A GOAL-BASED FRAMEWORK FOR SEMANTIC SERVICE PROVISIONING

Luiz Olavo Bonino da Silva Santos

A Goal-Based Framework for Semantic Service Provisioning

Luiz Olavo Bonino da Silva Santos

Enschede, The Netherlands, 2011

CTIT Ph.D.-Thesis Series, No. 11-195

Cover Design:	Ana Luiza Moori, Amsterdam
Book Design:	Lidwien van de Wijngaert and Henri ter Hofte
Printing:	Ipskamp, Enschede, the Netherlands

CTIT Ph.D.-Thesis Series, No. 11-195
ISSN 1381-3617
ISBN 978-90-365-3174-0
Centre for Telematics and Information Technology, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

A GOAL-BASED FRAMEWORK FOR SEMANTIC SERVICE PROVISIONING

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. H. Brinksma,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op donderdag 08 December 2011 om 12.45 uur

door
Luiz Olavo Bonino da Silva Santos
geboren op 08 Juli 1973
te Vitória, Espírito Santo, Brazilië

Dit proefschrift is goedgekeurd door:
prof.dr.ir. M. Akşit (promotor) , dr. L. Ferreira Pires (assistent-promotor),
en dr.ir. M. J. van Sinderen (assistent-promotor)

Abstract

Service-Oriented Computing (SOC) is a paradigm for the design, use and management of distributed system applications in the form of services. The vision of SOC is that services represent distributed pieces of functionality that can be combined (or composed, in SOC terms) to generate new functionality with more added-value. In an ideal scenario based on this vision, a service client expresses requirements to a software infrastructure, and the infrastructure discovers, selects and invokes services with no need for further human interaction. Non-functional requirements such as cost, trust and privacy, amongst others, should also be stated by the service client and resolved automatically by the infrastructure.

The SOC vision also overlaps with some of the characteristics of Pervasive Computing. In his seminal paper about Pervasive Computing (also known as Ubiquitous Computing), Weiser foresaw that computing, sensing and communication devices would be transparently embedded in our surrounding environment. These computer-enriched environments would grant access to information and services everywhere and anytime. Readily available information can contribute to the realization of the SOC vision specially by allowing a software infrastructure to gather information related to service execution without needing direct user interaction.

Although completely automatic service provisioning is the ultimate objective of SOC, much work still has to be done to realize this objective. Furthermore, the spreading of SOC and Pervasive computing will require these technologies to become more appealing to non-technical users in their daily life. Scenarios with significant numbers of available services, service providers and service clients, may give rise to issues such as: *(i)* how to express service requests in an intuitively appealing way (suited for non-technical end-users); *(ii)* how to tackle semantic interoperability issues among service requests, service descriptions and the internal interpretation of terms in the service operation that use different conceptual models; *(iii)* how to support the discovery, selection and invocation of services that fulfill the service client's goals in the least disruptive and invasive manner; and *(iv)* how to combine services executed by humans with services executed by computational systems.

In our work we have addressed scenarios where non-technical users are surrounded by communication-enabled intelligent devices and sensors, and where a large number of services is available. In these scenarios, additional support should be provided to the end-users to help them deal with the (possibly) overwhelming amount of decisions and interactions regarding service provisioning steps, namely, service request specification, service discovery, selection, agreement, composition and invocation.

In this thesis we present a conceptual framework to support dynamic service provisioning to non-technical users. The main contributions of our framework are: *(i)* to allow service clients to express their service requests using goals, which is a concept closer to their intuitive understanding than technical artifacts as, for instance, a WSDL document; *(ii)* to reduce the need of direct user interactions with the services through the use of information automatically gathered from the environment; *(iii)* to provide a domain specification language that supports domain specialists in modeling both domain concepts and services. Moreover, this domain specification language provides modeling primitives that allows the distinction between computational and social services, which are provided by computing systems and humans, respectively.

The concrete results of this thesis are: *(i)* the description and design of the goal-based framework for dynamic service provisioning; *(ii)* the design and the implementation of a prototype of the framework's software platform supporting the dynamic service provisioning; *(iii)* the definition of a foundational ontology providing the ontological grounding for the *(iv)* domain specification language.

The framework proposed in this thesis has been evaluated with representative case studies that cover the use of the framework in modeling application domains and the operation of the software platform to support dynamic service provisioning in these domains.

Acknowledgements

La riconoscenza è la memoria del cuore.
Gratitude is the memory of the heart.
– **Italian proverb**

Some say that a PhD is a lonely endeavor. Others, that the title earned after completing it should be your focus. Personally, I agree with the Buddhist teaching in which the path is the goal. About the loneliness, I could not disagree more. Although there is only one name as the author of a PhD thesis, it could have never been written without the support, collaboration, incentive and help of many people. Therefore, people has been an essential part of the path I took towards the PhD, and in this section I would like to acknowledge and thank those who were directly or indirectly involved in this accomplishment.

I would like to start by thanking my daily supervisors Luís Ferreira Pires and Marten van Sinderen for having the confidence to give me the opportunity to pursue the PhD at the University of Twente, and guided me all the way through. They gave me the freedom to research, and their attitude towards science inspired me. More than supervisors I consider them friends. And that says it all.

I would also like to thank my promoter Mehmet Akşit for receiving us in his Software Engineering research group and allowing us to continue the researches we were conducting. The transition from our extinct group to the Software Engineering group was smooth, in big part due to Mehmet's warm and welcoming attitude.

I would like to thank the members of my defense committee: Prof. dr. ir. Marco Aiello, Prof. dr. Willem-Jam van den Heuvel, Prof. dr. ir. Bart Nieuwenhuis, Prof. dr. Chris Visser, and Dr. Giancarlo Guizzardi. I am honored and grateful for devoting your time to read my dissertation and to participate in my defense.

It all started back in 2003, while I was doing my Masters,

and my supervisor José Gonçalves sent me information about the work he was conducting during his sabbatical at the University of Twente. I got interested and at the end my Master's dissertation was based on that research. At that time he started to incentive me to pursue a PhD abroad. As a trial, he suggested me to come to Enschede to finish my dissertation under the local supervision of Luís and Marten. Although short (only three months), this period was very productive and I could also have a hint of how would be the life if I would decