi-distributed or fully distributed according to the deployment policy of our middleware. A semi-distributed deployment scheme coming from the MUSDAC platform is discussed in Chapter 6.

Chapter 5

Service Composition in Pervasive Computing Environments

Although there is value in accessing a single service, the greater value is clearly derived through enabling a flexible composition of services [Singh and Huhns, 2005]. In pervasive computing environments, service composition can be a major enabler for the user-centrism paradigm by enabling the user to be at the heart of the realization of his/her daily tasks through the integration of relevant pervasive services available in the vicinity. In service-oriented pervasive computing, user tasks can be represented as abstract composite services with an associated conversation to be realized by dynamically integrating pervasive services available at the specific time and location. Provided as a functionality of a SOM, service composition builds upon other SOM functionalities. Specifically, service composition uses service location to dynamically discover relevant pervasive services. It further uses service access for the interaction with pervasive services taking part in the resulting composition. Finally, it may use service publication for advertising the composite service as a new pervasive service.

To fit the requirements of pervasive computing environments, service composition has to deal with a number of challenging issues. First, service composition has to consider heterogeneity of pervasive services, i.e., syntactic/semantic services, with/without associated conversations. In the case of services with associated conversations, service composition

has to be performed so that data and ordering constraints of services and tasks are fulfilled. Additionally, service composition has to assess the fulfilment of user tasks required non-functional properties from the aggregation of non-functional properties provided by the composed services. Finally, appropriate composition algorithms enabling on the one hand efficiency, to fit the resource constraints of thin devices, and on the other hand flexibility, to allow the user to benefit from the diversity of services available in the vicinity, are required. To deal with this trade-off between efficiency and flexibility, the middleware should adapt the flexibility of the composition algorithm according to the available resources of the devices on which the composition is carried out.

After an analysis of the related work in service composition in Section 5.1, we present in this chapter our solution to service composition in pervasive computing environments that deals with the above requirements. This solution takes the form of a set of middleware functionalities. A service discovery client, presented in Section 5.3, uses the middleware functionalities presented in Chapter 4 for pre-selecting a set pervasive services candidate for the composition. Service conformance, presented in Section 5.4, filters out from the pre-selected services those that do not conform to the data and ordering constraints of the user task. Service coordination, presented in Section 5.5, reconstitutes the user task conversation by integrating the selected services' conversations. Finally, QoS-aware composition, presented in Section 5.6 assesses the fulfilment of the global QoS requirements of the user task. We conclude this chapter by an assessment of the efficiency of our proposed solutions in Section 5.8 and a set of concluding remarks in Section 5.9.

5.1 Service Composition: State Of The Art

A large number of solutions for service composition have been proposed in the literature during the last decade. These solutions can be classified in two main categories depending on whether the composition is carried out based on the service interfaces, i.e., interface-based service composition or based on service conversations, i.e., conversation-based service composition. Interface-based service composition assumes services described as a list of independent capabilities, without associated conversations, while conversation-based service

composition assume services described with associated conversations. In both categories, user tasks may be specified with or without an associated conversation.

5.1.1 Interface-Based Service Composition

Approaches to interface-based service composition are represented in the second column of Figure 5.1. This type of composition decomposes in two cases according to whether the task is specified with or without an associated conversation, i.e., service chaining algorithms and conversation-driven service selection algorithms. Service chaining, including *forward chaining* and *backward chaining*, is used when both networked services and the target user task are described as individual capabilities without associated conversations. In these composition models, individual service capabilities are combined with each other based on the conformance of their signatures. The objective of this combination is to obtain a composite service that conforms to the signature specification of the target user task. Forward chaining starts by selecting services that match the task's provided inputs (and preconditions) and chains services forward based on their signature compatibility until all the task's required outputs (and effects) are generated. On the contrary, backward chaining starts by selecting services that generate the task's required outputs (and effects) and chains services backward until all the inputs (and preconditions) of the selected services can be satisfied by the task's provided inputs (and preconditions). A number of research proposals adopt these composition models [Ramasamy, 2006, Masuoka et al., 2003]. While this approach allows combining services without any previous knowledge about how services should be chained, its complexity is high as all the possible chaining schemes need to be investigated. Furthermore, as the chaining process is "blind" (i.e., capabilities are chained only on the basis of the compatibility of their signatures), unexpected capabilities may be employed, which generates uncertainty regarding how user's information is manipulated. Some approaches improve this solution by providing task decomposition rules in order to orient the service chaining process [Dan et al., 2003].

Conversation-driven service selection assumes user tasks described with an associated conversation and services described as independent capabilities. This model has been

often employed for dynamic service composition in pervasive computing environments [Chakraborty et al., 2005, Aggarwal et al., 2004, Benatallah et al., 2003]. In this model, provided service capabilities are matched against capabilities required in the target user task. The various approaches that follow this model differ from each other according to the expressiveness of the supported service description language and its associated matching algorithm. As this approach follows a user task conversation specification, the resulting composition meets the user's requirements without unobtrusively using the user provided information through the employment of unexpected capabilities. However, this composition model does not consider the behaviour of services when integrating them, which does not guarantee correct composition of services.

5.1.2 Conversation-Based Service Composition

Conversation-based service composition assumes that services to be combined have a complex behaviour. This category of composition algorithms is represented by the third column of Figure 5.1. It is divided in three different cases, i.e., *goal-driven conversation selection* and *goal-driven conversation integration* where the user task is specified without an associated conversation, and *conversation-driven conversation integration* where both services and tasks are specified with associated conversations.

Goal-driven conversation selection allows the selection of a service conversation that satisfies a user task specified as a single required capability as proposed in [Bernstein and Klein, 2002]. In this approach a process query language, i.e., PQL, is employed to find service conversations that contain a fragment that satisfies the user task. Thus, it is implicitly assumed that the user's request can be performed by a single service as opposed to integrating multiple service conversations.

On the contrary, goal-driven conversation integration [Brogi and Popescu, 2005], aims at integrating a set of service conversations to realize a user task described as a single required capability. In this composition model, the conversations of a set of preselected services are integrated in such way that the resulting composition satisfies some properties on the one hand (e.g., deadlock freedom) and conforms to the target user task by consuming

all its provided inputs and generating all its required outputs on the other hand.

As is the case of chaining algorithms, both these two composition models generate a degree of uncertainty regarding the way networked services are combined. Indeed, verifying that the resulting service composition is deadlock free does not guarantee that the user's information has not been transformed using unexpected and inappropriate capabilities (e.g., capabilities that a user would not have employed himself to achieve his objective) just in order to meet the target user task's input/output specification.

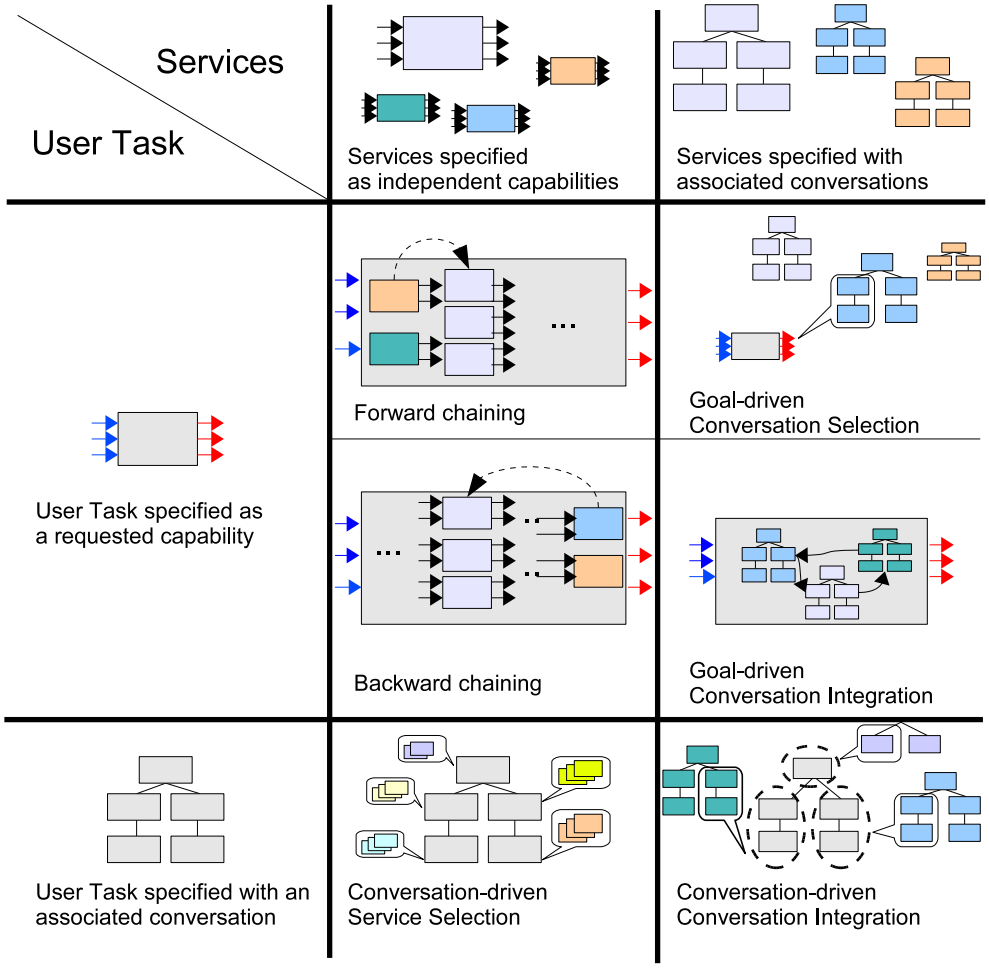


Figure 5.1: Composition Models

The last composition model, i.e., conversation-driven conversation integration assumes

a complex behaviour for both user tasks and services. In this model, conversations of networked services are integrated towards the realization of the user task's conversation. This composition model is further the most comprehensive of all the considered models as it supports maximum expressiveness for task and service functional descriptions. The benefit of this composition model is that:

1. The user task's behaviour is used as a basis for service composition, which ensures that the user requirements are fulfilled by construction.

2. A valid consumption of the composed services is ensured as their conversations are fulfilled.

Further, as shown in Figure 5.2, conversation-driven conversation integration allows reconstructing the user task conversation using different composition schemes. In this figure, a user task, depicted in the middle of the figure, is composed in four different pervasive environments using four different scenarios. In the first scenario, the task is realized through the integration of individual capabilities of pervasive services. The second scenario describes the case where a single service that conforms to the user task conversation is selected. The third scenario represents the case where the user task is realized through the composition of fragments from two service conversations. The last composition scheme is the most flexible where the realization of the user task is performed through the interleaving of two service conversations.

Conversation-driven conversation integration is investigated in [Berardi et al., 2003], where service conversations are represented as finite-state automata. In this approach, the authors propose an exponential-time algorithm that searches for a possible service composition by reducing this problem to the satisfiability of a DPDL (Deterministic Propositional Dynamic Logic) formula. However, this solution does consider neither service semantic specifications nor service and task non-functional properties. Furthermore, this algorithm employs costly formal verification algorithms, the efficiency of which is not assessed for resource-constrained devices.

Hence, the quest is still open for a solution to service composition that supports the integration of service conversations to realize the conversation of a user task. This func-

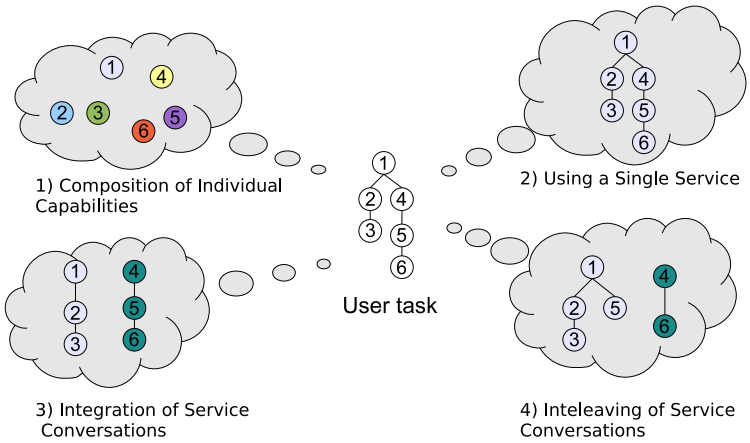


Figure 5.2: Flexibility of the Conversation-Driven Conversation Integration

tionality should further support the semantic specification of service and task capabilities, enable QoS-aware service composition and provide the means to adapt its flexibility according to the required efficiency with respect to the limited resources of thin devices.

5.2 Semantic-, QoS-aware Conversation-driven Conversation Integration: Overview

To deal with the above requirements, we present in this chapter, a solution to the semantic-, QoS-aware conversation-driven conversation integration. The objective of this solution is to provide a ranked list of concrete realizations of a user task. Each of these realizations semantically conforms to the target user task in terms of functional and non-functional properties. This solution, provided as a part of our SOM decomposes into four middleware functionalities, i.e., service discovery client, service conformance, service coordination and QoS-aware composition. Each of these functionalities is formalized as a set of communicating functions as presented in Figure 5.3.

In this figure, the service discovery client (Section 5.3) modelled with the function **ServiceDiscoveryClient**() uses a (local or remote) service registry to select a set of services $s_1, ..., s_{n_1}$ candidate to take part of the realization of the user task T based on their pro-

vided capabilities. Specifically, each service s_i provides capabilities that are semantically equivalent to some of the user task required capabilities and that further fulfil the user task local QoS properties. Then, the service conformance (Section 5.4) filters among the previously selected services those that have incompatible data or ordering constraints with those of the user task. This is performed using the function **DataConstraintSelection**() for data constraint verification and one of the functions **OrderingConstraintSelection**$_{IG}$() () or **OrderingConstraintSelection**$_{IL}$() for ordering constraint verification. This generates two possible sets of services candidate to take part in the user task realization, which differ from each other in the flexibility they enable for carrying out user task realizations. Hence, based on these two sets of services the service coordination functionality attempts to reconstitute the task conversation using four different algorithms, i.e., **ConversationIntegration**(), **ConversationInterleaving**(), **AdaptiveIntegration**() and **AdaptiveInterleaving**() (Section 5.5). All these algorithms assess the fulfilment of the user task global QoS requirements from the aggregation of the QoS provided by the composed services using formalisms presented in Section 5.6. These four algorithms differ in their flexibility and corresponding computation cost towards the user task realization as discussed in Section 5.8.

5.3 Service Discovery Client

The service discovery client selects from a set of services those providing capabilities that conform in terms of both functional and non-functional properties to capabilities of the user task. This functionality is a client functionality of the service location functionality provided by the service registry of our SOM. More formally, according to our conceptual model introduced in Chapter 3, consider a user task $T =< P_T, C_T >$ where C_T is the set of capabilities characterizing the user task and P_T is the set of its non-functional properties. Each capability c_i of the set C_T is called *sub-task* in the following. For instance, the *EASY-COM* user task presented in Chapter 3 is composed of two independent sub-tasks, i.e., *EASY-Movie* and *EASY-Phone*. These sub-tasks are independent from each other, thus the realization of the user task translates into the realization of each of its

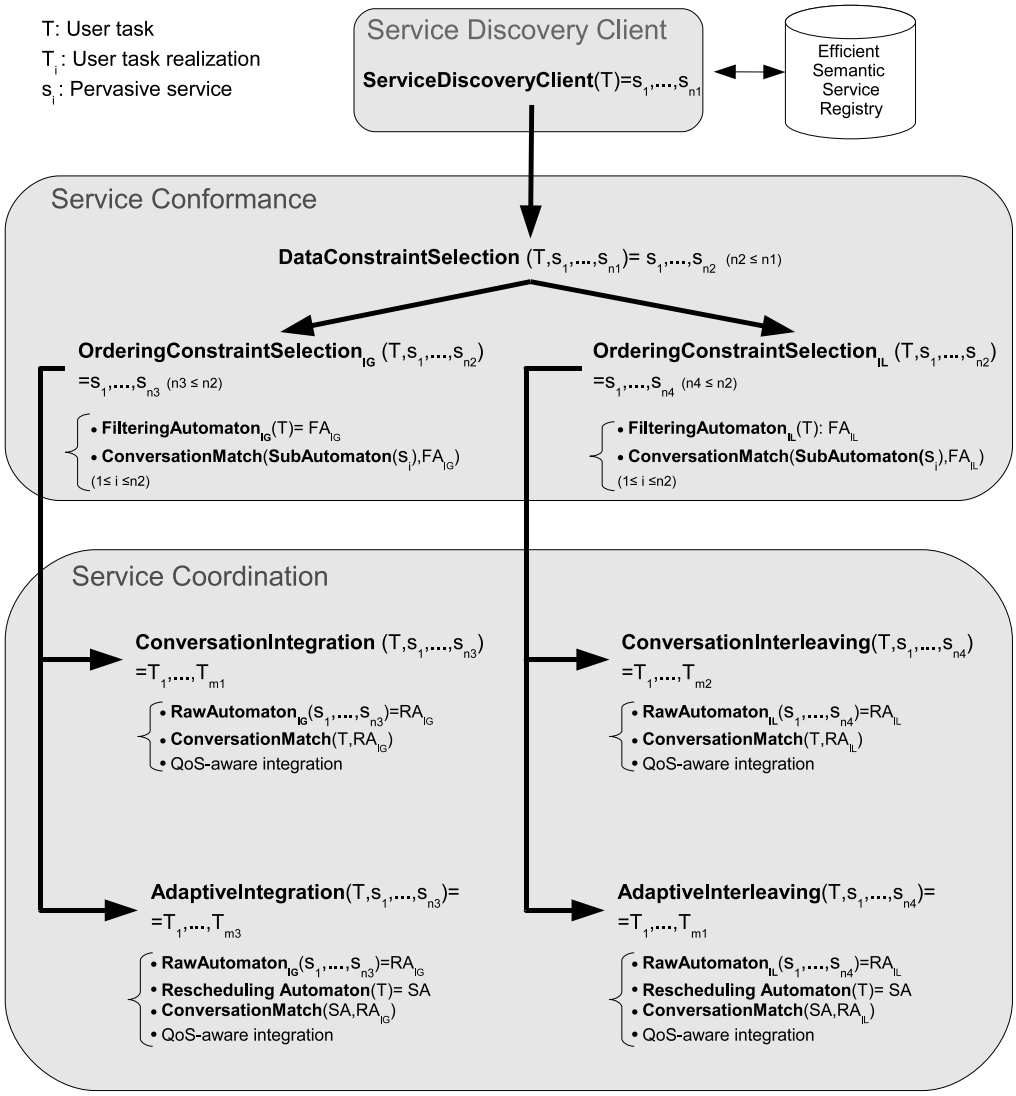


Figure 5.3: Overall Service Composition Process

sub-tasks independently. The non-functional properties associated with the user task, i.e., P_T apply to all the sub-tasks of the user task. For instance, if the user task has a required non-functional property related with the network connection, e.g., *Network is-a WiFi 802.11G*, then all the sub-tasks of the user task will have in addition to their *local* non-functional properties this *global* requirement of the user task. Each sub-task of the user task can be either elementary or composite, i.e., without or with an associated

conversation, respectively.

As the realization of each sub-task is performed using the same algorithms, we focus in the following on user tasks that are constituted of a single sub-task, and for the sake of clarity, we refer to the single sub-task of a user task by the *user task* itself. In this case, the user task can be defined either as: $T =< P_T, I_T, O_T, cat_T >$ if it is composed of a single elementary sub-task or as: $T =< P_T, A_T >$ if it is constituted of a composite sub-task. In these definitions, I_T, O_T and cat_T are the set of inputs, outputs and the category of the user task, P_T is the set of non-functional properties of the user task including both global and local non-functional properties, and A_T is the conversation of the user task expressed as a finite state automaton. In this chapter we focus on user tasks associated with a conversation. The case where the user task is expressed as an elementary (resp., a set of elementary) sub-task(s) can be treated using our solutions for efficient semantic service discovery presented in Chapter 4.

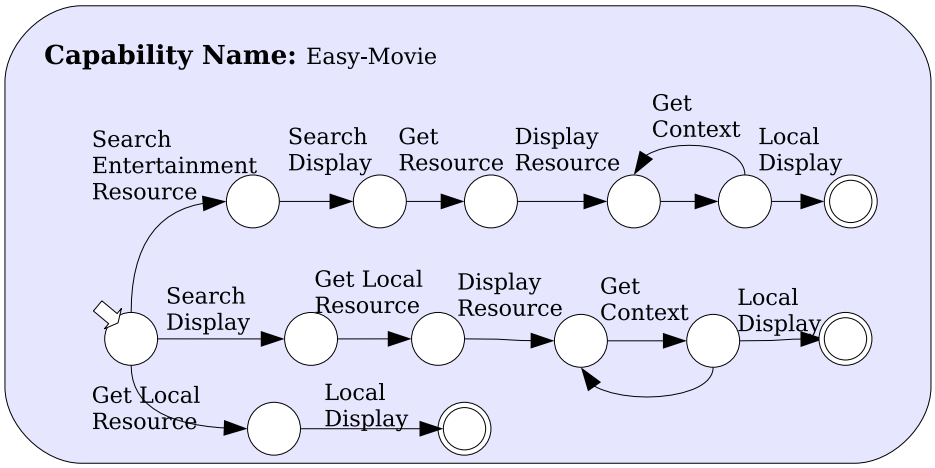


Figure 5.4: User Task Conversation

The automaton describing the task's conversation, i.e., $A_T =< Q_T, \Sigma_T, \delta_T, st_{0_T}, F_T >$, is composed of a set of states Q_T, which includes among others an initial state , i.e., st_{0_T} and a set of final states, i.e., F_T, a transition function δ_T and a set of symbols Σ_T. This set of symbols contains the capabilities that constitute the task's conversation. These capabilities, called *required* capabilities in the following, are assumed to be elementary. Pervasive

service conversations are on the contrary assumed to be *nested*, i.e., capabilities taking part in a conversation may themselves be composed of other capabilities. Nevertheless, our composition model considers only the highest level of nesting for the realization of the user task. The other levels of nesting are used for invoking the composed service when the realized user task is being executed.

Figure 5.4 and Figure 5.5 describe the conversations of a user task and a set of pervasive services, respectively, inspired from the scenario introduced in Chapter 2. For readability reasons, the transitions taking part in these conversations are labelled with capability names. The complete descriptions of the involved capabilities are given in Table 5.1. In Figure 5.4 each capability involved in the task's conversation is elementary, i.e., it is not composed of other capabilities. On the other hand, in Figure 5.5 each capability involved in the services' conversations may be either elementary or composite. For instance, the *GetLocalResource* capability provided by the user's PDA may have an associated conversation in which a login capability may be specified to restrict its utilization only to the PDA holder. This internal conversation is used at service invocation time.

The service discovery client returns to the client application a list of services, where each service provides at least one capability that semantically conforms both in terms of functional and (local) non-functional properties to a capability of the user task. More formally, the service discovery client can be modelled with a function as follows:

$$\textbf{ServiceDiscoveryClient}:\ \mathcal{T} \longrightarrow 2^{\mathcal{S}}$$
$$T \in \mathcal{T} \longmapsto \{s_1, ..., s_{n_1}\} \text{ such that:}$$
$$\forall s_i \in \{s_1, ..., s_{n_1}\} :$$
$$\exists c_T \in \Sigma_T, \exists c_{s_i} \in \Sigma_{s_i} :$$
$$\textbf{FunctionalCapabilityMatch}(c_{s_i}, c_T) \wedge$$
$$\textbf{PropertiesCapabilityMatch}(c_{s_i}, c_T)$$

where the relations **FunctionalCapabilityMatch**() and **PropertiesCapabilityMatch**() are defined in Chapter 4 and n_1 is the number of services that have been selected by the

Service	**Capability**	**In**	**Out**
User Task	Search Entertainment resource	Resource Name	Entertainment Resource Reference
	Search Display	Display features	Display Reference
	Get Resource	Resource Reference Retrieval Mode	Resource
	Display resource	Resource	
	Get Context	Requested Context	Context Information
	Get Local Resource	Resource Name	Resource
	Local Display	Resource	
Airport Entertainment Server	Search Resource	Resource Name	Video Resource ID Audio Resource ID
	Get Resource	Resource ID	Resource
Context Management System	Get Context Information	Requested Context	Context Information
	Get Number Of Persons		Number of Persons
	Get Lighting level		Lighting level
	Get Coordinates		Coordinates
	Get Temperature		Temperature
	Get Sound Level		Sound Level
Plasma Display	Display Image	Image Resource	
	Display Video Stream	Video Stream	
	Display Audio Stream	Audio Stream	
	Display Digital Resource	Digital Resource	
Carla Music Server	Search Music resource	Resource Name	Audio Resource Reference
	Get Audio Resource	Audio Resource Reference	Audio Resource
PDA	Get Local Resource	Resource Name	Resource
	Local Display	Digital Resource Reference	
	Search Displays	Display features	Display Reference

Table 5.1: Capabilities of the EASY-Movie User Task and Selected Pervasive Services

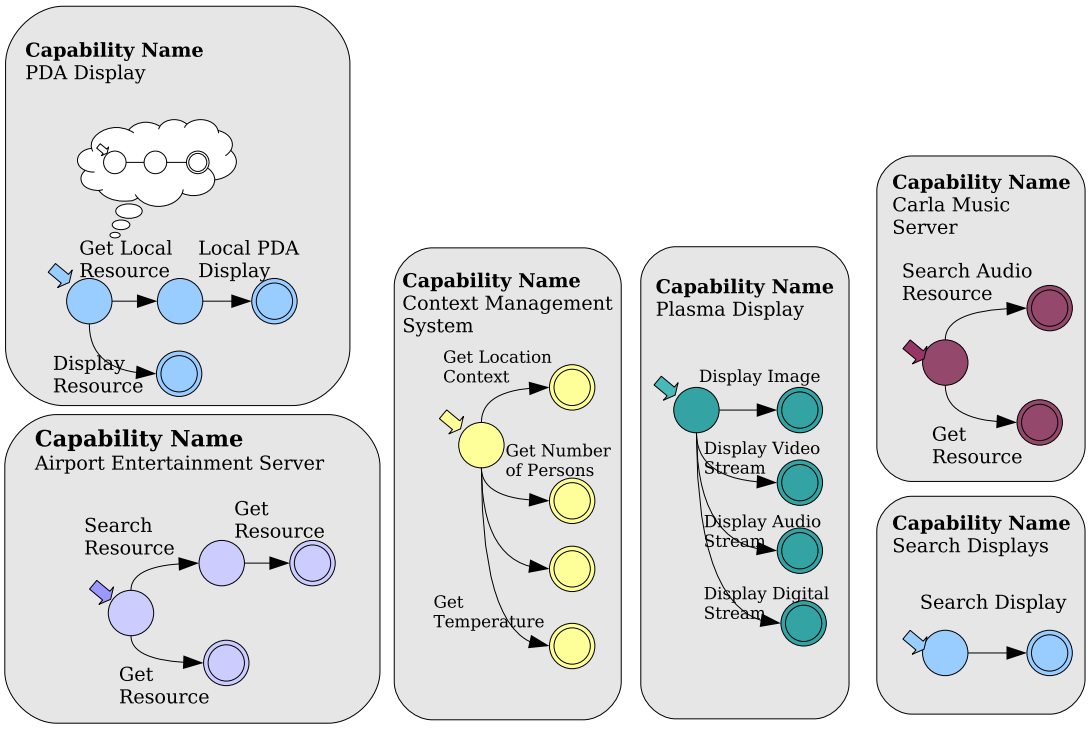


Figure 5.5: Selected Pervasive Services

service discovery client.

For the user task of Figure 5.4, all the services of Figure 5.5 are returned by the service discovery client because they provide at least one capability that semantically conforms to one of the user task required capabilities. Table 5.2 summarizes for each capability of the user task the capabilities of services such that a matching holds.

5.4 Service Conformance

Service conformance is responsible for filtering out from the set of services that have been returned by the discovery client those that will not be useful for the composition. Specifically, service conformance analyses pervasive service conversations and filters out those services that have incompatible data constraints, which are specific data flow specifications (Section 5.4.1), and ordering constraints (Section 5.4.2), with the target user task

Task Capability	**Service Capability**	**Service**
Search Entertainment Resource	-Search Resource -Search Music Resource	-Airport Entertainment Server -Carla Music Server
Search Display	Search Displays	PDA
Get Resource	-Get Resource -Get Audio Resource	-Airport Entertainment Server -Carla Music Server
Get Context	Get Context	Context Management System
Get Local Resource	Get Local Resource	PDA
Local Display	Local Display	PDA

Table 5.2: Matching Between Capabilities of the EASY-Movie User Task and Selected Pervasive Services

conversation.

5.4.1 Data Flow and Data Constraints

Our model for semantic service specification supports the specification of data flow between capabilities. According to our model presented in Chapter 3, a data flow is defined between two capabilities when output information produced by one capability serves as input information for another capability. This is defined with the function:

$$\Phi_D : \Sigma_T \longrightarrow 2^{\Sigma_T \times \mathcal{N}^2}$$
$$c \quad \longmapsto \{< c_i, o_i, i_i >: i = 0..n\}$$

where Σ_T is the set of required capabilities of the user task and $\mathcal{N}$ is a finite set of concepts over the finite set of ontologies $\mathcal{O}$.

This is interpreted as: the output o_i produced by the capability c is consumed by the capability c_i as the input i_i.

While data flow defines data transfer between the capabilities of the user task enabling an orchestration of the composed pervasive services, it does not play a role in the filtering of services. Indeed, the only condition that has to be checked is the compatibility between the capabilities inputs and outputs associated in a data flow definition. This compatibility, which can be assessed using the **ConceptMatch**() relation (defined in Chapter 4), is performed when the user task is specified and is valid by definition when the services are initially selected by the service discovery client. This is verified by Property [Prop 3], proved in Appendix A.

[Prop 3)]: $\forall n_1, n_2 \in \mathcal{N}^2$, **ConceptMatch**($n_1$,$n_2$):
$[\exists n'_1, n'_2 \in \mathcal{N}^2$: **ConceptMatch**($n'_1$,$n_1$) $\wedge$ **ConceptMatch**(n'_2,n_2) $\Rightarrow$ **ConceptMatch**(n'_1,n'_2)$]$

This property specifies that if the relation **ConceptMatch**() holds between two concepts n_1 and n_2 in a data flow definition of the user task, then this relation also holds between two concepts n'_1 and n'_2 of pervasive service capabilities that have been pre-selected by the service discovery client because n'_1 matches n_1 and n'_2 matches n_2.

In addition to the specification of data flow between capabilities of the user task, our model supports the specification of data constraints. Specifically, a data constraint between two capabilities of the user task specifies a data flow that must be performed between two capabilities of the same pervasive service. Thus, data constraints, contrary to simple data flow specification, directly influence the service filtering process performed by the service conformance functionality.

Figure 5.6 describes the data flow and data constraints specified in the user task of our example. In this figure, a data constraint is specified between the capabilities *Search Entertainment Resource* and *Get Resource*. This means that these two capabilities have to be provided by the same service, as the service supporting browsing and selecting a specific resource is most likely the service that delivers it.

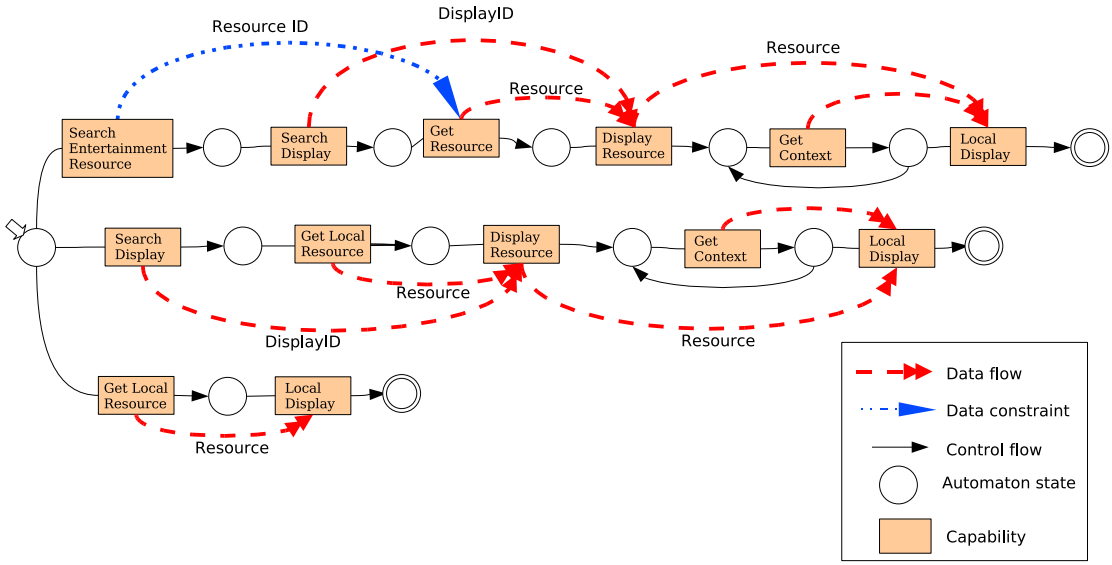


Figure 5.6: Data Flow and Data Constraints Example

More formally, the data constraints are modelled as a sub-set of the data flow specifications, i.e., the function Φ_C, that gives for a capability the set of data constraints with other capabilities is a restriction of the function Φ_D to a subset of Σ_T.

Data constraints are assessed by the service conformance functionality by checking for each data constraint definition whether the two capabilities on which the constraint is defined, belong to the same service.

The functionality provided by service conformance regarding data constraints can be modelled with the following function:

$$\textbf{DataConstraintSelection}: \mathcal{T} \times \mathcal{S}^{n_1} \longrightarrow 2^{\mathcal{S}}$$

$$< T, s_1, ..., s_{n_1} > \longmapsto \{s_1, ..., s_{n_2}\}:$$

$$\forall s_i \in \{s_1, ..., s_{n_2}\},$$

$$\forall (c_i, c_j) \in {\Sigma_T}^2, (i, o) \in \mathcal{N}^2 :< c_j, o, i > \in \Phi_C(c_i),$$

$$\big(\exists c'_i \in \Sigma_{s_i}:$$

$$\textbf{FunctionalCapabilityMatch}(c'_i, c_i) \wedge$$

$$\textbf{PropertiesCapabilityMatch}(c'_i, c_i)\big)$$

$$\Rightarrow$$
$$\big(\exists c'_j \in \Sigma_{s_i}:$$
$$\textbf{FunctionalCapabilityMatch}(c'_j,c_j) \wedge$$
$$\textbf{PropertiesCapabilityMatch}(c'_j,c_j)\big)$$

where $\mathcal{S} = \{s_1, ..., s_{n_1} = \textbf{ServiceDiscovery}(T)$ and $n_2 \leq n_1$ is the number of services that fulfil task data constraints.

5.4.2 Ordering Constraints

Ordering constraints are due to the structure of pervasive service conversations. As service conversations are specified using finite state automata, the elementary workflow patterns are translated to a combination of the sequence and the choice between capabilities (see Chapter 3). If two capabilities are specified one in sequence of the other in a service conversation, we say that these two capabilities have an *ordering constraint* between each other. For enforcing a valid consumption of the composed pervasive services, the dynamic realization of user tasks has to fulfil pervasive service ordering constraints. We present in the following, two functions for the filtering of pervasive services. The first function, i.e., $\textbf{OrderingConstraintSelection}_{IG}()$, allows selecting pervasive services that provide fragments of the user task conversation. These fragments are then used by the service coordination functionality to reconstitute the task conversation. The second function, i.e., $\textbf{OrderingConstraintSelection}_{IL}()$, allows further to the first one, the selection of services that enable the reconstitution of user tasks by interleaving their conversations. We distinguish between these two selection modes because they provide two levels of flexibility for the user task realization. Each of these two levels of flexibility come with their associated cost and our objective is to allow the middleware to choose the most appropriate composition model with respect to the available resources of their mobile devices. A discussion about the cost of each solution is given in Section 5.8.

Ordering Constraint-based Service Selection for the Support of Conversation Integration

We present in this section the first function, which allows the selection of pervasive services that have compatible ordering constraints with the user task conversation. This selection can be done using automata analysis algorithms. First, an automaton that allows filtering the services that have incompatible ordering constraints with the user task conversation is built from the user task automaton. This automaton, referred to in the following as the *filtering automaton for the support of conversation integration*, and noted FA_{IG}, is then compared with the automaton of each selected pervasive service to check whether there exists an intersection between the languages generated by the two automata.

The filtering automaton is built by performing the following transformations to the task's automaton A_T:

1. All the non-final states excluding the initial state, become final if they are not already final.

2. ϵ-transitions are added from the initial state to all the states of the automaton

More formally, consider the automaton $A_T =< Q_T, \Sigma_T, \delta_T, st_{0_T}, F_T >$, the filtering automaton $FA_{IG} =< Q_{FA}, \Sigma_{FA}, \delta_{FA}, st0_{FA}, F_{FA} >$ is defined as follows:

- $Q_{FA} = Q_T$
- $\Sigma_{FA} = \Sigma_T \cup \{\epsilon\}$
- $\delta_{FA} = \delta_T \cup < st0_{FA}, \epsilon, st_i >, \forall st_i \in Q_T - \{st0_{FA}\}$
- $st0_{FA} = st_{0_T}$
- $F_{FA} = Q_T - st_{0_T}$ if $\neg(st_{0_T} \in F_T)$

Service conformance assesses the fulfilment of the ordering constraints of selected pervasive services by comparing their conversations with the filtering automaton of the user task. Those services that have incompatible ordering constraints with the user task conversation are filtered out. This assessment can be performed using automata compatibility

checking algorithms. Specifically, we define the relation **ConversationMatch**() to compare two conversations specified using finite state automata. Let $A_1 =< Q_1, \Sigma_1, \delta_1, st0_1, F_1 >$, $A_2 =< Q_2, \Sigma_2, \delta_2, st0_2, F_2 >$ be two automata, this relation is defined as follows:

$$
\begin{aligned}
\textbf{ConversationMatch}(A_1,A_2) \Leftrightarrow\ & \exists R \text{ on } Q_1 \times Q_2 \text{ such that:} \\
& \forall < st_1, st_1' > \in R,\ \forall c \in \Sigma_1: \\
& \delta_1(st_1,c)=st_2 \Rightarrow \exists c' \in \Sigma_2 : \\
& \textbf{FunctionalCapabilityMatch}(c',c) \wedge \\
& \textbf{PropertiesCapabilityMatch}(c',c) \wedge \\
& \delta_2(st_1',c')=st_2' \wedge \\
& < st_2, st_2' > \in R
\end{aligned}
$$

where R is a binary relation defined on the set $Q_1 \times Q_2$. This relation is commonly named *automata simulation* in the automata theory, and we say that the automaton A_2 *simulates* the automaton A_1 or that A_1 is *simulated by* A_2.

A pervasive service is selected by the service conformance functionality if the **ConversationMatch**() relation holds between the filtering automaton and a sub-automaton of this service automaton. More formally, the selection (for the support of conversation integration) performed by the service conformance functionality regarding ordering constraints can be modelled with the following function:

$$
\begin{aligned}
\textbf{OrderingConstraintSelection}_{IG}:\ & \mathcal{T} \times \mathcal{S}^{n_2} \longrightarrow 2^{\mathcal{S}} \\
& < T, \{s_1, ..., s_{n_2}\} > \longmapsto \{s_1, ..., s_{n_3}\}: \\
& \qquad FA_{IG} = \textbf{FilteringAutomaton}_{IG}(T) \wedge \\
& \qquad \forall s_i \in \{s_1, ..., s_{n_3}\}, \\
& \qquad \exists A' : \textbf{SubAutomaton}(A_{s_i},A'): \\
& \qquad \textbf{ConversationMatch}(A',FA_{IG})
\end{aligned}
$$

where: $\mathcal{S} = \{s_1, ..., s_{n_2}\}$ is a set of services selected by the service discovery client and fulfil

the task data constraints (see Figure 5.3). Further, A_{s_i} is the automaton representing the conversation of the service s_i, $\mathbf{FilteringAutomaton}_{IG}()$ is a function for generating the filtering automaton FA_{IG} enabling the support of conversation integration related with the task T.

The relation **SubAutomaton**() is defined as follows. Let $A_1 =< Q_1, \Sigma_1, \delta_1, st0_1, F_1 >$, $A_2 =< Q_2, \Sigma_2, \delta_2, st0_2, F_2 >$ be two automata:

$$
\begin{aligned}
\mathbf{SubAutomaton}(A_1,A_2) \Leftrightarrow & \text{ - } Q_2 \subseteq Q_1 \\
& \text{ - } \Sigma_2 \subseteq \Sigma_1 \\
& \text{ - } \delta_2 : Q_2 \times \Sigma_2 \rightarrow Q_2 \\
& \qquad < st, c > \mapsto \delta_2(st, c) = \delta_1(st, c) \\
& \text{ - } \forall st \in Q_2, \forall c \in \Sigma_2 : \delta_2(st, c) = \emptyset \Rightarrow st \in F_2 \\
& \text{ - } st0_2 = st0_1 \\
& \text{ - } F_2 \subset F_1
\end{aligned}
$$

Note that from the definition, all the states of the sub-automaton A_2 that do not have outgoing transitions, are final.

Figure 5.7 describes the filtering automaton FA_{IG} associated with the user task of Figure 5.4. Using this service selection function, the service *Carla Music Server* is selected by the service conformance functionality as the whole automaton describing this service conversation is simulated by the filtering automaton of Figure 5.7. Moreover, the service *Plasma Display* is also selected even if its conversation additionally provides capabilities that are not required in the user task conversation. Indeed, there exist a sub-automaton of this automaton, i.e., the one containing the transition labelled with the *Display Digital Stream* capability, that is simulated by the filtering automaton of Figure 5.7. While all the capabilities of the *Airport Entertainment Server* match some capabilities of the user task (see Table 5.2), only a part of its conversation,i.e., the one containing the transition labelled with the *Get Resource* capability, is selected using this selection function. This

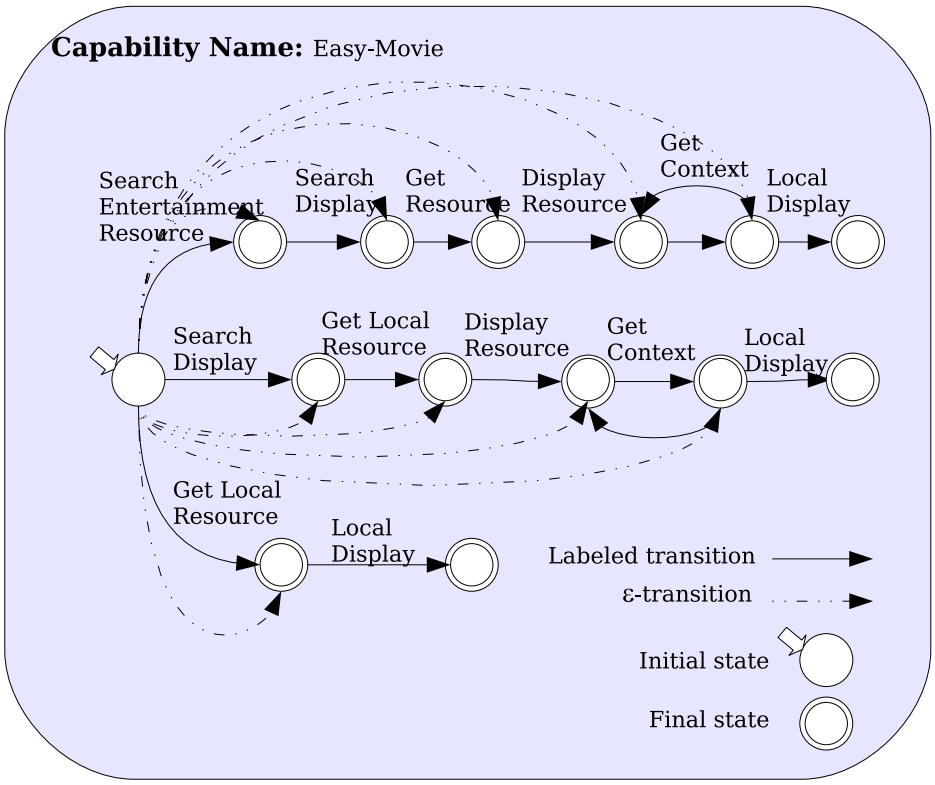


Figure 5.7: Filtering Automaton for the Support of Conversation Integration

is due to the fact that the selection function does not support the interleaving of service conversations. In the following section, we present a function that allows the selection of services for the support of conversation interleaving.

Ordering Constraint-based Service Selection for the Support of Conversation Interleaving

As in the previous case, the selection of pervasive services for the support of conversation interleaving is carried out using automata analysis algorithms. Specifically, a filtering automaton is built from the user task automaton. This automaton, noted FA_{IL} in the following, can then be compared with the automaton of a pervasive service to check whether there exists an intersection between the languages generated by these automata.

The filtering automaton is built by performing the following transformations to the

task's automaton A_T:

1. If there exists a path from the non-final state st_i to the non-final state st_j in the automaton A_T (by excluding loop transitions), add an ϵ-transition from st_i to st_j
2. All the non-final states excluding the initial state become final if they are not already final.

More formally, consider the automaton $A_T =< Q_T, \Sigma_T, \delta_T, st_{0_T}, F_T >$, the filtering automaton for the support of conversation interleaving $FA_{IL} =< Q_{FA}, \Sigma_{FA}, \delta_{FA}, st0_{FA}, F_{FA} >$ is defined as follows:

- $Q_{FA} = Q_T$
- $\Sigma_{FA} = \Sigma_T \cup \{\epsilon\}$
- $\delta_{FA} = \delta_T \cup < st_i, \epsilon, st_j >, \forall < st_i, st_j > \in Q_T^2$ such that:
  - $\exists \{st_1, ..., st_n\} \subset Q_T$: $\forall i, j \in \{0..n\} : st_i \neq st_j$, and $\exists \{c_0, ..., c_n\} \subset \Sigma_T$:
  - $\delta_T(st_i, c_0) = st_1$
  - $\delta_T(st_1, c_1) = st_2$
  - ...
  - $\delta_T(st_{n-1}, c_{n-1}) = st_n$
  - $\delta_T(st_n, c_n) = st_j$
- $st0_{FA} = st_{0_T}$
- $F_{FA} = Q_T - st_{0_T}$ if $\neg(st_{0_T} \in F_T)$

Figure 5.8 describes the filtering automaton FA_{IL} for the support of conversation interleaving extracted from the task automaton of Figure 5.4. Contrary to the previous service selection function, using this automaton, all the conversation of the service *Airport Entertainment Server* is selected as a potential candidate for the realization of the user task. Indeed, the whole automaton representing this service conversation can be simulated by the filtering automaton of Figure 5.8.

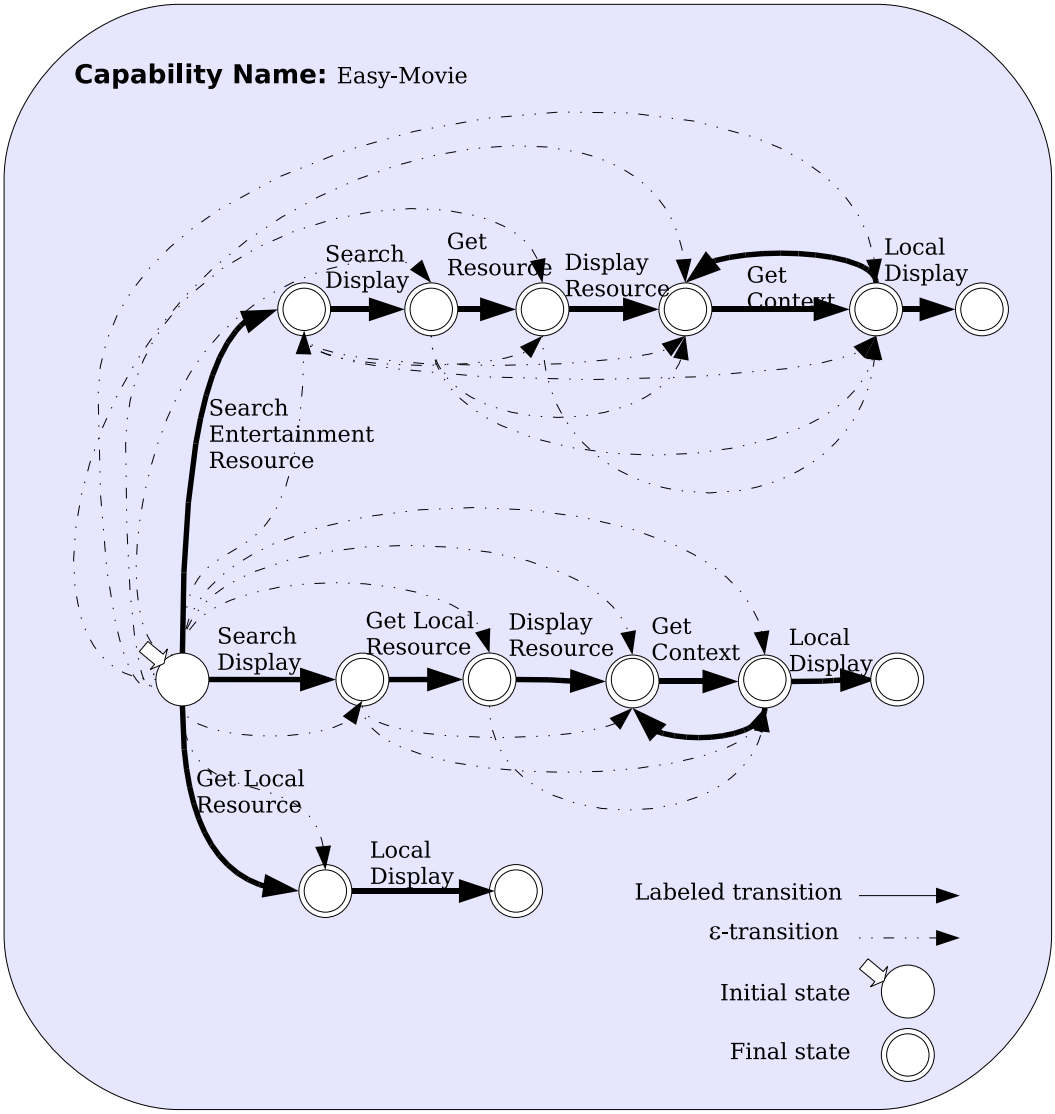


Figure 5.8: Filtering Automaton for the Support of Conversation Interleaving

As in the previous case, service conformance assesses the fulfilment of the ordering constraints of pervasive services by comparing their conversations with the filtering automaton of the user task. Specifically, a pervasive service is selected by the service conformance functionality if the **ConversationMatch**() relation holds between the filtering automaton and a sub-automaton of this service. This selection can be modelled using the following function (see Figure 5.3):

$$\textbf{OrderingConstraintSelection}_{IL}\colon \mathcal{T} \times \mathcal{S}^{n_2} \longrightarrow 2^{\mathcal{S}}$$
$$< T, \{s_1, ..., s_{n_2}\} > \longmapsto \{s_1, ..., s_{n_3}\}\colon$$
$$FA_{IL} = \textbf{FilteringAutomaton}_{IL}(T) \wedge$$
$$\forall s_i \in \{s_1, ..., s_{n_3}\},$$
$$\exists A' : \textbf{SubAutomaton}(A_{s_i}, A')\colon$$
$$\textbf{ConversationMatch}(A', FA_{IL})$$

where: $\mathcal{S} = \{s_1, ..., s_{n_2}\}$ is a set of services selected by the service discovery client and that fulfil the task data constraints. Furthermore, A_{s_i} is the automaton representing the conversation of the service s_i, $\textbf{FilteringAutomaton}_{IL}()$ is a function for generating the filtering automaton FA_{IL} enabling the support of conversation interleaving related with the task T.

While the selection performed by the second function is more flexible, it is also more costly. Indeed, the set of services selected by the second function is a superset of the set of services selected by the first function. This may return more service composition solutions, i.e., greater flexibility. However, as the filtering automaton built in the second case contains a number of additional $\epsilon-$transitions, the cost of the **ConversationMatch**() function is consequently increased, as further detailed in Section 5.8).

Summarizing, the functionality provided by the service conformance functionality can be expressed using two functions: $\textbf{ServiceConformance}_{IG}()$ and $\textbf{ServiceConformance}_{IG}()$ as follows (see Figure 5.3):

$$\textbf{ServiceConformance}_{IG}(\mathrm{T}) = \textbf{OrderingConstraintSelection}_{IG}$$
$$(T, \textbf{DataConstraintSelection}$$
$$(T, \textbf{ServiceDiscoveryClient}(T)))$$

$$\textbf{ServiceConformance}_{IL}(\mathrm{T}) = \textbf{OrderingConstraintSelection}_{IL}$$
$$(T, \textbf{DataConstraintSelection}$$

(T,**ServiceDiscoveryClient**(T)))

where, T is a user task.

5.5 Service Coordination

Based on the service conformance functionality, we present in this section the service coordination functionality. Service coordination is responsible of generating a set of concrete realizations of the user task. Each of these realizations fulfils the task's non-functional properties (as presented in Section 5.6) and references capabilities of pervasive services. We present four solutions for the dynamic realization of user tasks. These solutions differ in the flexibility they enable for finding user task realizations. Specifically, the first solution presented in Section 5.5.2, allows integrating service conversations to realize the user task. The second solution presented in Section 5.5.3, allows integrating service conversations by additionally enabling the interleaving of their conversations. Finally, the third and fourth solutions presented in Section 5.5.4, adapt the user task's conversation according to its data flow specification to further increase the probability of finding a composition. The difference between these latter two solutions resides in the support of conversation interleaving. By distinguishing between these solutions, we can provide the user with the most appropriate solution with respect to the available computing resources on his/her device. Thus, in a resource rich environment, the most flexible solution, which increases the probability of finding a user task realization, would be employed, while a less flexible solution would be used in a resource constrained environment. In the following, we formally define the problem of task realization, and then detail the solutions introduced above.

5.5.1 Problem Definition

Consider a user task $T =< P_T, A_T >$, where P_T is the set of non-functional properties of the task and A_T is the automaton describing the task's conversation. The set of services candidate for the composition of a user task T are given by: **ServiceConformance**(T),

where the **ServiceConformance**() function can be either **ServiceConformance**$_{IG}$() or **ServiceConformance**$_{IL}$() defined in the previous section. The objective of the service coordination functionality is then to find a ranked list of concrete realizations of the user task: $T_1 =< P_{T_1}, A_{T_1} >$, $T_2 =< P_{T_2}, A_{T_2} >$, ... , $T_n =< P_{T_n}, A_{T_n} >$ such that:

$\forall T_i =< P_{T_i}, A_{T_i} >$, where $A_{T_i} =< Q_{T_i}, \Sigma_{T_i}, \delta_{T_i}, st0_{T_i}, F_{T_i} >$:

- $\forall p \in P_T, \exists p' \in P_{T_i}$:
  - **ConceptMatch**(p',p), if p and p' are qualitative properties
  - **NumericExpressionMatch**(p',p), if p and p' are quantitative properties
- $Q_{T_i} = Q_T$
- $\Sigma_{T_i} \subset \cup\Sigma_{s_i}$ $\forall s_i \in$ **ServiceConformance**(T): $\forall c_i \in \Sigma_T, \exists c_j \in \Sigma_{T_i}$:
  - **FunctionalCapabilityMatch**(c_j,c_i)
  - **PropertiesCapabilityMatch**(c_j,c_i)
- $\delta_{T_i} = \delta_T$
- $F_{T_i} = F_T$
- **TaskDoM**(T_i,T) < **TaskDoM**(T_{i+1},T)

where: the relations **ConceptMatch**(), **NumericExpressionMatch**(), **FunctionalCapabilityMatch**() and **PropertiesCapabilityMatch**() are defined in Chapter 4.

The concrete realizations of the user task are ranked according to their degree of match with the initial task using the **TaskDoM**() function defined as follows:

TaskDoM(T_i, T) = **PropertiesDoM**(T_i, T) + **ConversationDoM**(T_i, T)

where the **PropertiesDoM**() function between tasks is defined similarly to the **PropertiesDoM**() function between capabilities (see Chapter 4), as:

$$\textbf{PropertiesDoM}(T_i,\, T) = \sum_{i=1}^{n} w_i * p_i$$

where w_i is the relative importance of the property p_i and $n = |P_T|$ is the number of global non-functional properties required by the user task.

The **ConversationDoM**() function allows the evaluation of the semantic distance between two conversations that conform to each other. It is defined as the sum of the semantic distances between each pair of capabilities that match from the first and the second conversation. More formally, if we consider a user task $T =< P_T, A_T >$ and a concrete realization of this task $T_i =< P_{T_i}, A_{T_i} >$. Assume that the sets of capabilities Σ_T and Σ_{T_i} are ordered in the form: $\Sigma_T = \{c1_T, ..., cn_T\}$, $\Sigma_{T_i} = \{c1_{T_i}, ..., cn_{T_i}\}$ such that $\forall j \in \{1, ..., n\}$: **CapabilityMatch**($cj_{T_i}$,$cj_T$). The **ConversationDoM**() function is defined as:

$$\textbf{ConversationDoM}(T_i,\, T) = \sum_{i=1}^{n} \textbf{CapabilityDoM}(c_i, c_T)$$

where n=$|\Sigma_T|$ is the number of required capabilities of the user task.

5.5.2 Integrating Service Conversations

We present in this section the first solution to the dynamic realization of the user task. This solution uses the first service conformance function defined in Section 5.4.2, i.e., **ServiceConformance**$_{IG}$(), which allows the selection of services that can potentially be integrated without the support of conversation interleaving.

In this solution, we first need to build an automaton that connects the automata of the selected services together. The resulting automaton is called the *raw automaton* and noted RA_{IG} in the following. This automaton contains a new start state and empty transitions that connect this state with the start states of all selected services automata.

The raw automaton also contains empty transitions that connect the final states of each selected automaton with the new start state, which allows, if needed, the integration of the same service multiple times in the same composition. More formally, consider a set of services $s_1, s_2, ..., s_{n_3}$ selected by the service conformance functionality with their associated conversations $A_1, A_2, ..., A_{n_3}$ respectively, where $A_i =< Q_i, \Sigma_i, \delta_i, st0_i, F_i >$. The raw automaton $RA_{IG} =< Q_{RA}, \Sigma_{RA}, \delta_{RA}, st0_{RA}, F_{RA} >$ generated by a function $\mathbf{RawAutomaton}_{IG}(s_1, s_2, ..., s_{n_3})$, is defined as follows:

- $Q_{RA} = \cup_{i=1}^{n} Q_i \cup st0_{RA}$
- $\Sigma_{RA} = \cup_{i=1}^{n} \Sigma_i \cup \{\epsilon\}$
- $\delta_{RA} : Q_{RA} \times \Sigma_{RA} \rightarrow Q_{RA}$

$$st, c \mapsto \delta_{RA}(st,c)=\delta_i(st,c) \text{ when } st \in Q_i \text{ and } c \in \Sigma_i$$
$$\delta_{RA}(st,c)=st0_i \text{ when } st = st0_{RA} \text{ and } c = \epsilon$$
$$\delta_{RA}(st,c)=st0_{RA} \text{ when } st \in F_{RA} \text{ and } c = \epsilon$$

- $F_{RA} = \cup_{i=1}^{n} F_i$

Figure 5.9 presents the raw automaton built from the user task of Figure 5.4. Based on this automaton, service coordination uses the relations **ConversationMatch**() defined in Section 5.4.2 to find user task realizations. Specifically, there exists a realization of the user task if the raw automaton A_{RA} simulates the task automaton A_T. More formally, this can be represented by the function **ConversationIntegration**() as follows:

ConversationIntegration: $\mathcal{T} \times \mathcal{S}^{n_3} \longrightarrow 2^{\mathcal{T}}$

$$< T, s_1, ..., s_{n_3} > \longmapsto \{T_1, ..., T_n\}:$$
$$RA_{IG} = \mathbf{RawAutomaton}_{IG}(s_1, ..., s_{n_3}) \wedge$$
$$\forall T_i \in \mathcal{T},$$
$$T_i = \mathbf{SubAutomaton}(RA_{IG}):$$
$$\mathbf{ConversationMatch}(A_T, A_{T_i})$$

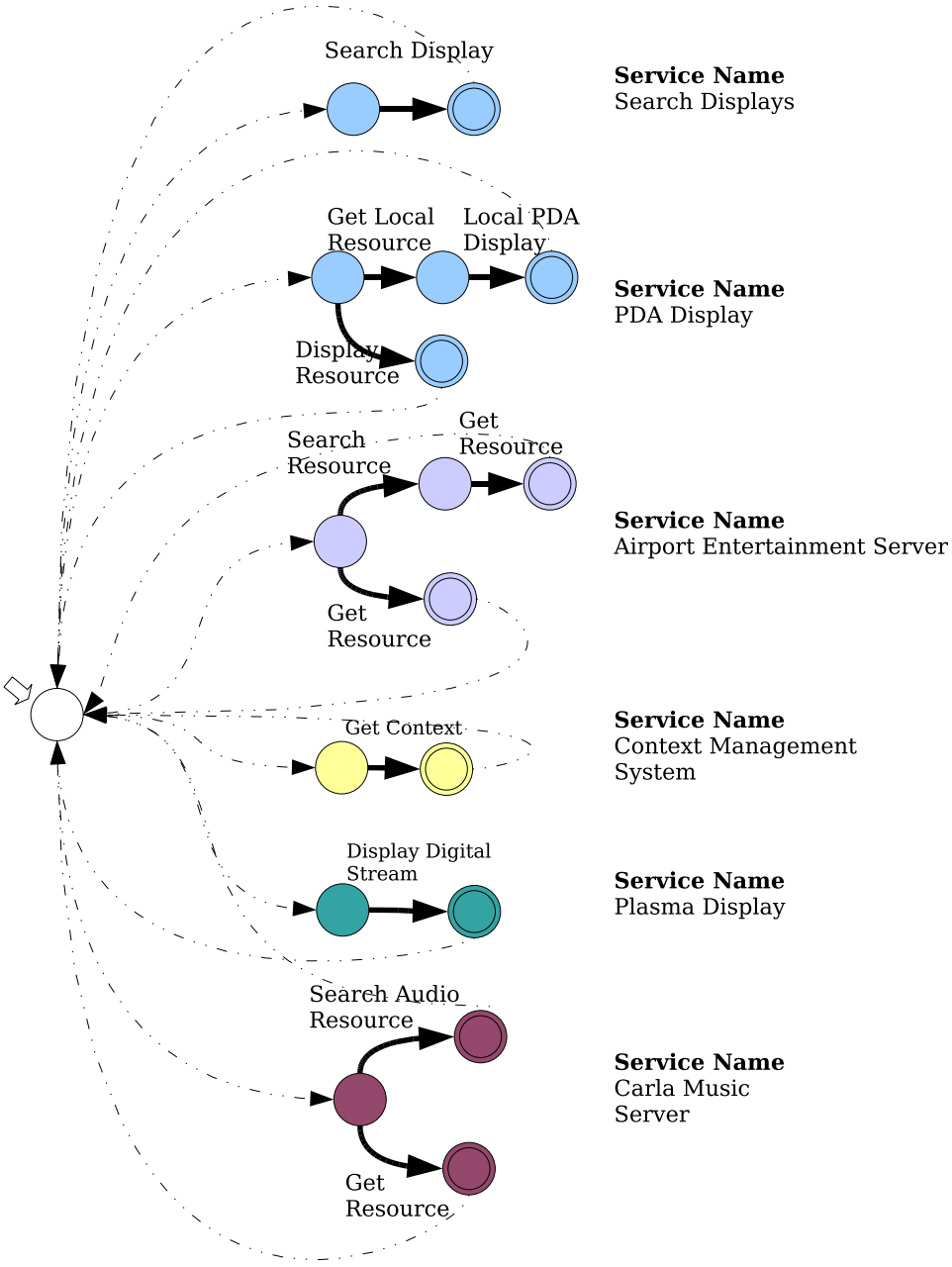


Figure 5.9: Raw Automaton

The user task realizations should further fulfil the task's non-functional properties. This verification is discussed in Section 5.6.

5.5.3 Support of Conversation Interleaving

We present in this section the second solution to dynamic user task realization. This solution supports the interleaving of pervasive service conversations. It uses the second service conformance function defined in Section 5.4.2, i.e., **ServiceConformance**$_{IL}$(), which allows the selection of services that can potentially be integrated with possible interleaving of their conversations. Again in this solution, we need to build a raw automaton, noted RA_{IL} in the following, from the set of selected services. However, this automaton is different from the raw automaton used in the previous solution. The raw automaton built to support conversation interleaving represents the *asynchronous free product* of the selected services automata. More formally, consider the set of selected services $s_1, s_2, ..., s_n$ with their associated conversations $A_1, A_2, ..., A_n$ respectively, where $A_i =< Q_i, \Sigma_i, \delta_i, st0_i, F_i >$. The raw automaton $RA_{IL} =< Q_{RA}, \Sigma_{RA}, \delta_{RA}, st0_{RA}, F_{RA} >$ resulting from the asynchronous free product of the automata $A_1, A_2, ..., A_n$ performed by the function **RawAutomaton**$_{IL}$(), is defined as follows:

- $Q_{RA} \subset Q_1 \times Q_2 \times ... \times Q_n$
- $\Sigma_{RA} = \cup_{i=1}^{n} \Sigma_i \cup \{\epsilon\}$
- $\delta_{RA} : Q_{RA} \times \Sigma_{RA} \rightarrow Q_{RA}$

  $(< st_1, ..., st_n >, c) \mapsto \delta_{RA}(< st_1, ..., st_n >, c) =< st'_1, ..., st'_n >$ if

  $\exists k \in \{1...n\}$ such that:

  $st'_j = st_j \ \forall j \neq k \ \wedge$

  $\delta_k(st_k, c) = st'_k$
- $st0_{RA} =< st0_1, st0_2, ..., st0_n >$
- $F_{RA} \subset \{< st_1, st_2, ..., st_n > \in Q_{RA} | st_1 \in F_1 \wedge st_2 \in F_2 \wedge ... \wedge st_n \in F_n\}$

Based on this raw automaton, service coordination uses the relation **Conversation-Match**() defined in Section 5.4 to find user task realizations. Specifically, there exists a realization of the user task if the raw automaton RA_{IL} simulates the user task automaton A_T. More formally, this can be represented by the function **ConversationInterleaving**() as follows:

ConversationInterleaving: $\mathcal{T} \times \mathcal{S}^{n_3} \longrightarrow 2^{\mathcal{T}}$

$< T, s_1, ..., s_{n_3} > \longmapsto \{T_1, ..., T_n\}$:

$RA_{IL} = \textbf{RawAutomaton}_{IL}(s_1, ..., s_{n_3}) \wedge$

$\forall T_i \in \mathcal{T},$

$T_i = \textbf{SubAutomaton}(RA_{IL})$:

$\textbf{ConversationMatch}(A_T, A_{T_i})$

5.5.4 Support of Adaptive User Tasks

We present in this section the two last solutions to the dynamic realization of the user task. These solutions allow finding a greater number of concrete realizations of the user task than the first two. They are based on the following observation: *The user task is defined by a service developer. Its conversation represents one of the possible ways of satisfying the user's intention.* Indeed, a user task can be compared to a cooking recipe. In this recipe, it is first important to find and employ the advertised ingredients, i.e., the capabilities required by the task. Then, it is important to respect the main steps of the preparation. For instance, we can not fry the onions before peeling them, i.e., data and ordering constraints. However, in some cases, if allowed by the recipe, it is possible to invert some phases of the preparation. For instance, putting the eggs before the flour or the flour before the eggs in the mixture does not change the taste of the pie. Similarly, in the dynamic realization of user tasks it may be possible to change the structure of the user task in order to increase the probability of finding a composition. The modification in the structure of the task constitutes in changing the order between capabilities. This can be done under the condition that there is no data flow between the capabilities to be rescheduled. More specifically, we generate from the user task conversation an automaton that contains all the rescheduling possibilities that fulfil the task data flow specification. This automaton called the *rescheduling automaton* and noted SA in the following, is built based on a dependency graph between capabilities. This graph is extracted from the graphical representation of the data flow of the user task by removing the automaton

states. For instance, the dependency graph extracted from the data flow specification of Figure 5.6 is represented in Figure 5.10. This graph decomposes in three independent sub-graphs. Then, the rescheduling automaton is built based on the dependency graph by the function **RescedulingAutomaton**() as follows:

1. Create an initial state for the rescheduling automaton
2. If there is no more capabilities in the dependency graph, stop
3. Else: for each capability c that does not have a predecessor in the dependency graph generate from the current state st_i of the rescheduling automaton the transitions $< st_i, c, st_j >$ and remove c from the current dependency graph
4. Redo the same process with the generated states st_j

The rescheduling automaton built from the dependency graph of Figure 5.10 is depicted in Figure 5.11.

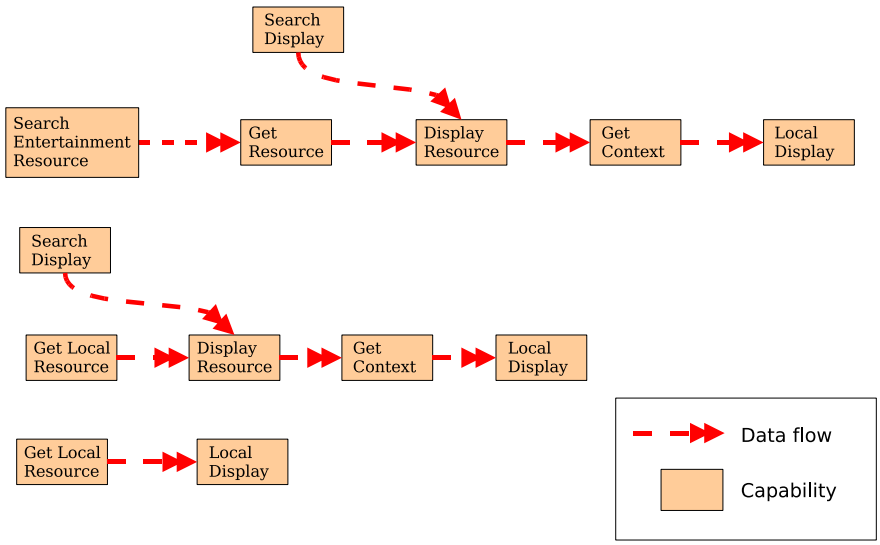


Figure 5.10: Data Dependency Graph

The rescheduling automaton can then be compared using the **ConversationMatch**() with either the raw automaton RA_{IG} or RA_{IL} to find user task realizations without or

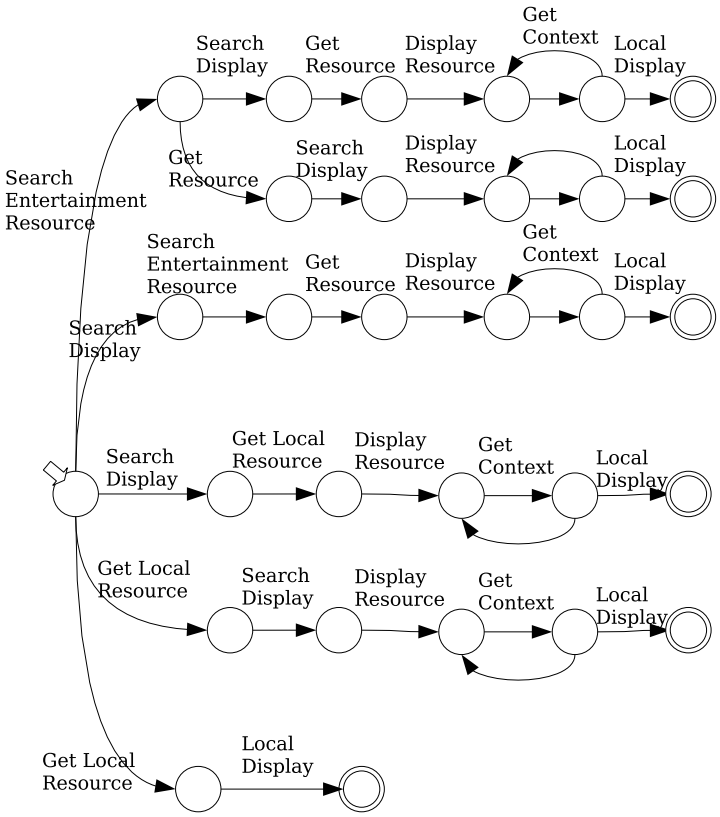


Figure 5.11: Rescheduling Automaton

with the support of conversation interleaving, respectively. This generates two additional solutions to the user task realization that support the adaptation of the user task, defined with the two functions **AdaptiveIntegration**() and **AdaptiveInterleaving**() as follows (see Figure 5.3, p. 99):

AdaptiveIntegration: $\mathcal{T} \times \mathcal{S}^{n_3} \longrightarrow 2^{\mathcal{T}}$

$< T, s_1, ..., s_{n_3} > \longmapsto \{T_1, ..., T_n\}$:

$RA_{IG} = \textbf{RawAutomaton}_{IG}(s_1, ..., s_{n_3}) \wedge$

$SA = \textbf{ReschedulingAutomaton}(T) \wedge$

$\forall T_i \in \mathcal{T},$

$A_{T_i} = \textbf{SubAutomaton}(RA_{IG})$:

$\textbf{ConversationMatch}(SA, A_{T_i})$

$$\textbf{AdaptiveInterleaving}: \mathcal{T} \times \mathcal{S}^{n_4} \longrightarrow 2^{\mathcal{T}}$$

$$< T, s_1, ..., s_{n_4} > \longmapsto \{T_1, ..., T_n\}:$$

$$RA_{IL} = \textbf{RawAutomaton}_{IL}(s_1, ..., s_{n_4}) \wedge$$

$$SA = \textbf{ReschedulingAutomaton}(T) \wedge$$

$$\forall T_i \in \mathcal{T},$$

$$A_{T_i} = \textbf{SubAutomaton}(RA_{IL}):$$

$$\textbf{ConversationMatch}(SA, A_{T_i})$$

Computed realizations will not exactly conform to the initial task conversation, but they will still fulfil the task data flow specification. While this solution increases the probability of finding service compositions, it is more costly than the first and second solutions, as the automata used as input of the **ConversationMatch**() relation are larger in terms of the number of states and transitions to be processed.

5.6 Matching Global Non-Functional Properties of Composed User Tasks

We present in this section our solution for evaluating non-functional properties of composed user tasks. This evaluation is performed by the service coordination functionality, which is responsible for assessing, simultaneously to building user task realizations, whether these realizations fulfil the user task non-functional properties. As introduced in our model for semantic service specification (see Chapter 3), non-functional properties can be of two types: *qualitative* and *quantitative*. The evaluation of qualitative properties of user tasks is straightforward, as these properties are defined semantically by referencing ontology concepts. Indeed, to assess the fulfilment of these properties, we only need to assess for each qualitative property the **ConceptMatch**() relation (see Chapter 4) with the corresponding property of each composed pervasive service. For instance, if the user task has a global non-functional property concerning the network connection, e.g., *Network is-a*

WiFi 802.11G, then each composed pervasive service must have a value of this property that matches the task's value.

Assessing the fulfilment of quantitative properties of the user task requires special care, as values of these properties provided by the composed pervasive services have to be aggregated to infer the estimated value for the composed user task. For instance, if the user task has a global non-functional property related with the execution latency, e.g., $Latency < 5$, then an aggregation of the latency values advertised by pervasive services have to be calculated following the structure of the user task conversation.

To perform this estimation, we extract from the task's conversation the mathematical formula for calculating each quantitative non-functional property. These formulae are extracted in advance and stored with the task's description.

A number of research efforts propose reduction rules to compute quantitative properties of a workflow [Cardoso et al., 2004, Menasce, 2004, Zeng et al., 2004]. We use the model proposed by J. Cardoso *et al.* in [Cardoso et al., 2004] to extract the formula of each property of the user task corresponding to the task's automaton structure. In this approach, a mathematical model is used to compute quantitative properties for a given workflow process. More precisely, an algorithm repeatedly applies a set of reduction rules on a workflow until only one atomic node remains. This remaining node contains the formula for estimating the considered property corresponding to the workflow under analysis. The algorithm uses a set of six reduction rules: (1) sequential, (2) parallel, (3) conditional, (4) fault-tolerant, (5) loop and (6) network. However, as our automata model is an abstraction of a set of elementary workflow patterns, we only need to keep the reduction rules for sequential, conditional, and loop systems.

Furthermore, we provide two estimations for each quantitative property: (1) a history-based probabilistic estimation and (2) a pessimistic estimation. The former corresponds to an average estimation, while the latter corresponds to a worst case estimation. We consider both the previous estimations, which depend on the user's task requirement (e.g., deterministic or probabilistic) in the user's request. For example, if the user demands a deterministic QoS, our approach compares the requested QoS with the pessimistic estimation of the composite service. If the user requires an average QoS, the latter is compared

against the probabilistic estimation.

Figure 5.12 and Tables 5.3 and 5.4 show how we perform these estimations. Figure 5.12 describes the reduction rules to be applied for sequence, choice and both simple and dual loop constructs. In the figure, capabilities represented on each transition (named c_i) provide some quantitative attributes (i.e., availability, latency and cost[1] noted a_i, l_i, and ct_i, respectively, in the two tables). We focus on the three dimensions: availability, latency and cost, because they are considered as important QoS dimensions of user tasks (e.g., [Cardoso et al., 2004]). Furthermore, other quantitative dimensions can be calculated in a similar way.

Beside these attributes, capabilities involved in the choice and loops constructs, provide additional information, i.e., the probability to be selected (p_i). These probabilities are only used in the case of a probabilistic estimations. The formulae to be applied in this case are described in Table 5.3. Note that in this Table, for each loop case, the probabilities p_i described in Figure 5.12, are changed to p'_i after reduction, where: $p'_i = \frac{p_i}{1-p}$. On the other hand, evaluating a worst case estimation of the quantitative properties is done by using the same reduction rules and applying the formulae described in Table 5.4. In this case some other information is required for the two loop cases that is the maximum number of times a loop has been executed. This information is represented by N in Table 5.4.

	Seq	**Choice**	**Simple Loop**	**Dual Loop**
Availability	$a_1 * a_2$	$\sum a_i p_i$	$\frac{(1-p)*a_o}{1-pa_o}$	$\frac{(1-p)*a_o}{1-a_o a_{o'}}$
Latency	$l_1 + l_2$	$\sum l_i p_i$	$\frac{l_o}{1-p}$	$\frac{l_o+l_{o'}-(1-pl_{o'})}{1-p}$
Cost	$ct_1 + ct_2$	$\sum ct_i p_i$	$\frac{ct_o}{1-p}$	$\frac{ct_o+ct_{o'}-(1-p)ct_{o'}}{1-p}$

Table 5.3: Probabilistic Quantitative Properties Evaluation

The QoS formulae for estimating the latency and availability of the user task of Figure 5.4 extracted by applying the above reduction rules are depicted in Figure 5.13.

[1]In the following, by cost we mean any resource-related cost, e.g., CPU load, memory, price.

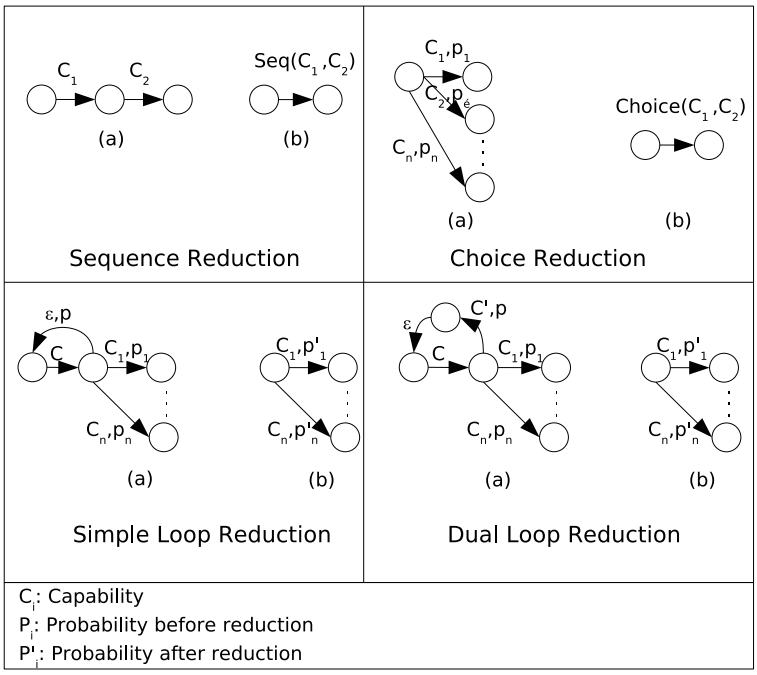


Figure 5.12: Reduction Rules for Estimating Quantitative Properties

	Seq	**Choice**	**Simple Loop**	**Dual Loop**
Availability	$a_1 * a_2$	$Min(a_i)$	$N * a_o * Min(a_i)$	$N * a_o * a_{o'} * Min(a_i)$
Latency	$l_1 + l_2$	$Max(l_i)$	$N * l_o + Max(l_i)$	$N * (l_o + l_{o'}) + Max(l_i)$
Cost	$ct_1 + ct_2$	$Max(ct_i)$	$N * ct_o + Max(ct_i)$	$N * (ct_o + ct_{o'}) + Max(ct_i)$

Table 5.4: Pessimistic Quantitative Properties Evaluation

5.7 On the fly User Task Realization for Meeting Pervasive Computing Requirements

All the presented task realization algorithms are based on the **ConversationMatch**() relation. For meeting pervasive computing requirements, this relation, which semantically compares two automata is assessed on the fly by the service coordination functionality of our SOM. This is performed by simultaneously parsing the structure of the two automata

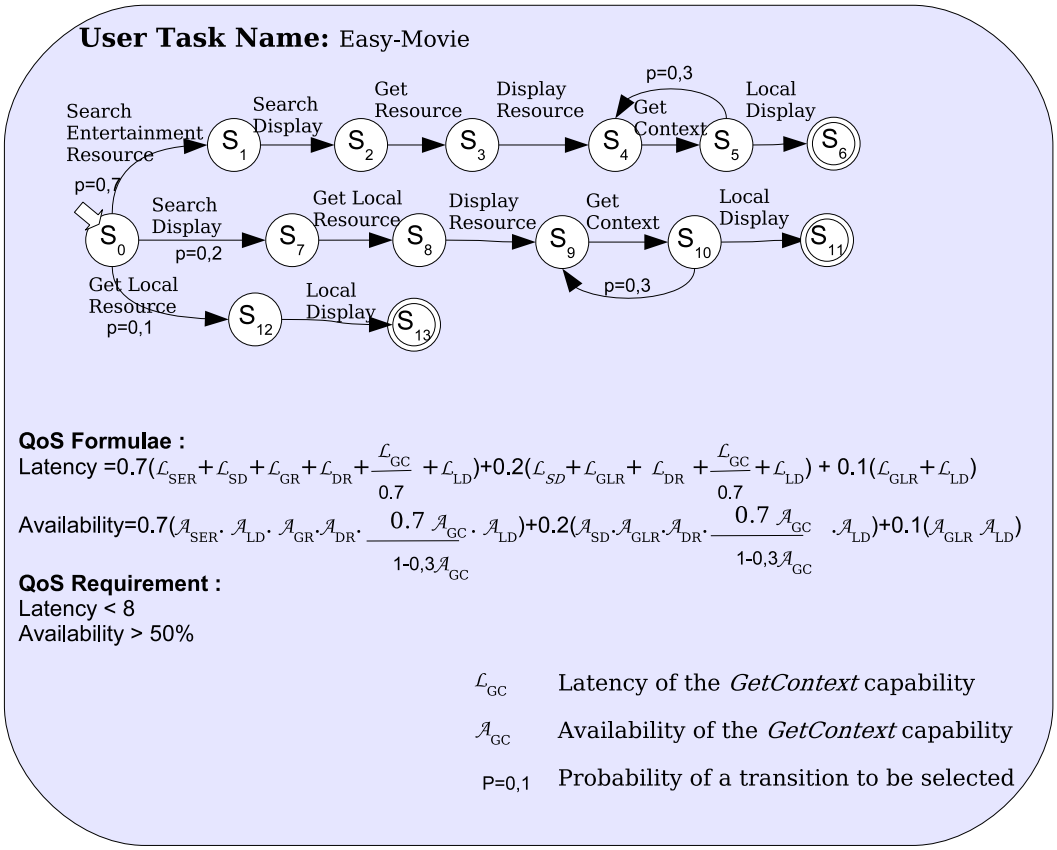


Figure 5.13: QoS Formulae of the EASY-Movie User Task

to be compared.

Specifically, the algorithm parses each state of the smallest automaton (which is assumed to be simulated by the other automaton), i.e., A_1, starting with its start state and following its transitions. Simultaneously, a parsing of the second automaton, i.e., A_2, is carried out in order to find for each state of the automaton A_1 a state of the automaton A_2 that can *simulate* it. An automaton state st_i is simulated by another automaton state sr_i when :

- For each outgoing transition of the former, i.e., $\delta_{A_1}(st_i,c)=st_j$, there is at least one equivalent outgoing transition of the latter, i.e., $\delta_{A_2}(sr_i,c')=sr_j$, such that:
  - **FunctionalCapabilityMatch**(c',c)

– **PropertiesCapabilityMatch**(c',c)

- All the states following st_i are simulated by states following sr_i

An example of using the on the fly conformance between automata for realizing a user task by integrating a set of services conversations is depicted in Figure 5.14. In this figure, the task automaton is represented on the left part of the figure, the raw automaton RA_{IG} is represented in the right part of the figure and the on the fly verification is shown step by step.

For instance, the state st_5 of the task's automaton can be simulated by the state sr_1 of the raw automaton because the set of outgoing transitions of st_5, i.e., the transition labelled with the capability $LocalDisplay$ is a subset of the set of outgoing transitions of the state sr_1, i.e., the transitions labelled with the capabilities $LocalDisplay$ and $GetLocalResource$, respectively.

Using this same algorithm and by using the simple raw automaton RA_{IG}, we can perform the user task realization with the support of conversation interleaving. This avoids building the raw automaton RA_{IL}, which results from the product of a set of service automata and may have a size that grows exponentially according to the involved automata sizes. This is done by managing service sessions. Specifically, a service session characterizes the exploration state of a service conversation (while parsing its automaton structure). A session is opened when a service conversation starts and ends when this conversation finishes. Several sessions with several pervasive services can be opened at the same time. This allows interleaving the interactions with distinct networked services. Indeed, a session opened with a service A can remain opened (temporary inactive) during the interaction of the client with another service B.

Figure 5.14 also describes the different steps performed to assess the **Conversation-Match**() relation, while managing sessions for enabling the interleaving of service conversations. An example of managing sessions is given in Step (1) of the task realization. In this step, the capability $SearchEntertainmentResource$ of the task's automaton is matched against the capability $SearchResource$ of the raw automaton provided by the $AirportEntertainmentServer$ Service. The next step is to find the capability

SearchDisplay of the task's automaton (Step (2)). However, this capability is not provided by the *AirportEntertainmentServer* service. This leads to open a session with the *SearchDisplays* service as this service provides the sought capability. In the meanwhile, the session opened with the *AirportEntertainmentServer* remains opened (vertical red arrow in the figure). After matching the capability *SearchDisplay*, the capability *GetResource* is sought in Step (3). A semantically equivalent capability, i.e., the *GetResource* capability, is accessible in the *AirportEntertainmentServer* Service from the previously opened session.

An important condition that has to be observed when managing sessions is that each opened session must be closed, i.e., it must arrive to a final state of the service automaton before the task realization is completed.

The verification of the conformance to the QoS constraints of the user task is also performed on the fly by the service coordination functionality simultaneously to the user task realization. Specifically, the service coordination functionality uses the QoS formula corresponding to each QoS metric extracted as explained in Section 5.6, and each time a service capability is being integrated, these formulae are used to check the fulfilment of the task's global QoS requirements. This verification is performed by starting with the QoS formula for each QoS dimension, in which we initially assume that all capabilities will provide the best value of the considered QoS dimension (for example, latency = 0, availability = 1). Then, each time we examine a service capability, we replace the corresponding best value in the formula of the considered dimension, with the real QoS value of the capability. This allows evaluating at each step of the integration the values of all QoS dimensions in the case that the current capability is selected. These values are then compared to the corresponding values required by the user task, and if the constraints are not met, the path in the global automaton that includes this capability is rejected.

The service coordination functionality gives a set of sub-automata from the raw automaton that conforms to the task's automaton structure. Each of these automata is a user task realization that conforms to the conversation of the target user task in terms of functional and non-functional properties, further enforcing valid service consumption.

Figure 5.15 gives an example of the on the fly verification of the task's QoS requirements

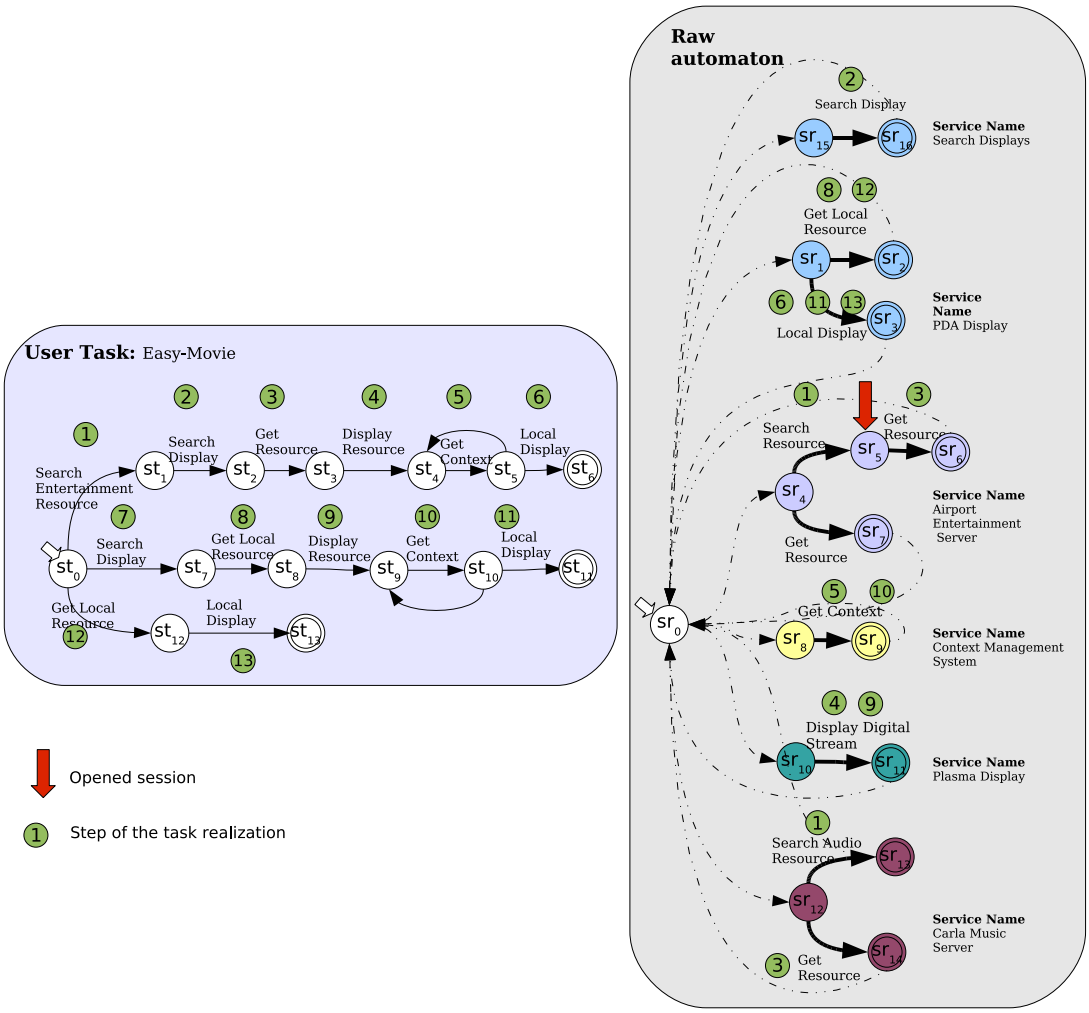


Figure 5.14: On The Fly User Task Realization

and the generation of user task realizations. In this figure each pervasive service provides (right part of the figure) an estimation of its provided *Latency* and *Availability* QoS dimensions. The user task, has two QoS requirements i.e., $Latency < 8$ and $Availability > 50\%$ (left higher part of the figure). There are two resulting user task realizations (left middle and left lower parts of the figure) that both fulfil the task non-functional properties.

Once the set of possible task realizations is given, the ordering of these realizations is performed using the **TaskDoM**() function defined in Section 5.5.1. The best among these realization is then returned to the client in the form of an executable description. The

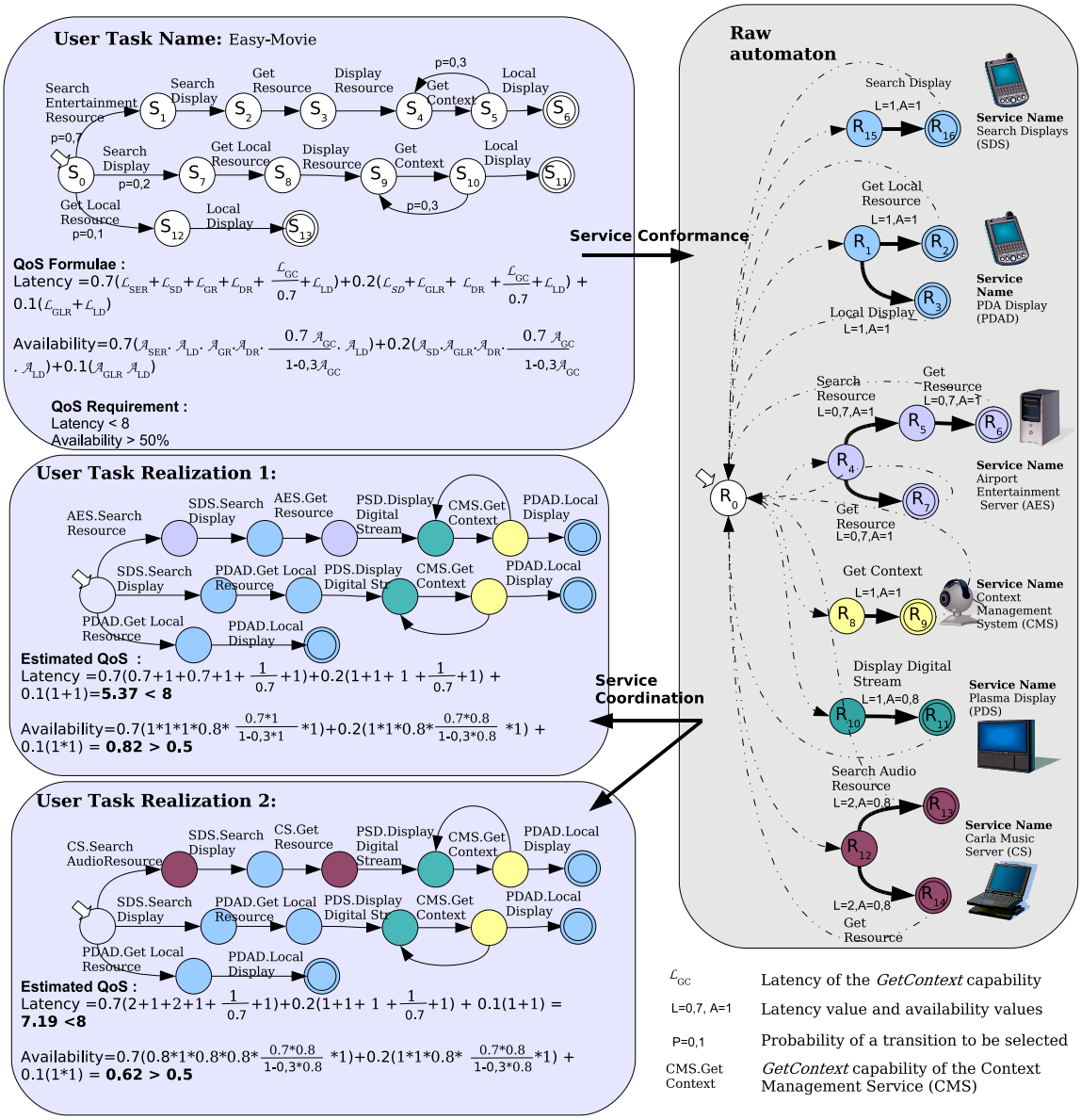


Figure 5.15: QoS-aware User Task Realization

client finally executes this realization by using a local or a remote execution engine.

5.8 Assessing the Efficiency of the Composition Model

We discuss in this section the computation cost of our overall service composition SOM functionality. This functionality decomposes in different other functionalities. First, the computation cost of the service discovery client depends on the efficiency of the underlying semantic service registry, which is discussed in Chapter 4. Then, the complexity of all the other functionalities (including the various service conformance and coordination functions) mainly depends on the complexity of the **ConversationMatch**() relation. The complexity of this relation can be approximated by process algebra simulation (bisimulation) algorithms that assess the conformance of two processes by comparing their corresponding automata. A detailed analysis of the cost of process simulation (bisimulation) algorithms is described in [Moller and Smolka, 2003]. Specifically, bisimulation between two non-deterministic finite state automata that have a total of n states and m transitions can be decided in polynomial time, O(nm) time to be exact [Kanellakis and Smolka, 1990]. This result was subsequently improved upon in [Paige and Tarjan, 1987], where an algorithm that runs in O(m log n) time has been defined. The condition to be observed in both cases is that the two automata to be compared can be represented with right-linear grammars. We prove in this thesis (Appendix A) that all the automata (of the user task and pervasive services) generated using our rules for mapping basic control patterns to finite state automata defined in Chapter 3 can be represented with right-linear grammars. Hence, the complexity of our **ConversationMatch**() relation can be approximated with the complexity of the process bisimulation algorithm defined in [Paige and Tarjan, 1987], which is in the order of O(mlog n) for two automata that have a total of n states and m transitions. Consequently, our solution to the dynamic user task realization, which essentially relies on the **ConversationMatch**() relation, performs better than existing related work in the area of conversation-based service composition ([Berardi et al., 2003]), which rely on exponential time verification algorithms.

From this statement, we can now discuss the efficiency of the service conformance and coordination functionalities. As the cost of the **ConversationMatch**() relation grows (linearly) with the size of the compared automata it is obvious that the selection and

task realization algorithms that support the interleaving of service conversations, i.e., **OrderingConstraintSelection**$_{IL}$(), **ConversationInterleaving**() and **AdaptiveInterleaving**() are more costly than the functions **OrderingConstraintSelection**$_{IG}$(), **ConversationIntegration**() and **AdaptiveIntegration**(), as their employed filtering and row automata, i.e., FA_{IL} and RA_{IL}, respectively, are larger. Consequently, a client may choose the most appropriate task realization solution according to its available resources.

Another point that we can discuss is the impact the selection process performed by the service conformance functionality regarding ordering constraints. This functionality assesses the **ConversationMatch**() relation for each service returned by the service discovery client. This verification can be avoided as the service coordination also assesses the **ConversationMatch**() relation with the raw automaton, which is composed of a set of pervasive services. However, verifying the compatibility of service ordering constraints priori to the task realization phase can be performed in a distributed manner by the distributed instances of the middleware that host the service registry, contrary to the task realization that have to be performed in a single node. Hence, suppressing the pre-selection phase implies that a potentially larger number of service descriptions may be sent to the node that performs the service integration, which may overload the network on the one hand, and generate an additional overhead on the **ConversationMatch**() relation on the other hand.

To complement this theoretical assessment, a practical assessment evaluating the performance of the on the fly user task realization, with and without QoS-awareness, in terms of execution overhead is presented in Chapter 6.

5.9 Concluding Remarks

We presented in this chapter our solution to the dynamic user task realization. This solution supports the integration of service conversations to realize the conversation of a user task, providing the user with four different levels of flexibility. It further supports the semantic specification of service and task capabilities and enable QoS-aware service composition.

The theoretical assessment of our solution shows that the cost of our algorithms grows linearly with the size of the automata to be compared, which is more efficient than existing related work in the area of conversation-based service composition. This result is further consolidated with a practical assessment provided in the following chapter.

Chapter 6

PERSE: Pervasive Semantic-aware Middleware

We present in this chapter the PERvasive SEmantic (PERSE) middleware, which provides a comprehensive solution for service discovery and composition in pervasive computing environments. This middleware integrates the SOM functionalities presented in this thesis, i.e., service publication, location, matching (Chapter 4) and service composition (Chapter 5), complemented with multi-network and multi-protocol management provided by the MUSDAC middleware [Raverdy et al., 2006].

The remainder of this chapter is structured as follows. First, we present in Section 6.1 an overview of the MUSDAC middleware. Then, we introduce in Section 6.2 the PERSE middleware. Finally, we present a prototype implementation and performance evaluation of this middleware in Section 6.3.

6.1 Baseline MUSDAC Middleware for Multi-Network, Multi-Protocol Service Discovery in Pervasive Computing Environments

The MUSDAC middleware enables multi-network, multi-protocol service discovery and access in pervasive computing environments. For multi-network management, MUSDAC

dynamically composes nearby networks through application-level routing components provided on devices having multiple network interfaces, which enables the dissemination of service location and access requests in the whole environment.

Multi-protocol interoperability decomposes into service discovery protocol interoperability and service access protocol interoperability. Service access protocol interoperability is performed by translating service access messages from one protocol to another. Service discovery protocol interoperability decomposes in two parts, i.e., the translation between protocol messages and the translation of heterogeneous service advertisements into a common XML format (the MUSDAC service description format).

As depicted in Figure 6.1, the MUSDAC architecture is composed of three main components:

- The Manager deals with service publication for local service providers, i.e., providers that reside on the same network, and performs service location, matching and access for local and remote service requesters.

- Service Discovery and Access (SDA) Plugins allow the interaction with service providers and requesters using specific service discovery protocols to collect service information and perform service access.

- Bridges interconnect diverse networks accessed through the network interfaces of a device, and manage the dissemination of service location requests as well as the access to remote services.

While MUSDAC constitutes an innovative solution for multi-network multi-protocol service discovery in pervasive computing environments it focuses on interoperability among syntactic SDPs. Furthermore, MUSDAC relies on a proprietary service description format that does not support the specification of service non-functional properties and does not support service composition. We introduce in the following section a middleware that integrates the functionalities provided by MUSDAC into a PERvasive SEmantic-aware middleware.

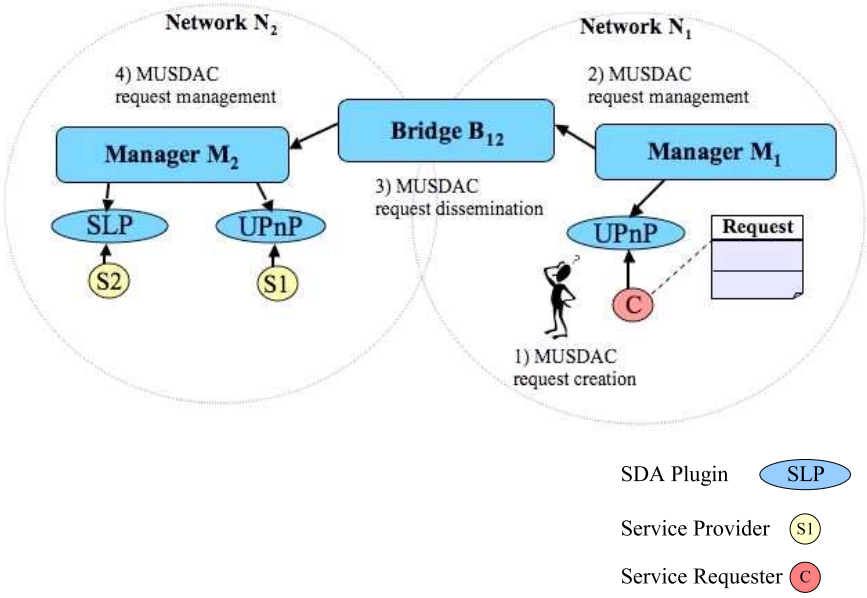


Figure 6.1: MUSDAC Architecture

6.2 The PERSE Middleware Architecture

We present in this section, the architecture of the PERSE middleware, and its deployment. As depicted in Figure 6.2, PERSE is composed of two main layers: the communication middleware layer and the semantic SOM functionalities layer. The semantic SOM layer implements the SOM functionalities presented in Chapter 4 and Chapter 5 of this thesis. The communication middleware deals with multi-network, multi-protocol service discovery and access.

The deployment of PERSE builds upon the deployment mechanism of the MUSDAC middleware. Specifically, PERSE registers as a service, i.e., the *PERSE Service*, using each service discovery protocol available in the local network. Then, client applications explicitly interact with the PERSE Service using their preferred discovery and access protocol. The PERSE Service interface is composed of a set of capabilities enabling a client to perform a semantic-enhanced service discovery and to realize user tasks. It also allows service providers to publish semantic-enhanced service advertisements. Providing such an explicit interface enables the extension of existing protocols with new features such

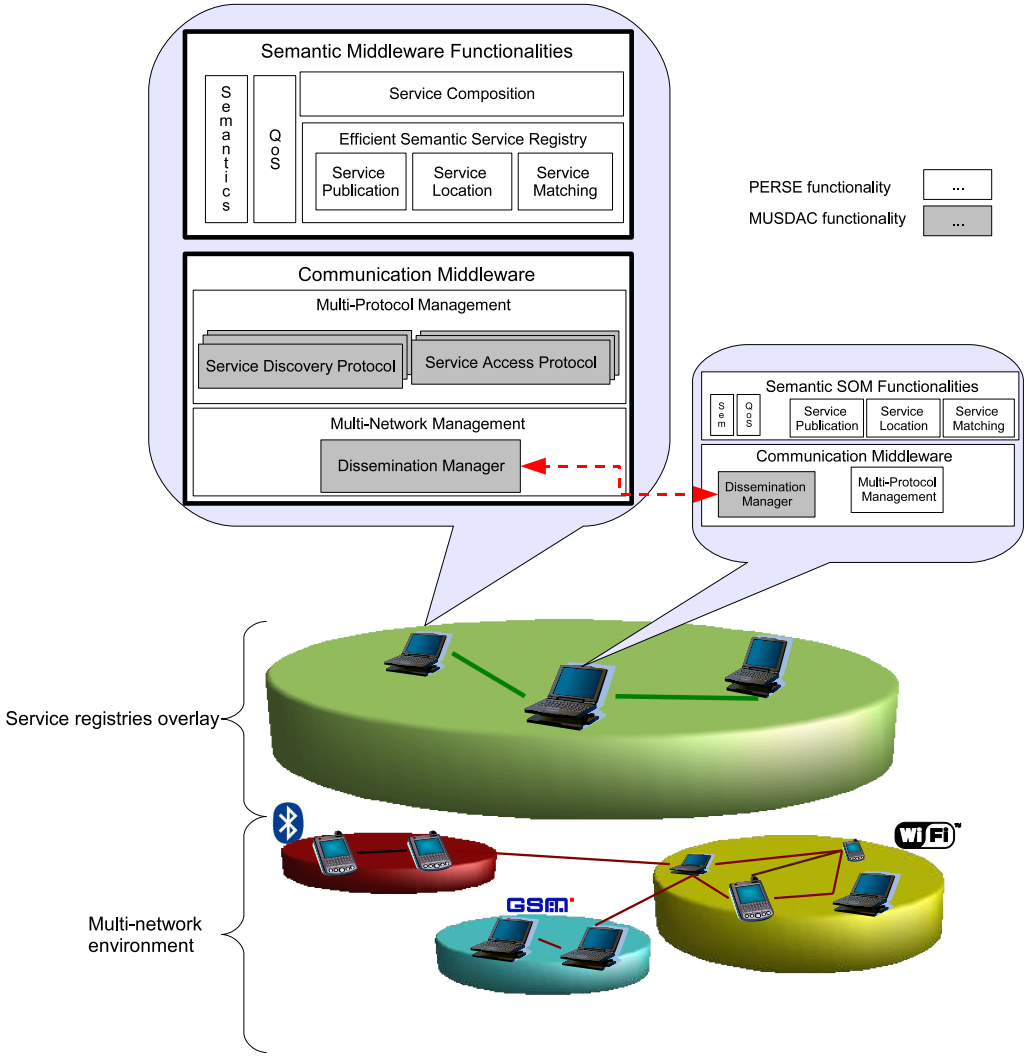


Figure 6.2: PERSE Middleware

as support of rich semantic annotations (e.g., support of annotation type) QoS-awareness as well as service composition.

A node that hosts the PERSE Service is dynamically elected to control the network. Specifically, PERSE-aware devices exchange their profile information and then use a multi-criteria algorithm to elect their PERSE Service (benefit value based on number of SDPs supported, device expected lifetime, device processing capabilities). Once elected, the PERSE Service periodically sends presence beacons so that other PERSE-aware devices

in the network can detect its absence (i.e., no presence beacon received after a given time) as well as duplicates, and elect a new PERSE Service.

6.2.1 Multi-Network Management

Network heterogeneity leads to many independent networks being available to users at a location, which can be loosely interconnected with today's mobile devices. Innovative solutions are then required for the efficient inter-network publication, location and matching of service requests and advertisements. To deal with multi network management, PERSE builds upon the MUSDAC multi-network management functionality. Specifically, PERSE operates independently in each network of the pervasive environment, and each instance of PERSE selects with which other instances (in nearby/distant networks) to interact with. PERSE instances residing in different networks communicate with each other using their *Dissemination Manager*, i.e., Bridge in MUSDAC.

6.2.2 Service Discovery in PERSE

The PERSE Service translates service advertisements to a common language, i.e., the Interoperable Service Description Language (ISDL) described in Section 6.2.5. For each legacy SDP, the translation is performed by its corresponding plugin (in communication middleware layer). Then the generated service description is stored in the local service registry (semantic SOM layer), which performs local service publication (as described in Chapter 4). A client looking for a service in the pervasive environment first discovers the PERSE Service using its preferred SDP and sends its service request expressed using ISDL. Upon the reception of a service request PERSE performs local service location implemented by the semantic service registry (semantic SOM layer). Also, PERSE uses the dissemination manager (multi-network management) to propagate the service request to nearby networks. A PERSE Service receiving a remote service request processes it as a local one (i.e., performs local service location and disseminates the request to the other reachable networks) but returns the results to the originating dissemination manager. Finally, local PERSE Service in contact with the client collects local and remote results

and returns them to the client.

6.2.3 Service Composition in PERSE

Clients that want to realize an abstract user task available in their device send a composition request to the PERSE Service. The PERSE Service performs local and remote service location (if needed) to pre-select a set of services candidate to the composition. An executable description of the user task is then generated by the PERSE Service using the service conformance and coordination functionalities as presented in Chapter 5. The execution of this task can then be performed by an execution engine available either on the user's device or hosted in another node in the network.

6.2.4 Service Access in PERSE

PERSE integrates the service access functionality provided by MUSDAC. It supports client access to services hosted in remote networks and assume that both clients and services use SOAP as access protocol (e.g., UPnP, Web services). In this case, message translation is simplified, as only the only the message headers needs to be modified (for managing application level routing) while the content of the access message remains the same. Accessing a remote service via PERSE is performed through the creation of a communication channel. The creation of this channel is transparent to the client as it is done when accessing the remote service for the first time. When the client initiates an interaction with a service, it uses a local address that have been added to the service description by the PERSE Service, instead of the target service address. The communication channel is composed of the client address, the local address (provided by the PERSE service) and the dissemination list, i.e., the list of all the bridges in between and the target service. Once created, messages from the client are translated (i.e., change in the header of the SOAP message) and encapsulated in a message sent over the communication channel. Each dissemination manager that receives an access message checks the unique identifier of the communication channel for this message and forwards it until it reaches the target service. The result is returned in a similar way to the client.

6.2.5 Interoperable Service Description Language

SDP interoperability is achieved in PERSE through the translation of heterogeneous service advertisements to a common language. Based on the conceptual model presented in Chapter 3, we define the Interoperable Service Description Language (ISDL) as a concrete realization of this model. For the implementation of ISDL, we opted for an XML-based schema defining a container, which is combined with the two emergent standard service description languages namely SAWSDL and WS-BPEL. ISDL is not yet another service description language. It acts primarily as a top-level container for additional files describing facets of the service. SAWSDL is used to describe the capability interfaces, while WS-BPEL is used to express conversations associated with capabilities. We employ SAWSDL for the definition of capability interfaces, as it supports both semantic and syntactic specification of service attributes (e.g., inputs, outputs). Thus, both legacy syntactic descriptions and rich semantic descriptions can be translated to SAWSDL. On the other hand, WS-BPEL is a comprehensive language for workflow specification, which is adequate for conversation specification. It has largely been adopted both in the industrial community and in academia. WS-BPEL supports only syntactic conversation specification, however, if combined with SAWSDL, semantic conversations can be defined. Additional files may be optionally linked to the ISDL container to describe a service's non-functional properties using existing QoS models (e.g., SLAng[1], EASY [Ben Mokhtar et al., 2007b]). Figure 6.3 shows an example of a ISDL description. In this example, the service is composed of two capabilities. The first capability has a complete functional and non-functional description that comprises a reference to a SAWSDL file defining the capability interface, a WS-BPEL description that defines the conversation associated with the capability, as well as a QoS description given in a SLAng and an OWL file respectively. The second capability of this service is only given with an interface description defined in a SAWSDL file.

One of the particular features of ISDL is the support of heterogeneous service description languages. This is realized through the translation of the incoming heterogeneous service descriptions into ISDs. These descriptions are then stored by the service registry

[1] http://www.cs.ucl.ac.uk/staff/j.skene/slang/

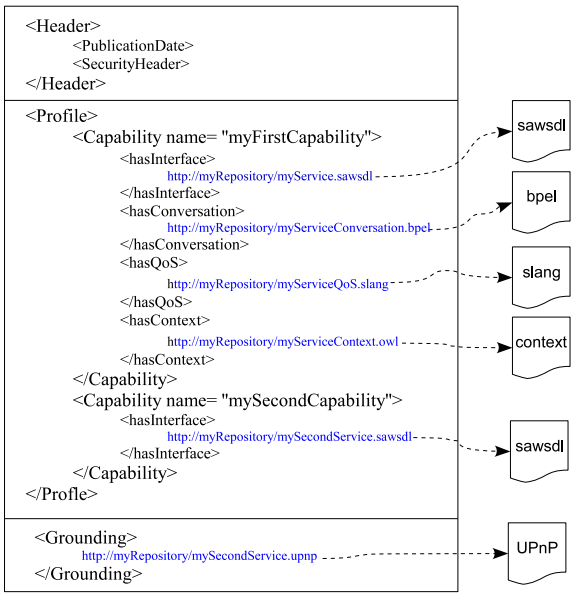


Figure 6.3: Example of an ISD Description

and used to assess the conformance with incoming service requests. Figure 6.4 gives an overview of how various legacy service descriptions are translated to ISDL. In this figure five different scenarios are identified.

The first scenario describes the case of a legacy service specified with the name of its provided functionality (e.g., a SLP service). In this case, the SLP2ISDL plugin translates the SLP description to a ISD description. This description contains the SLP grounding information and links to a SAWSDL description that contains a single operation having as name the name of the SLP service without any input and output specification.

The second scenario describes the case of a service that provides a list of operations described syntactically with their signatures, as it is the case for UPnP services or Web services. In this scenario, the corresponding plugin (e.g., UPnP2ISDL or WSDL2ISDL) translates the given description to a ISD description, which links to a SAWSDL description that comprises a list of WSDL operations corresponding to the operations specified in the legacy description without semantic annotations.

The third scenario describes the case of a service described as a set of semantically annotated operations (e.g., given as a SAWSDL description). In this case, the mapping is

straightforward as it consists of linking the ISD description to the given SAWSDL file or to map the terminology of the given file to SAWSDL if different.

The fourth scenario describes the case of a syntactic capability described with an associated conversation of operations (e.g., a service described as a WSDL operation that is realized through the execution of a WS-BPEL conversation). In this case the ISD description contains the specification of both a ISD interface and a conversation. The ISD interface points to a SAWSDL description that contains a single operation without semantic specification used to describe the capability. On the other hand, the ISD conversation links to a WS-BPEL description that describes the conversation associated with the operation. This WS-BPEL description uses itself another WSDL file that specifies the operations used in the conversation.

The last scenario describes the case of a semantic capability having an associated conversation of semantic operations (e.g., an OWL-S service with a profile that describes the semantic capability and a process model that describes the associated conversation). In this case, the generated conversation also comprises both a ISD interface and a ISD conversation. However, compared with the previous case, the SAWSDL description used to describe the capability comprises semantic annotations of the capability elements (i.e., inputs, outputs). Furthermore, the WS-BPEL file describing the conversation associated with the capability uses another SAWSDL description in which the operations are also semantically annotated.

To perform efficient semantic service matching, SAWSDL descriptions attached with a ISDL descriptions are pre-encoded. We developed an application that allows generating encoded SAWSDL descriptions. This application helps a service developer in semantically annotating its service descriptions by graphically loading ontology description files. It further implements the prime-number based encoding and allows the integration of codes associated with ontology concepts as part of the generated SAWSDL description. The GUI of this application is shown in Figure 6.5.

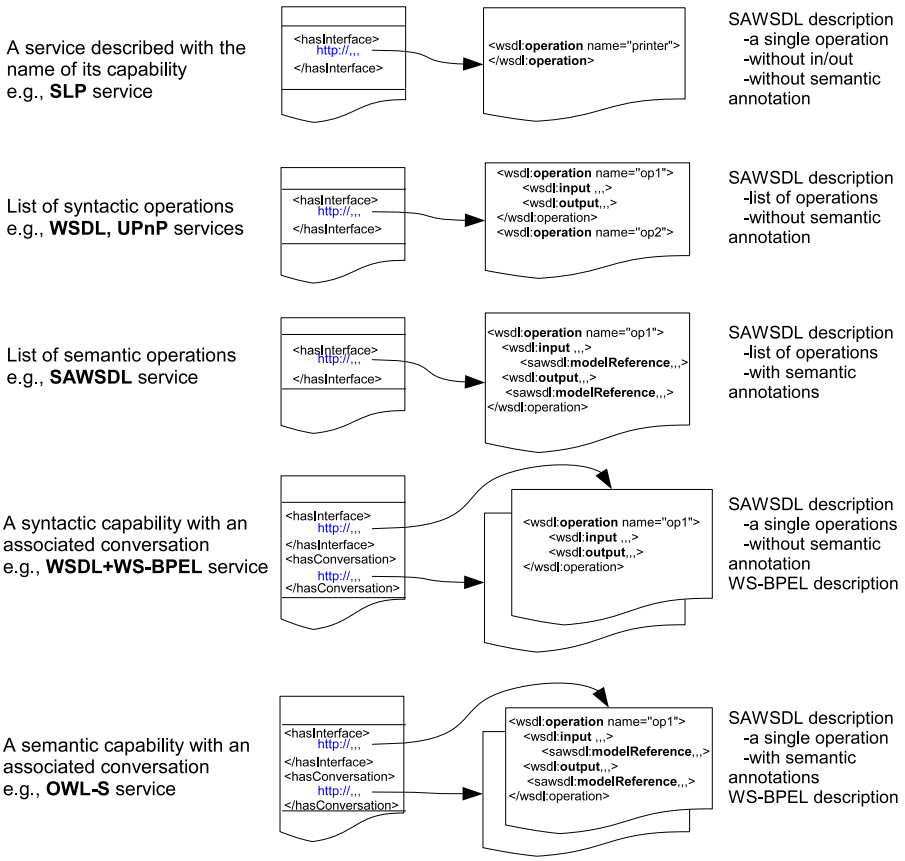


Figure 6.4: Interoperability Enabled by ISDL

6.3 Prototype Implementation and Performance Evaluation

We have implemented a prototype of PERSE using Java 1.5. Selected legacy plugins have been developed for SLP using jSLP, UPnP using Cyberlink, and UDDI using jUDDI.

To evaluate the efficiency of PERSE, we have been interested in the evaluation of:

1. The performance of the prime-number based ontology encoding algorithm in terms of code lengths in Section 6.3.1. Indeed, as service descriptions are to be stored in devices with potentially limited storage capabilities, we have been interested in the comparison of code lengths generated by our employed prime-number based encoding algorithm compared to other existing encoding algorithms.

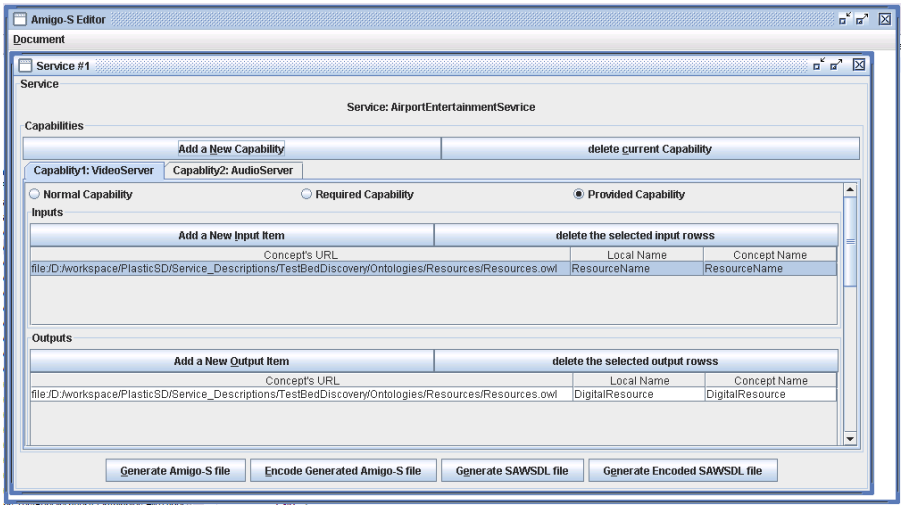


Figure 6.5: SAWSDL Editor

2. The processing time to create ISDL descriptions as a result of the translation of a legacy description, in Section 6.3.2. This evaluation estimates the cost of interoperability realized through language to language translation.

3. The cost of semantic service matching performed using online reasoning on ontologies compared with our efficient interoperable service matching in Section 6.3.3. We further present in this section the processing time of matching various combinations of heterogeneous service requests and descriptions as well as the scalability of our service matching function.

4. The efficiency of the PERSE semantic service registry in Section 6.3.4. Specifically, we have been interested in the evaluation of the time to organize the semantic service registry, the time to publish and locate a semantic service description as well as the comparison of the scalability of our registry compared with a WSDL service registry.

5. The processing time for service composition with and without the support of QoS in Section 6.3.5.

Ontology	Classes	Caseau	Krall	Prime Max/Avg
SUMO	630	48	30	83 / 42
Wine & Food	133	39	33	53 / 23
Pizza	99	40	37	40 / 23
Gene Ontology	20945	2155	151	361 / 82
Java 1.30	5438	1568	68	112 / 31
OpenCyc	25565	1420	350	681 / 272

Table 6.1: Comparison of Encoding Length of a Single Class (in bits)

6.3.1 Performance of the Prime Number-Based Ontology Encoding Algorithm

We have been interested in this experiment by the performance of the prime number based encoding algorithm in terms of code lengths compared to other encoding algorithms. We have performed the encoding of a set of multiple ontologies, including: the Suggested Upper Merged Ontology [Niles and Pease, 2001]; the OpenCyc upper ontology[2]; several well-known ontology tutorial examples[3],[4]; the Gene Ontology[5], which provides a vocabulary of genes from any organism; and the Java 1.30 types hierarchy, which is part of a subtyping benchmark[6]. Table 6.1 provides an overview of the code lengths achieved by various encoding algorithms described in Chapter 4 and the prime number based algorithm. The results for existing algorithms show the largest encoding length for a class in the hierarchy, expressed in bits. For the binary matrix method, this is equal to the size of the hierarchy. For the prime-based algorithm, the last column shows: the largest encoding lengths for the heuristic that minimizes the largest encoding length; and the average encoding length for the heuristic that minimizes the total encoding length. Besides achieving conflict-free incremental encoding, the encoding lengths produced by the prime number based algorithm are comparable if not better than the ones of existing algorithms.

[2]OpenCyc: http://www.opencyc.org/
[3]OWL Guide: http://www.w3.org/TR/owl-guide/
[4]Pizza Ontology: http://www.co-ode.org/ontologies/pizza/
[5]Gene Ontology: http://www.geneontology.org/
[6]Java subtyping benchmarks: http://www.zibin.net/subtyping-benchmarks.html

6.3.2 Cost of Legacy to ISDL Translation

We have been interested in this experiment by measuring the cost of multi-protocol interoperability, and particularly the cost of translating legacy service descriptions to ISDL descriptions. This experiment has been carried out on a Windows XP PC with a 2.6 GHz processor and 512 MB of memory. Results presented below are the average of ten runs. The standard deviation for the results presented in this experiment is negligible (less than 1%). As presented in [Raverdy et al., 2006], providing interoperability on top of simple, limited SDPs such as SLP may incur a significant overhead (i.e., overhead of over 200 milliseconds for a native discovery time of less than 1 millisecond for a similar configuration). It was analyzed that this overhead was by and large (two-thirds or almost 140 milliseconds) triggered by the SOAP-based interface of the interoperability service. This overhead however becomes negligible when interoperating with other SD service such as UDDI that have a native discovery time between 1 and 6 seconds).

For our PERSE prototype, the processing time for the translation of service descriptions (requests and advertisements) from selected legacy SDPs to ISDL descriptions are provided in Table 6.2. The first line of this table represents the time to process a discovery request using SLP, UPnP and WSDL excluding the time to parse XML descriptions. The second line represents the time to process a discovery request in addition to the time to translate the request to ISDL. Finally, the third line represents the overhead of the translation. In this experiment times are given in micro-seconds.

As can be observed, the overhead of the translation to ISDL increases with the complexity of the original description, and in particular the complexity and size of the original XML data to process. Nevertheless, the overhead for translating WSDL descriptions to ISDL is less than the overhead for translating UPnP description. This is due to the similarity between SAWSDL and WSDL, which eases the translation process. It should be noted that, in the case of UPnP and WSDL, the libraries used by the legacy plugins already retrieve and parse all the necessary information (e.g., the device and service XML descriptions in UPnP). All the processing in the legacy plugins is thus performed in memory. Overall, the translation time is not significant (tens to hundreds of micro-seconds)

compared to the overall discovery time.

	SLP	UPnP	WSDL
Discovery Request	22.8	32.4	243
Discovery Request+Translation to ISDL	23.4	85.1	287
Overhead of the translation	0.6	52.7	44

Table 6.2: Legacy to ISDL Translation (in micro-seconds)

6.3.3 Performance of the Interoperable Service Matching

We present in this section three main experiments. The first experiment measures the performance of semantic service matching performed using online semantic reasoning. The second experiment measures the performance of the interoperable service matching performed in PERSE between heterogeneous service advertisements and service requests. The last experiment measures the scalability of a PERSE service registry. These three experiments have been carried out on a Windows XP PC with a 2.6GHz processor and 512 MB of memory.

The first experiment, which measures the performance of semantic matching using online reasoning, includes the use of three DL-reasoners to infer the subsumption relationships between concepts, i.e., Racer[7], FaCT++[8] and Pellet[9]. We provide this evaluation employing each one of the aforementioned three reasoners in order to assess their impact on the semantic matching process.

This experiment provides the time taken by each reasoner to match the concepts involved in a single service request and a single service advertisement for an increasing number of concepts. Both the request and the advertisement use the Pizza ontology[10]. This ontology contains 99 OWL classes, 4 datatype properties, 11 object properties, 24 annotation properties and 5 individuals.

[7]Racer: http://www.sts.tu-harburg.de/ r.f.moeller/racer/

[8]FaCT++: http://owl.man.ac.uk/factplusplus/

[9]Pellet: http://www.mindswap.org/2003/pellet/

[10]http://www.co-ode.org/ontologies/pizza/

In this experiment, we increase the number of concepts involved in the service request from 4 to 14. The time measured time includes (1) the time to parse the service advertisement and the service request; (2) the time to load to the reasoner and classify the ontologies involved in the service advertisement and request descriptions; and (3) the time to match the concepts involved in the advertisement and the request, i.e., to assess the relations between these concepts within the classified ontologies.

Figure 6.6 shows the results of this experiment. We notice that for all the three reasoners the processing time increases proportionally to the number of concepts involved in the service description and service request. Furthermore, the matching time for all the three reasoners is in the order of 4 to 6 seconds. In all the cases, the most expensive phase is the one of loading and classifying the involved ontologies (in this case a single ontology): from 76% to 78%. From this experiment we conclude that the semantic matching using online semantic reasoning is a very heavy process. Let's compare it with our efficient semantic service matching performed in PERSE.

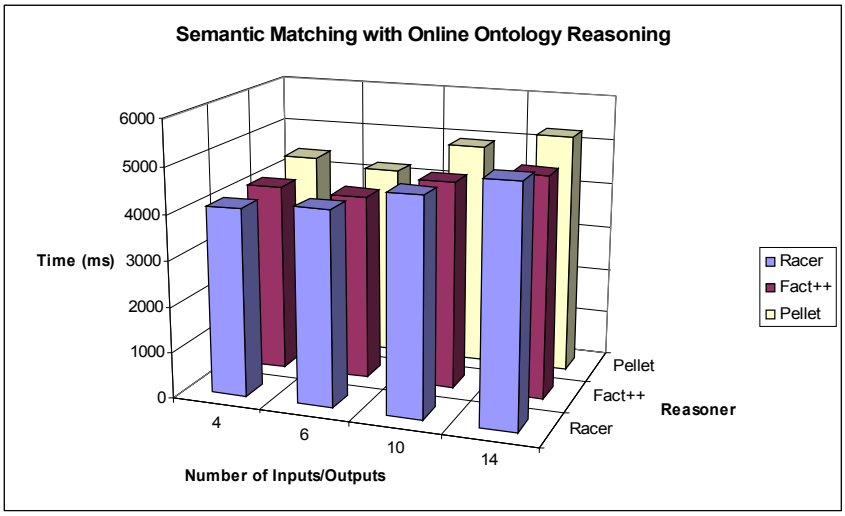


Figure 6.6: Matching Using Online Semantic Reasoning

The second experiment measures the processing time of the matching algorithm performed in PERSE for different combinations of service requests and advertisements. Re-

sults are depicted in Figure 6.7. In this experiment, the registry contains only one service having one capability with a single input and a single output (except for SLP services that are only described with capability names). For each couple service advertisement/service request, we measure the time to parse the two descriptions and the time to assess the matching between them.

We can notice that there are three different cases of matching depicted in the figure with three different coloured zones:

- syntactic matching of capability names performed when a SLP description is matched against any other description, i.e., first line and first column of the table.
- syntactic matching of capability signatures performed when both the advertisement and the request are defined as a set of capabilities and one (or both of them) does not have a semantic specification, i.e., the rest of the table except the bottom right double-cell.
- efficient semantic matching, performed when both the service advertisement and the service request have a pre-encoded semantic specification, i.e., the bottom right double-cell of the table.

From the results of this experiment we can notice that:

- The time to parse service and request descriptions is almost the same for all the kinds of descriptions, because they are all ISDL descriptions.
- Syntactic matching based on capability names is the most efficient, which is due to the fact that there are less information to compare (only capability names)
- Thanks to our encoding mechanism, semantic matching in PERSE performs as efficiently as syntactic matching of capability signatures.
- Our efficient semantic matching performed in PERSE is around 2500 time faster than semantic matching based on online reasoning on ontologies

Overall, it can be concluded that as for cost interoperability, the cost of the matching algorithm performed by PERSE is also negligible when compared to the total discovery time (and in particular the processing time for SOAP communication).

		REQUESTED SERVICE							
		ISDL SLP		ISDL UPNP		ISDL WSDL		ISDL SAWSDL	
		Time* to Parse	Time to Match	Time to Parse	Time to Match	Time to Parse	Time to Match	Time to Parse	Time to Match
ADVERTISED SERVICE	ISDL SLP	17.98	9.2	18.44	9.0	18.32	8.9	18.94	11.3
	ISDL UPNP	19.07	10.7	18.68	18.0	18.59	16.9	19.23	18.8
	ISDL WSDL	18.82	10.3	18.29	18.1	18.22	17.4	18.96	19.0
	ISDL SAWSDL	18.88	10.9	18.51	19.3	18.41	18.8	19.10	21.3

Syntactic matching of capability names

Syntactic matching of capability signatures

Efficient Semantic Matching

* All times are in micro-second

Figure 6.7: PERSE Matching Performance

The third experiment that we present in this section measures the scalability of our semantic service matching performed in a PERSE service registry. In this experiment we increase the number of services in the registry from 1 to 128 services of different types (SLP, UPnP, WSDL and SAWSDL) and we perform the matching between a service request and services of the same type. All the times are in milliseconds and do not include the time for parsing service descriptions.

In this experiment, semantic services (i.e., described using SAWSDL) are organized into graphs of similar capabilities as presented in Chapter 4. In this experiment we have been interested in two extreme scenarios of registry organization: the case where all the semantic services are semantically different from each other, i.e., the registry is not organized (curve *SAWSDL Worst*) and the case where all the services are semantically equivalent to each other, i.e., there is a single graph, with a single node that contains all the service capabilities (curve *SAWSDL Best*). A case where the registry is partially grouped (real case scenario) would be represented with a curve between these two extreme curves.

The results of this experiment are depicted in Figure 6.8. From this experiment we can notice that matching cost increases substantially when registry holds more than 100 services for the cases of UPnP, WSDL and unorganized semantic services (SAWSDL worst case). Nevertheless, when semantic services are related to each other (SAWSDL best case), thanks to our grouping algorithm, the scalability of the semantic matching is similar to the scalability of SLP based matching.

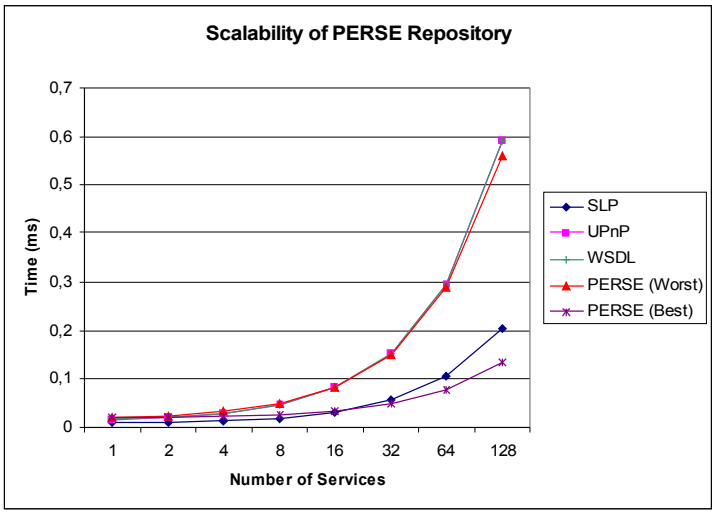


Figure 6.8: PERSE Registry Scalability

6.3.4 Efficiency of the PERSE Service Registry

We have considered in the last experiment of the previous section the two extreme cases of semantic service grouping, i.e., all the services of the registry are similar or all the services are different. This allowed us to determine an interval in which a normal case scenario of semantic service grouping would be situated. We focus in this section on the performance of a fully semantic PERSE registry where services are assumed to be grouped according to a specific grouping scenario. Specifically, we consider from 1 to 100 semantic services using 22 different ontologies and grouped in 12 groups of various sizes.

We conducted four different experiments to evaluate:

1. The processing overhead for organizing the registry.
2. The time for service publication into the organized registry.
3. The time for service location in the organized vs unorganized registry.
4. The scalability of our semantic service registry compared with a classical WSDL registry.

These four experiments have been conducted on a Toshiba Satellite notebook with a 1.6 GHz Intel Centrino processor and 512 MB of RAM. Note that for all these four experiments each value is calculated from an average of ten runs.

Figure 6.9 shows the results of our first experiment, which evaluates the time to create graphs of services in an empty registry. A scenario for this experiment would be realized when a registry leaves the network and when another one is elected and has to host the set of service descriptions available in its vicinity. Figure 6.9 shows three measurements: (1) the time to parse the service descriptions; (2) the time to organize the service capabilities into graphs; and (3) the total time, i.e., time to parse and create the graphs. From this figure, we notice that the time to create the graphs is negligible compared to the time to parse service descriptions, i.e., XML parsing time, which is mandatory due to the use of Web services and Semantic Web technologies.

The results given by the second experiment that we performed are depicted in Figure 6.10. This experiment shows the time to insert a new service advertisement in a registry. This figure shows 3 measurements: (1) the time to parse the ISDL description of the new service; (2) the time to classify the service capabilities within the registry graphs; and (3) the total time, i.e., the time to parse and classify the service capabilities. Results show that the time to classify service capabilities in a set of existing graphs is negligible compared to XML parsing time of the service description. We also notice that this time is nearly constant. This is due to the fact that the number of semantic matchings performed in the registry in order to insert a capability depends neither on the total number of services on the registry nor on the number of graphs. The time to insert a capability depends on the number of root and leaf nodes in the registry graphs as well as the number of

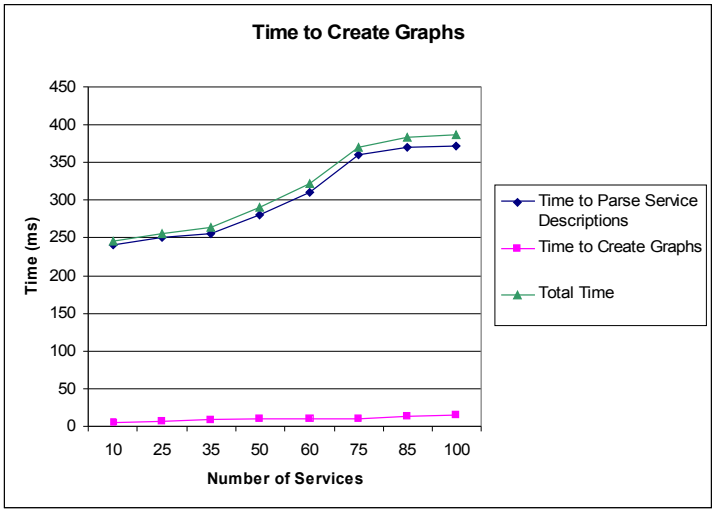


Figure 6.9: Time to Organize a PERSE Registry

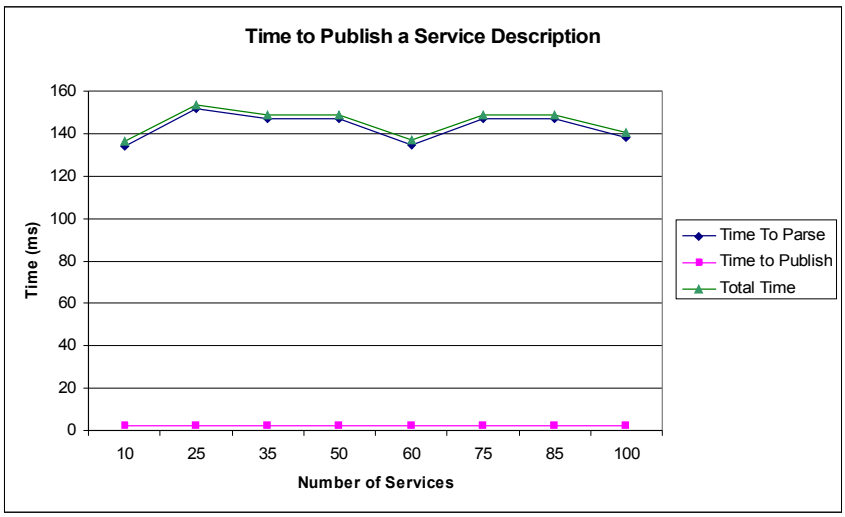


Figure 6.10: Publication Time in a PERSE Registry

capabilities contained in the graph in which the capability will be inserted. This is due to the fact that graphs are indexed using the ontologies that are being used in the capabilities' descriptions, which allows pre-selecting a subset of graphs that are likely to be appropriate

for the insertion of the new capability. Thus, only a few number of semantic matches are performed in order to insert a capability in a registry.

The results of the third experiment that we performed are depicted in Figure 6.11. In this experiment, we evaluate the time to match a service request with services hosted by a registry. Furthermore, we compare the time to match a request in an organized registry with the time to match a request in an unorganized registry. Results are given without the XML parsing time of the request description. In this figure, we notice that without registry organization, the average overhead for matching is around 50% of the time to match when the registry is organized. Moreover, we notice that the time to match a request in the classified registry is nearly constant, which is due to the graphs indexing and the registry organization. We also notice that the response time to match a required capability, excluding XML parsing time, is in the order of few milliseconds compared to the original ontology-based semantic matching (few seconds).

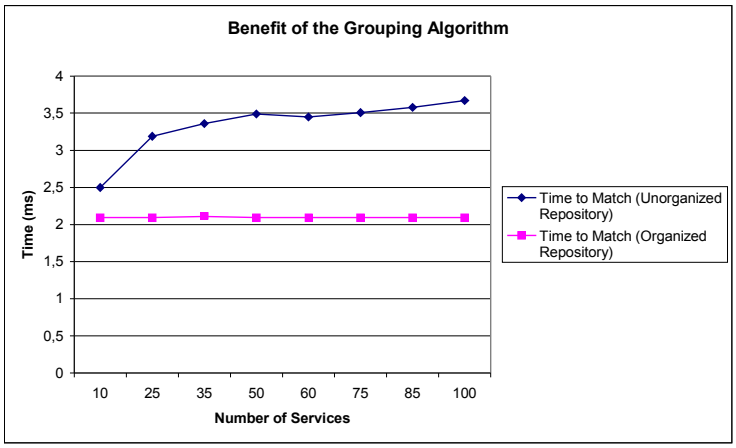


Figure 6.11: Benefit Of Grouping Service Advertisements

The last experiment that we performed is a comparison of the response time given by the classical syntactic-based matching performed by a WSDL registry and the optimized semantic matching performed by PERSE. The results are given in Figure 6.12. This figure shows that the response time given by the WSDL registry is increasing with the number of services available in the registry, while PERSE has an almost stable response time,

which is due to the following reasons: (1) using PERSE, the services are parsed once at the publishing phase and their capabilities are classified, which avoids matching a request with all the services of the registry; (2) due to the numeric encoding of ontologies, the semantic matching performed by PERSE reduces to a numeric comparison of codes, while in the case of the WSDL registry the matching is performed by syntactically comparing the WSDL descriptions. We conclude that, using PERSE, semantic matching, which allows to leverage the openness of pervasive computing environments, can be performed more efficiently than classical syntactic matching. Furthermore, thanks to registry indexing and structuring, PERSE is more scalable than existing unorganized syntactic based and semantic based registries.

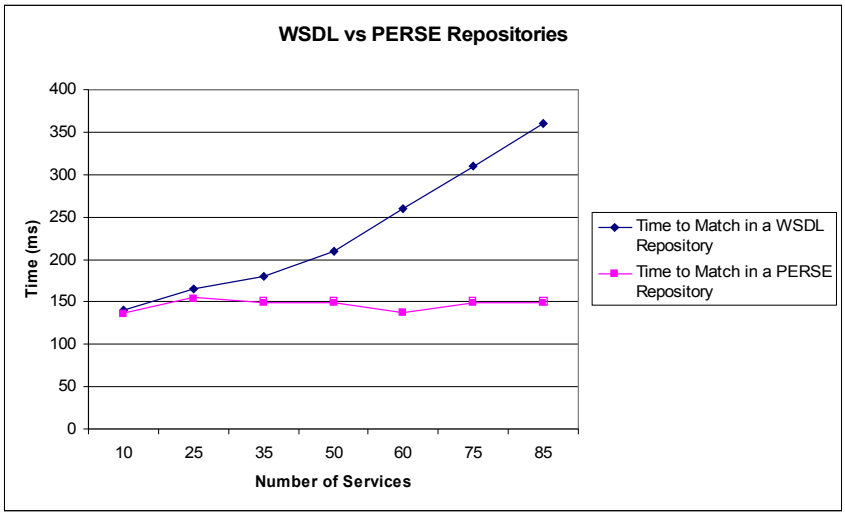


Figure 6.12: WSDL vs PERSE Service Registries

6.3.5 Performance of the QoS-Aware, Conversation-Based Service Composition

After the evaluation of the cost of interoperability and the performance of our PERSE service registry we present in this section the performance evaluation of the semantic service composition performed in PERSE.

We have implemented a prototype of our composition algorithm that supports the integration of service conversations (called PERSE-Composition in the following), presented in Chapter 5. All the experiments presented in this sections have been carried out on a Linux platform running on a laptop with an Intel Pentium 4, 2.80 GHz CPU and 512 MB of memory. The performance of PERSE-Composition is proportional to the complexity of the task and services' conversations. Specifically, the response time of the algorithm is proportional to the number of possible (intermediate) composition paths investigated during the execution of the algorithm. There are two main factors contributing to the increase of the intermediate composition paths: (1) the number of semantically equivalent capabilities provided by networked services; (2) the number of capabilities required in the task's conversation. We have carried out three experiments. The first two evaluate the impact of each factor on the performance of PERSE-Composition, while the third experiment evaluates the impact of QoS-awareness in PERSE-Composition. In all these experiments, each value is calculated from an average of ten runs.

Figure 6.13 considers the first factor. In this figure, the number of capabilities provided by networked services is increasing from 10 to 100 capabilities that are semantically equivalent. Two cases for the user task are considered: the case where the task is composed of a single capability, and the case where the task is composed of 5 semantically equivalent capabilities in sequence. We compare the performance of with the XML parsing of the services and task descriptions. The resulting curves show that the cost of our algorithm is lower than the XML parsing time. Furthermore, the time to find a service composition is proportional to the number of available services and to the task size.

Figure 6.14 considers the second factor. In this figure, the number of capabilities provided by the networked services is fixed to the worst case coming from the previous experiment, i.e., 100 semantically equivalent capabilities, while the number of capabilities required in the task's conversation is increasing from 1 to 20. The experiment that is depicted in this figure corresponds to the comparison of the performance of PERSE-Composition with the XML parsing of the services and the task conversation descriptions. The figure shows an extreme scenario for our algorithm, as each capability required in the task's conversation is matched against 100 capabilities, and the resulting number of pos-

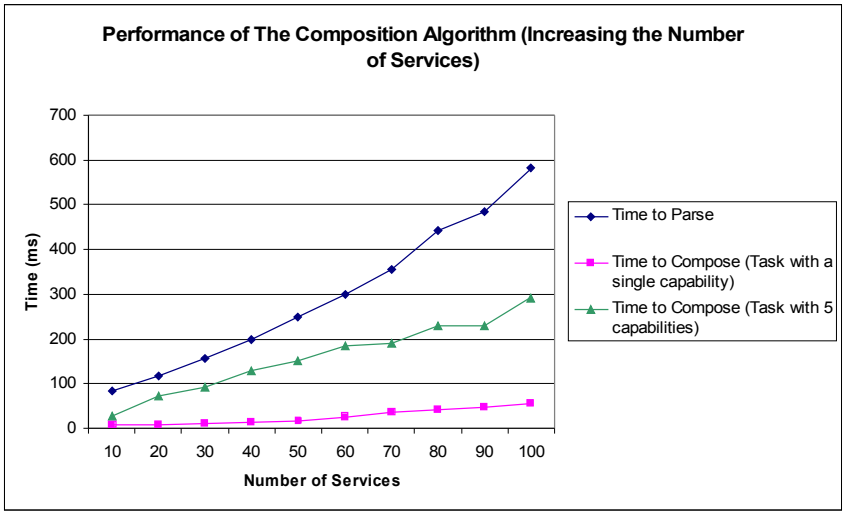


Figure 6.13: Performance of the Composition Algorithm (Increasing the Number of Services)

sible compositions is equal to: 100^{nb} in each case, where nb is the number of capabilities required in the task's conversation. We can see that for a number of possible compositions less than 100^{10}, our algorithm takes less time than the XML parsing time.

In realistic cases, both the user task and networked services will contain various capabilities organized using various workflow constructs, thus leading to the decrease of possible resulting compositions. Consequently, the response time will be reasonable for the pervasive computing environment. Indeed, we have applied our algorithm in a real case example in which the task's conversation contains twenty required capabilities and the selected services provide thirty capabilities, including various control constructs (e.g. sequence, choice, loop). In spite of the large number of capabilities required in the task's conversation, the algorithm spent only 32 milliseconds to find the two resulting compositions among 36 intermediate compositions, against 152 milliseconds for the XML parsing time.

The last experiment that we performed measures the impact of introducing QoS-awareness in PERSE-Composition. Specifically, we have compared the results of the pre-

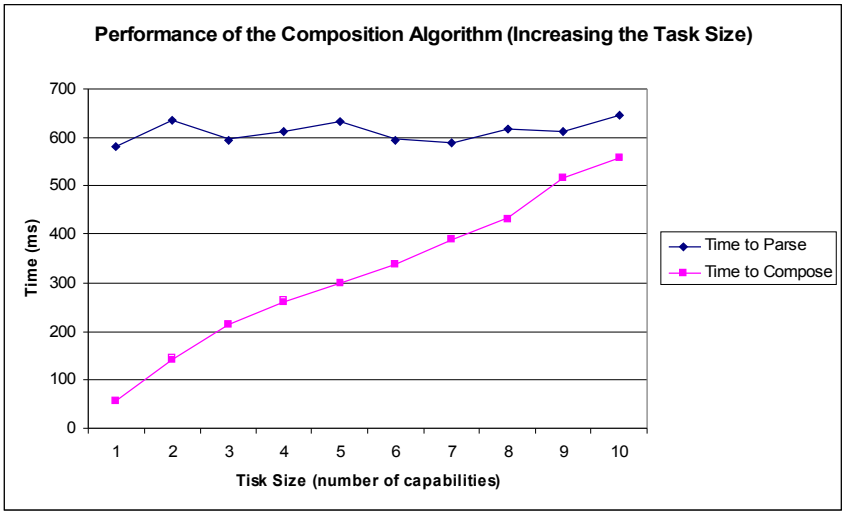


Figure 6.14: Performance of the Composition Algorithm (Increasing the Task Size)

vious experiment with the same experiment modified by introducing QoS constraints in the user task and QoS properties in each service description. The introduced QoS constraints and properties are related with with service latency and availability. Results are depicted in Figure 6.15. These results show that the impact of introducing QoS-awareness is amounts to a small increase in the XML parsing time, which is due to the addition of XML tags for describing QoS, while at the same time to a considerable decrease of the execution time of our algorithm. This is attributed to the rejection of a number of paths that do not fulfil the QoS requirements of the user task during the integration of service conversations.

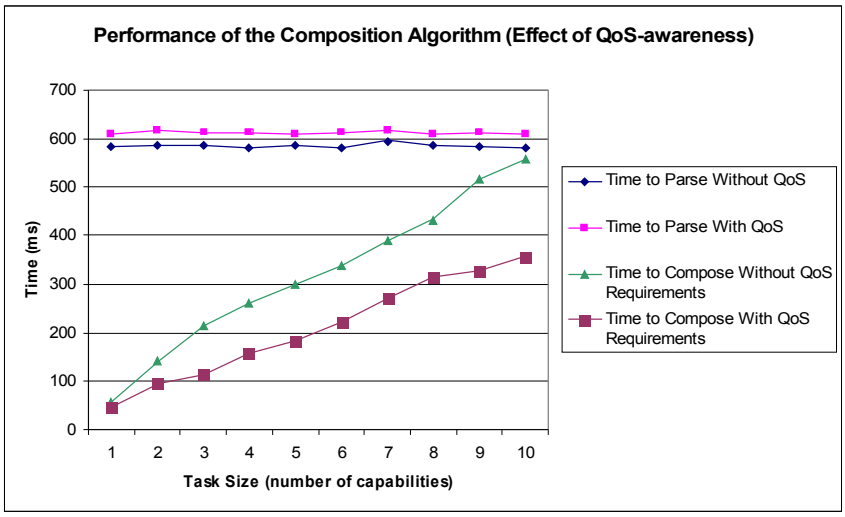


Figure 6.15: Performance of QoS-aware Service Composition

Chapter 7

Conclusion

The pervasive computing vision is increasingly enabled by the large success of wireless networks and devices. In pervasive environments, functionalities provided by heterogeneous software and hardware resources may be discovered and integrated transparently towards assisting users in the realization of their daily tasks. Building upon the service oriented architecture paradigm allows having a homogeneous view of the heterogeneous functionalities populating pervasive environments, as services have standard descriptions and are discovered and communicate using standard protocols. Having such homogeneous view enables the dynamic discovery access and composition of services towards the realization of user tasks. However, the emergence of a large number of candidate service description, discovery and access "standards", and their associated middleware platforms has generated middleware heterogeneity for which interoperability methods have to be developed to deal with the openness of pervasive computing environments. In addition to middleware heterogeneity, pervasive computing environments are characterized with network heterogeneity, which further restricts the ability to discover, access and compose services available in the vicinity.

Middleware heterogeneity, which essentially concerns service discovery and access protocols, has been the focus of intensive research in the last decade [Grace et al., 2003, Bromberg and Issarny, 2005, Raverdy et al., 2006]. However, existing solutions rely on the syntactic conformance of service interfaces supported by syntactic protocols. Such

assumption requires a common agreement on the syntax employed for describing service capabilities world-wide, which is hardy achievable in open pervasive environments. This raises the issue of syntactic heterogeneity of service interfaces.

Similarly to service discovery and access, most existing solutions to service composition in pervasive environments poorly deal with syntactic heterogeneity, since they assume that services being integrated have been pre-developed to conform syntactically in terms of interfaces [Sousa and Garlan, 2002, Shiva Chetan and Campbell, 2005, Issarny et al., 2005, Walker, 2004, Kumar et al., 2003]. Furthermore, existing service discovery and composition solutions provide limited support of service QoS properties, which is a key requirement towards the realization of the user-centric vision aimed at by the pervasive paradigm.

Building upon semantic Web technologies, and particularly ontologies, allows the unambiguous semantic specification of service functional and QoS properties in pervasive computing environments as investigated in [Masuoka et al., 2003, Singh et al., 2005, Chakraborty et al., 2006, Chakraborty et al., 2005]. However, such rich specifications require the use of costly semantic reasoning on the employed ontologies in order to asses the conformance of service capabilities against service requests, which is inappropriate regarding the resource constraints and the highly interactive feature of pervasive computing environments.

Hence, the realization of the pervasive computing vision calls for an efficient, semantic, QoS-aware middleware for service-oriented pervasive computing that supports multi-network and multi-protocol interoperability.

7.1 Contribution

To address the above challenges, we presented in this thesis a semantic, service-oriented middleware for pervasive computing and a prototype implementation of this middleware, i.e., the PERSE middleware.

This middleware provides a service model to support interoperability between heterogeneous both semantic and syntactic service description languages. It further enhances the specification of semantic services with the explicit specification of annotation types to

deal with a new source of semantic heterogeneity identified in this thesis, which is related with the sense associated with a semantic annotation. This enables service providers and requesters to provide more accurate semantic specifications, which allows our middleware to perform more accurate semantic service matching. Our model further supports the formal specification of service conversations as finite state automata, which enables the automated reasoning about service behaviour independently from the underlying conversation specification language. Hence, pervasive service conversations described with different service conversation languages can be integrated towards the realization of a user task. Finally, our model supports the specification of service non-functional properties based on existing QoS models to meet the specific requirements of each pervasive application.

As part of the PERSE prototype, we presented the instantiation of our model into the Interoperable Service Description Language, i.e., ISDL language. ISDL is an XML-based schema defining a container, which is combined with the two emergent standard service description languages namely SAWSDL for the specification of syntactic and semantic service capabilities and WS-BPEL for the specification of service conversations. Then, interoperability is achieved by translating heterogeneous service descriptions into ISDL. The performance evaluation of the computation cost of legacy to ISDL translation demonstrated that this cost is not significant compared to the overall discovery time.

As part of our middleware, we presented an efficient semantic service registry for pervasive computing environments. This registry supports a set of conformance relations for matching both syntactic and rich semantic service descriptions as well as their heterogeneous non-functional properties. These conformance relations also identify the semantic distance between service descriptions, and rate services with respect to their suitability for a specific service request, so that selection can be made among them.

In addition to the support of interoperable service matching and service ranking, our registry proves to be highly efficient thanks to an appropriate ontology encoding algorithm, which translates the costly semantic reasoning on ontologies to a numeric comparison of codes. Furthermore, service descriptions in our registry are semantically organized to enable both efficient both service publication and location.

The performance evaluation of our registry as part of the PERSE prototype, shows that

semantic service matching carried out by our registry performs as efficiently as syntactic-based service matching. Furthermore, thanks to the combination of ontology encoding and registry organization, PERSE service registry achieves efficient both service publication and location contrary to existing efficient semantic service registries that overload the service publication phase to achieve efficiency at service location.

To support mobile users of the pervasive environment in the realization of their daily tasks our middleware provides a service composition middleware functionality. Contrary to existing research efforts on service composition that assume complex behaviour for either services or tasks, our functionality enable the flexible QoS-aware composition of a set of services described with a complex behaviour to realize a user task also described with a complex behaviour. Flexibility is enabled by a set of composition algorithms that may be carried out according to the current resource constraints of the user's device. These algorithms further support the assessment of the QoS requirements of user tasks by aggregating the QoS provided by the composed networked services. Furthermore, this integration is performed efficiently as it relies on our efficient semantic service registry to discover services and on efficient formal verification algorithms to build the user task realizations. Specifically, the theoretical assessment of our various composition algorithms proves that their computation cost grows linearly with the complexity of the composed services and the target user task. Furthermore, the performance evaluation of our conversation integration algorithm performed as part of PERSE, show that in more realistic cases, the overhead of this algorithm is negligible compared to XML parsing of service descriptions. We have further done experiments for evaluating the impact of introducing QoS-awareness. Results show the introduction of QoS constraints improves the performance the conversation integration algorithm.

Finally, the PERSE prototype implementation of our middleware, which constitutes an innovative, efficient and comprehensive solution towards the realization of the pervasive computing vision, has been successfully integrated in the IST Amigo project, which envisions ambient intelligence in the networked home environments[1]. The PERSE pro-

[1]Amigo: http://www.extra.research.philips.com/euprojects/amigo/

totype has further been demonstrated in the Phillips research homelab[2] as well as in [Ben Mokhtar et al., 2007c].

7.2 Perspectives

Besides the contributions presented above, short term and long term perspectives are still to be investigated to enable the full potential of the pervasive computing vision. Short term perspectives represent enhancements of our proposed middleware functionalities while long term perspectives can be realized through the extension of our middleware with additional functionalities.

Among the short term perspectives that can be investigated is the extension of our semantic service model with the specification of service preconditions and effects along with efficient solutions for matching them. This would increase the richness of service descriptions and would lead to more accurate service matching. However, existing solutions for matching service preconditions and effects rely either on costly theorem proving algorithms [Zaremski and Wing, 1997] or on the querying of centralized knowledge bases [Sycara et al., 1999]. Both these solutions are not appropriate to be employed in the inherently distributed, resource constrained pervasive environment.

Another short term perspective that can be investigated is the support of ontology extension by service providers and requesters. Indeed, due to the offline encoding of ontology hierarchies, our current solution provides the mean of using (different versions of) existing ontologies for semantic service annotation, while ontology evolution is assumed to performed only by the ontology developers. Enabling service providers and requesters to extend existing ontologies by defining their customized concepts can be done by the support of more complex semantic annotations. Instead of referring to a single concept for annotating a service element, a service description will support semantic annotations expressed as logical expression (e.g., conjunction, disjunction between existing ontology concepts).

We mainly focused in this thesis on the realization of user-centric tasks by dealing

[2]http://www.research.philips.com/technologies/misc/homelab/

with various forms of heterogeneity and device resource constraints characterizing pervasive computing environments. Long term perspectives include the extension of our middleware with mobility-awareness. For instance, service location can be enhanced with mobility-awareness by performing signal strength analysis [Liu, 2006] or by considering the service providers and requesters mobility patterns [Mcnamara et al., 2006]. Furthermore, solutions to the dynamic service reconfiguration can be investigated to deal with the appearance of (better) services and the disappearance of services taking part of a user task being executed [Zarras et al., 2006]. Additionally, the extension of our middleware with the support of (distributed) context management would give our middleware the aptitude to be aware of user characteristics, system behaviour and state of the physical environment [Fournier et al., 2006]. This can improve the enforcement of service non-functional properties assumed so far to be provided by services.

Appendix A

Proofs

A.1 Proof of the property [Prop 1]

Prop 1 : ¬ **FunctionalCapabilityMatch**(***Capability****(Root$_i$), Adv***): Root$_i$ ∈ **Roots**(G) ⇒
∀ N ∈ **SubGraph**(*Root$_i$*): ¬ **FunctionalCapabilityMatch**(***Capability***(N), *Adv*)

We prove [Prop 1] by contradiction. Assume ¬ [Prop 1], i.e.:

¬ **FunctionalCapabilityMatch**(***Capability****(Root$_i$), Adv***): Root$_i$ ∈ **Roots**(G) and (1)

¬ (∀ C ∈ **SubGraph**(*Root$_i$*): ¬ **FunctionalCapabilityMatch**(***Capability****(N), Adv***)) (2)

(2) ⇔ ∃ C ∈ **SubGraph**(*Root$_i$*): **FunctionalCapabilityMatch**(***Capability****(N), Adv*)
On the other hand: C ∈ **SubGraph**(*Root$_i$*) ⇒ **FunctionalCapabilityMatch**(***Capability****(Root$_i$),* ***Capability****(N)*) from the definition of the function **SubGraph**(); thus:

(2) ⇔ ∃ C ∈ **SubGraph**(*Root$_i$*): **FunctionalCapabilityMatch**(***Capability****(Root$_i$),* ***Capability****(N)*) and **FunctionalCapabilityMatch**(***Capability****(N), Adv*)

From the transitivity property of the function **FunctionalCapabilityMatch**(), we have: (2) ⇔ **FunctionalCapabilityMatch**(***Capability****(Root$_i$), Adv*)

Replacing (2) in the list of our assumptions with this equivalence results into:
¬ **FunctionalCapabilityMatch**(***Capability***$(Root_i)$, Adv) and **FunctionalCapabilityMatch**(***Capability***$(Root_i)$, Adv). This can never be true, and therefore, the assumption is false and [Prop 1] is true.

A.2 Proof of the property [Prop 2]

Prop 2 : ¬ **FunctionalCapabilityMatch**(Adv, ***Capability***$(Leaf_i)$): $\text{Leaf}_i \in$ **Leaves**(G) ⇒
∀ N ∈ **ParentGraph**($Leaf_i$): ¬ **FunctionalCapabilityMatch**(Adv, ***Capability***(N))

We prove [Prop 2] by contradiction. Assume ¬ [Prop 2], i.e.:

¬ **FunctionalCapabilityMatch**(Adv, ***Capability***$(Leaf_i)$): $\text{Leaf}_i \in$ **Leaves**(G) and (1)
¬ (∀ C ∈ **ParentGraph**($Leaf_i$): ¬ **FunctionalCapabilityMatch**(Adv, ***Capability***(N))) (2)

(2) ⇔ ∃ C ∈ **ParentGraph**($Leaf_i$): **FunctionalCapabilityMatch**(Adv, ***Capability***(N))
On the other hand: C ∈ **ParentGraph**($Leaf_i$) ⇔ **FunctionalCapabilityMatch**(***Capability***(N), ***Capability***$(Leaf_i)$) from the definition of the function **ParentGraph**(); thus:

(2) ⇔ ∃ C ∈ **ParentGraph**($Leaf_i$): **FunctionalCapabilityMatch**(Adv, ***Capability***(N)) and **FunctionalCapabilityMatch**(***Capability***(N), ***Capability***$(Leaf_i)$)

From the transitivity property of the function **FunctionalCapabilityMatch**(), we have: (2) ⇔ **FunctionalCapabilityMatch**(Adv, ***Capability***$(Leaf_i)$)

Replacing (2) in the list of our assumptions with this equivalence results into:
¬ **FunctionalCapabilityMatch**(Adv, ***Capability***$(Leaf_i)$) and **FunctionalCapabilityMatch**(Adv, ***Capability***$(Leaf_i)$). This can never be true, and therefore, the assumption is false and [Prop 2] is true.

A.3 Proof of the property [Prop 3]

[Prop 3)]: $\forall n_1, n_2 \in \mathcal{N}^2$, **ConceptMatch**($n_1$,$n_2$):
[$\exists n'_1, n'_2 \in \mathcal{N}^2$: **ConceptMatch**($n'_1$,$n_1$) $\wedge$ **ConceptMatch**(n_2,n'_2) $\Rightarrow$ **ConceptMatch**(n'_1,n'_2)]

The prove of this property is trivial and relies on the transitivity property of the relation **ConceptMatch**() proved below.

A.4 Proof of the transitivity of the relation FunctionalCapabilityMatch()

Lets C_1, C_2 and C_3 be three capabilities. Lets prove that:

FunctionalCapabilityMatch(C_1,C_2) $\wedge$ **FunctionalCapabilityMatch**(C_2,C_3) $\Rightarrow$ **FunctionalCapabilityMatch**(C_1,C_3)

Lets assume:

FunctionalCapabilityMatch(C_1,C_2) $\wedge$ **FunctionalCapabilityMatch**(C_2,C_3)

FunctionalCapabilityMatch(C_1,C_2) $\Leftrightarrow$
$\forall\ in_2 \in I_2, \exists\ in_1 \in I_1$: **ConceptMatch**($in_1$, in_2) and (1)
$\forall\ out_2 \in O_2, \exists\ out_1 \in O_1$: **ConceptMatch**($out_1$, out_2) and (2)
ConceptMatch(cat_1, cat_2) (3)

FunctionalCapabilityMatch(C_2,C_3) $\Leftrightarrow$
$\forall\ in_3 \in I_3, \exists\ in_2 \in I_2$: **ConceptMatch**($in_2$, in_3) and (4)
$\forall\ out_3 \in O_3, \exists\ out_2 \in O_2$: **ConceptMatch**($out_2$, out_3) and (5)
ConceptMatch(cat_2, cat_3) (6)

From the transitivity property of the relation **ConceptMatch**() we have:

(4) $\wedge$ (1) $\Rightarrow \forall\ in_3 \in I_3, \exists\ in_1 \in I_1$: **ConceptMatch**($in_1$, in_3) (7)

(5) $\wedge$ (2) $\Rightarrow \forall\ out_3 \in O_3, \exists\ out_1 \in O_1$: **ConceptMatch**($out_1$, out_3) (8)

(6) $\wedge$ (3) $\Rightarrow$ **ConceptMatch**(cat_1, cat_3) (9)

According to the definition of the **FunctionalCapabilityMatch**() relation we can infer that:
(7) $\wedge$ (8) $\wedge$ (9) $\Rightarrow$ **FunctionalCapabilityMatch**(C_1,C_3).

A.5 Proof of transitivity of the relation ConceptMatch()

Lets n_1, n_2 and n_3 be three concepts in an ontology. Lets prove that:

ConceptMatch(n_1,n_2) $\wedge$ **ConceptMatch**(n_2,n_3) $\Rightarrow$ **ConceptMatch**(n_1,n_3)

Lets assume:

ConceptMatch(n_1,n_2) $\wedge$ **ConceptMatch**(n_2,n_3)
According to the annotation semantics of n_1, n_2 and n_3 and the definition of the **ConceptMatch**() relation, there are four possible cases:

Case 1: $n_1^{\otimes}$, $n_2^{\otimes}$, $n_3^{\otimes}$
In this case we have:

ConceptMatch($n_1^{\otimes}$,$n_2^{\otimes}$) $\Rightarrow$ **Subsume**(n_1,n_2)(1)

ConceptMatch($n_2^{\otimes}$,$n_3^{\otimes}$) $\Rightarrow$ **Subsume**(n_2,n_3)(2)

Using the transitivity property of the relation **Subsume**() we have:

(1) and (2) $\Rightarrow$ **Subsume**(n_1,n_3)

From the definition of the relation **ConceptMatch**() we have:

Subsume(n_1,n_3) $\Leftrightarrow$ **ConceptMatch**($n_1^{\otimes}$,$n_3^{\otimes}$)

Case 2: $n_1^{\otimes}$, $n_2^{\oplus}$, $n_3^{\oplus}$

In this case we have:

$$\textbf{ConceptMatch}(n_1^{\otimes},n_2^{\oplus}) \Rightarrow n_1^{\otimes} = c_1 \wedge c_2 \wedge c_3 \wedge ... \wedge$$
$$n_2^{\oplus} = c'_1 \vee c'_2 \vee c'_3 \vee ... \wedge$$
$$\exists c_i, \exists c'_j : c_i = c'_j \ (3)$$

$$\textbf{ConceptMatch}(n_2^{\oplus},n_3^{\oplus}) \Rightarrow n_2^{\oplus} = c'_1 \vee c'_2 \vee c'_3 \vee ... \wedge$$
$$n_3^{\oplus} = c''_1 \vee c''_2 \vee c''_3 \vee ... \wedge$$
$$\forall c'_j, \exists c''_k : c'_j = c''_k \ (4)$$

(3) and (4) $\Rightarrow \exists c_i, \exists c''_k : c_i = c''_k$

From the definition of the relation **ConceptMatch**() we have:

ConceptMatch($n_1^{\otimes}$,$n_3^{\oplus}$)

Case 3: $n_1^{\oplus}$, $n_2^{\oplus}$, $n_3^{\oplus}$

In this case we have:

ConceptMatch($n_1^{\oplus}$,$n_2^{\oplus}$) $\Rightarrow$ **Subsume**(n_2,n_1)(5)

ConceptMatch($n_2^{\oplus}$,$n_3^{\oplus}$) $\Rightarrow$ **Subsume**(n_3,n_2)(6)

Using the transitivity property of the relation **Subsume**() we have:

(5) and (6) $\Rightarrow$ **Subsume**(n_3,n_1)

From the definition of the relation **ConceptMatch**() we have:

Subsume(n_3,n_1) $\Leftrightarrow$ **ConceptMatch**($n_1^{\oplus}$,$n_3^{\oplus}$)

Case 4: $n_1^{\otimes}$, $n_2^{\otimes}$, $n_3^{\oplus}$

In this case we have:

$$\begin{aligned}\textbf{ConceptMatch}(n_1^{\otimes},n_2^{\otimes}) \Rightarrow\ & n_1^{\otimes} = c_1 \wedge c_2 \wedge c_3 \wedge ... \wedge \\ & n_2^{\times} = c'_1 \wedge c'_2 \wedge c'_3 \wedge ... \wedge \\ & \forall c'_j, \exists c_i : c_i = c'_j \ (7)\end{aligned}$$

$$\begin{aligned}\textbf{ConceptMatch}(n_2^{\times},n_3^{\oplus}) \Rightarrow\ & n_2^{\times} = c'_1 \wedge c'_2 \wedge c'_3 \wedge ... \wedge \\ & n_3^{\oplus} = c''_1 \vee c''_2 \vee c''_3 \vee ... \wedge \\ & \exists c'_j, \exists c''_k : c'_j = c''_k \ (8)\end{aligned}$$

(7) and (8) $\Rightarrow \exists c_i, \exists c''_k : c_i = c''_k$

From the definition of the relation **ConceptMatch**() we have:

ConceptMatch($n_1^{\otimes}$,$n_3^{\oplus}$)

In all the four cases, we have:

ConceptMatch(n_1,n_2) $\wedge$ **ConceptMatch**(n_2,n_3) $\Rightarrow$ **ConceptMatch**(n_1,n_3)

A.6 Complexity of the ConversationMatch() relation

In the classic formalization of generative grammars a grammar G=$< N, \Sigma, P, S >$ consists of the following components:

- N is a finite set of non-terminal symbols.
- Σ is a finite set of terminal symbols that is disjoint from N.
- P is a finite set of production rules.
- S is a starting non-terminal from N.

Right regular grammar is a formal grammar $G =< N, \Sigma, P, S >$ such that all the production rules in P are of one of the following forms:

1. A $\rightarrow$ a - where A is a non-terminal in N and a is a terminal in Σ
2. A $\rightarrow$ aB - where A and B are in N and a is in Σ
3. A $\rightarrow \epsilon$ - where A is in N and ϵ denotes the empty string, i.e. the string of length 0.

Lets prove that all the automata (of the user task and pervasive services) generated using our rules for mapping basic control patterns to finite state automata defined in Chapter 3 and represented in Figure 3.8 (p. 47) can be represented with right regular grammars. There are six different cases:

- c is an elementary capability $G :< N, \Sigma, P, S >$
  - N=A
  - $\Sigma = c$
  - P= $\{A \rightarrow c\}$
  - S=A

  G is by definition right regular.

- c=**Sequence**$(c_1, c_2, ..., c_n)$ where $G_{c_i} =< N^i, \Sigma^i, P^i, S^i >$ is assumed to be right regular. Let's prove that the grammar representing c is right regular:
  The grammar representing the capability c is given by : $G =< N, \Sigma, P, S >$ where:
  - $N = \bigcup N^i$
  - $\Sigma = \bigcup \Sigma^i$
  - P contains the following production rules:
    All the production rules from P^i in the following form are kept in P:
    $\{A^i_j \rightarrow aA^i_k\}$

    All the production rules from P^i $(i \neq n)$ in the following form:
    $\{A^i_j \rightarrow a\}$
    are replaced in P by productions in the following form:
    $\{A^i_j \rightarrow aS^{i+1}\}$, which are right regular by substitution as all the production rules starting with S^{i+1} are right regular

    All the production rules from P^i $(i \neq n)$ in the following form:
    $\{A^i_j \rightarrow \epsilon\}$
    are replaced in P by productions in the following form:
    $\{A^i_j \rightarrow S^{i+1}\}$, which are right regular by substitution as all the production rules starting with S^{i+1} are right regular

  - $S = S^1$

  As all the production rules of G are either coming from the grammars G^i that are right regular or are right regular by definition, then G is also right regular.

- c=**ExcusiveChoice**$(c_1, c_2, ..., c_n)$ where $G_{c_i} =< N^i, \Sigma^i, P^i, S^i >$ is assumed to be right regular. Let's prove that the grammar representing c is right regular:
  The grammar representing the capability c is given by : $G =< N, \Sigma, P, S >$ where:
  - $N = \bigcup N^i \cup S_{init}$, $(i = 1..n)$

- $\Sigma = \bigcup \Sigma^i$, $(i = 1..n)$
- $P = \bigcup P^i \cup \{S_{init} \rightarrow S^i\}$, $(i = 1..n)$
- $S = S_{init}$

As all the production rules of G are either coming from the grammars G^i that are right regular or are right regular by definition, then G is also right regular.

- c=**SimpleMerge**$(c_1, c_2, ..., c_n, c')$ where $G_{c_i} =< N^i, \Sigma^i, P^i, S^i >$ is assumed to be right regular. The capability c is treated as
  c=**Sequence**(***ExclusiveChoice***$(c_1, c_2, ..., c_n)$,c')
  By substitution from the definition of the grammars representing the **Sequence**() and **ExclusiveChoice**() control patterns, the grammar representing the **SimpleMerge**() control pattern is also right regular.

- c=**ParallelSplit**(c_1, c_2), where $G_{c_i} =< N^i, \Sigma^i, P^i, S^i >$ is assumed to be right regular. The capability c is treated as
  c=**ExclusiveChoice**(***Sequence***(c_1, c_2),***Sequence***(c_2, c_1))
  By substitution from the definition of the grammars representing the **Sequence**() and **ExclusiveChoice**() control patterns, the grammar representing the **ParallelSplit**() control pattern is also right regular.

- c=**Synchronisation**$(c_1, ..., c_n)$, where $G_{c_i} =< N^i, \Sigma^i, P^i, S^i >$ is assumed to be right regular. The capability c is treated as
  c=**SimpleMerge**(***ParallelSplit***$(c_1, ..., c_n)$)
  By substitution from the definition of the grammars representing the **SimpleMerge**() and **ParallelSplit**() control patterns, the grammar representing the **Synchronisation**() control pattern is also right regular.

In all the above six cases there exist a grammar G representing the basic control patterns that we use for generating the automata for user tasks in pervasive services, which is right regular.

Bibliography

[Aggarwal et al., 2004] Aggarwal, R., Verma, K., Miller, J., and Milnor, W. (2004). Dynamic web service composition in meteor-s. Technical report, LSDIS Lab, Computer Science Dept., UGA.

[Agrawal et al., 1989] Agrawal, R., Borgida, A., and Jagadish, H. V. (1989). Efficient management of transitive relationships in large data and knowledge bases. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data(SIGMOD '89)*, pages 253–262.

[Ait-Kaci et al., 1989] Ait-Kaci, H., Boyer, R. S., Lincoln, P., and Nasr, R. (1989). Efficient implementation of lattice operations. *Programming Languages and Systems*, 11(1):115–146.

[Ben Mokhtar et al., 2005a] Ben Mokhtar, S., Fournier, D., Georgantas, N., and Issarny, V. (2005a). Context-aware service composition in pervasive computing environments. In *Proceedings of the 2nd International Workshop on Rapid Integration of Software Engineering Techniques (RISE'05)*.

[Ben Mokhtar et al., 2005b] Ben Mokhtar, S., Georgantas, N., and Issarny, V. (2005b). Ad hoc composition of user tasks in pervasive computing environments. In *Proceedings of the 4th Workshop on Software Composition (SC'05)*.

[Ben Mokhtar et al., 2006a] Ben Mokhtar, S., Georgantas, N., and Issarny, V. (2006a). Cocoa: Conversation-based service composition in pervasive computing environments. In *Proceedings of the IEEE International Conference on Pervasive Services (ICPS'06)*.

[Ben Mokhtar et al., 2007a] Ben Mokhtar, S., Georgantas, N., and Issarny, V. (2007a). Cocoa: Conversation-based service composition in pervasive computing environments with qos support. *Journal of System and Software*.

[Ben Mokhtar et al., 2006b] Ben Mokhtar, S., Kaul, A., Georgantas, N., and Issarny, V. (2006b). Efficient semantic service discovery in pervasive computing environments. In *Proceedings of ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06)*.

[Ben Mokhtar et al., 2006c] Ben Mokhtar, S., Kaul, A., Georgantas, N., and Issarny, V. (2006c). Towards efficient matching of semantic web service capabilities. In *Proceedings of the workshop of Web Services Modeling and Testing (WS-MATE'06)*.

[Ben Mokhtar et al., 2005c] Ben Mokhtar, S., Liu, J., Georgantas, N., and Issarny, V. (2005c). Qos-aware dynamic service composition in ambient intelligence environments. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*.

[Ben Mokhtar et al., 2007b] Ben Mokhtar, S., Preuveneers, D., Georgantas, N., Issarny, V., and Berbers, Y. (2007b). Easy: Efficient semantic service discovery in pervasive computing environments with qos and context support. *Journal of System and Software*.

[Ben Mokhtar et al., 2007c] Ben Mokhtar, S., Raverdy, P.-G., Cardoso, R. S., Urbieta, A., and Georgantas, N. (2007c). Discovering social services in pervasive environments with privacy. In *Proceedings of ACM/IFIP/USENIX 8th International Middleware Conference (Middleware'07), Demonstration paper*.

[Benatallah et al., 2003] Benatallah, B., M.Dumas, and Sheng, Q. Z. (2003). The self-serv environment for web services composition. In *IEEE Internet Computing*.

[Berardi et al., 2003] Berardi, D., Calvanese, D., Giacomo, G. D., Hull, R., and Mecella, M. (2003). Automatic composition of web services in colombo. In *Proceedings of the 13th Italian Symposium on Advanced Database Systems (SEBD'05)*.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*.

[Bernstein and Klein, 2002] Bernstein, A. and Klein, M. (2002). Towards high-precision service retrieval. In *Proceedings of The First International Semantic Web Conference(ISWC'02)*.

[Brogi and Popescu, 2005] Brogi, A. and Popescu, R. (2005). Towards semi-automated workflow-based aggregation of web services. In *Proceedings of Third International Conference on Service Oriented Computing (ICSOC'05)*.

[Bromberg, 2006] Bromberg, Y.-D. (2006). *Résolution de l'hétérogénéité des intergiciels d'un environnement ubiquitaire*. PhD thesis, L'université de Versailles Saint-Quentin-en-Yvelines.

[Bromberg and Issarny, 2005] Bromberg, Y.-D. and Issarny, V. (2005). Indiss: Interoperable discovery system for networked services. In *Proceedings of ACM/IFIP/USENIX 5th International Middleware Conference (Middleware'05)*, pages 164–183.

[Cámara et al., 2006] Cámara, J., Canal, C., Cubo, J., and Vallecillo, A. (2006). Formalizing wsbpel business processes using process algebra. *Electr. Notes Theor. Comput. Sci.*, 154(1):159–173.

[Cardoso et al., 2004] Cardoso, J., Sheth, A., Miller, J., Arnold, J., and Kochut, K. (2004). Quality of service for workflows and Web service processes. *Journal of Web Semantic*.

[Caseau, 1993] Caseau, Y. (1993). Efficient handling of multiple inheritance hierarchies. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications (OOPSLA'93)*, pages 271–287.

[Chakraborty et al., 2005] Chakraborty, D., Joshi, A., Finin, T., and Yesha, Y. (2005). Service Composition for Mobile Environments. *Journal on Mobile Networking and Applications, Special Issue on Mobile Services*, 10(4):435–451.

[Chakraborty et al., 2006] Chakraborty, D., Joshi, A., Yesha, Y., and Finin, T. (2006). Toward Distributed Service Discovery in Pervasive Computing Environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112. Available at http://doi.ieeecomputersociety.org/10.1109/TMC.2006.26.

[Constantinescu and Faltings, 2003] Constantinescu, I. and Faltings, B. (2003). Efficient matchmaking and directory services. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI'03)*.

[Dan et al., 2003] Dan, W., Bijan, P., Evren, S., James, H., and Dana, N. (2003). Automating DAML-S web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC'03)*.

[Donini et al., 1996] Donini, F. M., lenzerini, M., Nardi, D., and Schaerf, A. (1996). Reasoning in description logics. pages 191–236.

[Ducatel et al., 2001] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., and Burgelman, J.-C. (2001). Scenarios for ambient intelligence in 2010. Technical report, ISTAG.

[Filho and van Sinderen, 2003] Filho, J. G. P. and van Sinderen, M. (2003). Web service architectures - semantics and context-awareness issues in web services platforms. Technical report, Telematica Instituut.

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

[Fournier et al., 2006] Fournier, D., Ben Mokhtar, S., Georgantas, N., and Issarny, V. (2006). Towards ad hoc contextual services for pervasive computing. In *Proceedings of the MW4SOC Workshop of the 7th International Middleware Conference*.

[Grace et al., 2003] Grace, P., Blair, G. S., and Samuel, S. (2003). Remmoc: A reflective middleware to support mobile client interoperability. In *CoopIS/DOA/ODBASE*, pages 1170–1187.

[Hellerstein et al., 1995] Hellerstein, J. M., Naughton, J. F., and Pfeffer, A. (1995). Generalized search trees for database systems. In *Proceedings of the 21st International Conference of Very Large Data Bases, VLDB'95*.

[Issarny et al., 2007] Issarny, V., Caporuscio, M., and Georgantas, N. (2007). A perspective on the future of middleware-based software engineering. In Briand, L. and Wolf, A., editors, *Future of Software Engineering 2007*. IEEE-CS Press.

[Issarny et al., 2005] 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*.

[Kanellakis and Smolka, 1990] Kanellakis, P. C. and Smolka, S. A. (1990). Ccs expressions finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68.

[Krall et al., 1997] Krall, A., Vitek, J., and Horspool, N. (1997). Near optimal hierarchical encoding of types. In *11th European Conference on Object Oriented Programming (ECOOP'97)*, pages 128–145. Springer.

[Kumar et al., 2003] Kumar, M., Shirazi, B. A., Das, S. K., Sung, B. Y., Levine, D., and Singhal, M. (2003). Pico: A middleware framework for pervasive computing. *IEEE Pervasive Computing*.

[Liu, 2006] Liu, J. (2006). *Supporting QoS-aware Service Discovery in Ubiquitous Computing Environments*. PhD thesis, University of Versailles-Saint Quentin en Yvelines.

[M. P. Papazoglou, 2003] M. P. Papazoglou, D. G. (2003). *Special section in Communications of the ACM*, chapter Service-oriented computing. ACM Press.

[Majithia et al., 2004] Majithia, S., Walker, D. W., and Gray, W. A. (2004). A framework for automated service composition in service-oriented architecture. In *1st European Semantic Web Symposium*.

[Masuoka et al., 2003] Masuoka, R., Parsia, B., and Labrou, Y. (2003). Task computing - the semantic web meets pervasive computing. In *2nd International Semantic Web Conference (ISWC2003)*.

[M'Bareck and Tata, 2007] M'Bareck, N. O. A. and Tata, S. (2007). How to consider requester's preferences to enhance web service discovery? In *Second International Conference on Internet and Web Applications and Services (ICIW '07)*.

[Mcnamara et al., 2006] Mcnamara, L., Mascolo, C., , and Capra, L. (2006). Trust and mobility aware service provision for pervasive computing. In *Workshop on Requirements and Solutions for Pervasive Software Infrastructures*.

[Menasce, 2004] Menasce, D. (2004). Composing Web services: A QoS view. *IEEE Internet Computing*, Vol. 8.(No. 6).

[Moller and Smolka, 2003] Moller, F. and Smolka, S. A. (2003). On the computational complexity of bisimulation, redux. In *Proceedings of the Paris C. Kanellakis memorial workshop on Principles of computing & knowledge (PCK50)*.

[Narayanan and McIlraith, 2002] Narayanan, S. and McIlraith, S. A. (2002). Simulation, verification and automated composition of Web services. In *Proceedings of the eleventh international conference on World Wide Web*.

[Niles and Pease, 2001] Niles, I. and Pease, A. (2001). Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems(FOIS'01)*, pages 2–9.

[Paige and Tarjan, 1987] Paige, R. and Tarjan, R. E. (1987). Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989.

[Paolucci et al., 2002] Paolucci, M., Kawamura, T., R.Payne, T., and Sycara, K. (2002). Semantic matching of Web services capabilities. *Lecture Notes in Computer Science*, 2342:333–347.

[Preuveneers and Berbers, 2006] Preuveneers, D. and Berbers, Y. (2006). Prime numbers considered useful: Ontology encoding for efficient subsumption testing. Technical Report CW464, Department of Computer Science, Katholieke Universiteit Leuven, Belgium.

[Ramasamy, 2006] Ramasamy, V. (2006). Syntactical and semantical web services discovery and composition. In *Proceedings of the cec-eee'06 conference.*

[Raverdy et al., 2006] Raverdy, P.-G., Issarny, V., Chibout, R., and de La Chapelle, A. (2006). A multi-protocol approach to service discovery and access in pervasive environments. In *Proceedings of MOBIQUITOUS - The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services.*

[Roman et al., 2002] Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K. (2002). Gaia: a middleware platform for active spaces. *SIGMOBILE Mobile Computing and Communication Review*, 6(4).

[Satyanarayanan, 2001] Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. *IEEE Personal Communications.*

[Shiva Chetan and Campbell, 2005] Shiva Chetan, A. R. and Campbell, R. H. (2005). Towards fault tolerant pervasive computing. *IEEE Technology and Society*, 24(1).

[Singh and Huhns, 2005] Singh, M. P. and Huhns, M. N. (2005). *Service-Oriented Computing Semantics, Processes, Agents.* John Wiley and Sons.

[Singh et al., 2005] Singh, S., Puradkar, S., and Lee, Y. (2005). Ubiquitous computing: Connecting pervasive computing through semantic web. *Information Systems and e-Business Management Journal.*

[Sirin et al., 2005] Sirin, E., Parsia, B., and Hendler, J. (2005). Template-based composition of semantic web services. In *AAAI Fall Symposium on Agents and the Semantic Web.*

[Sousa and Garlan, 2002] Sousa, J. P. and Garlan, D. (2002). Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of*

the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture.

[Srinivasan et al., 2004] Srinivasan, N., Paolucci, M., and Sycara, K. (2004). Adding owl-s to uddi, implementation and throughput. In *Proceedings of the Workshop on Semantic Web Service and Web Process Composition.*

[Sycara et al., 1999] Sycara, K., Lu, J., Klusch, M., and Widoff, S. (1999). Matchmaking among heterogeneous agents on the internet. In *Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace.*

[Sycara et al., 2003] Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N. (2003). Automated discovery, interaction and composition of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web.*

[Trastour et al., 2001] Trastour, D., Bartolini, C., and Gonzalez-Castillo, J. (2001). A semantic web approach to service description for matchmaking of services. In *Proceedings of the first Semantic Web Working Symposium, (SWWS).*

[van Bommel and Beck, 1999] van Bommel, M. F. and Beck, T. J. (1999). Incremental encoding of multiple inheritance hierarchies. In *Proceedings of the eighth international conference on Information and knowledge management(CIKM '99)*, pages 507–513.

[van der Aalst et al., 2000] van der Aalst, W. M. P., Barros, A. P., ter Hofstede, A. H. M., and Kiepuszewski, B. (2000). Advanced workflow patterns. In *Conference on Cooperative Information Systems*, pages 18–29.

[Walker, 2004] Walker, L. (2004). Mit wants to make computing as easy as breathing. *The Washington Post.*

[Wang et al., 2007] Wang, H. H., Gibbins, N., Payne, T., Saleh, A., and Sun, J. (2007). A formal semantic model of the semantic web service ontology (wsmo). In *Twelfth IEEE International Conference on Engineering of Complex Computer Systems.*

[Weiser, 1991] Weiser, M. (1991). The computer of the 21st century. *Scientific American*, 3(265):66–75.

[Wombacher et al., 2004] Wombacher, A., Fankhauser, P., and Neuhold, E. (2004). Transforming bpel into annotated deterministic finite state automata for service discovery. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, page 316, Washington, DC, USA. IEEE Computer Society.

[Zahariadis and Doshi, 2004] Zahariadis, T. and Doshi, B. (2004). Applications and services for the B3G/4G era. *IEEE Wireless Communications*.

[Zaremski and Wing, 1995] Zaremski, A. M. and Wing, J. M. (1995). Signature matching: a tool for using software libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146–170.

[Zaremski and Wing, 1997] Zaremski, A. M. and Wing, J. M. (1997). Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*.

[Zarras et al., 2006] Zarras, A., Fredj, M., Georgantas, N., and Issarny, V. (2006). *Rigorous engineering of fault tolerant systems*, chapter Engineering Reconfigurable Distributed Software Systems: Issues Arising for Pervasive Computing. LNCS.

[Zeng et al., 2004] Zeng, L., Benatallah, B., Ngu, A. H. H., Dumas, M., Kalagnanam, J., and Chang, H. (2004). Qos-aware middleware for web services composition. *IEEE Transactions on Software Engeneering*.

[Zibin and Gil, 2001] Zibin, Y. and Gil, J. (2001). Efficient subtyping tests with PQ-encoding. In *Conference on Object-Oriented*, pages 96–107.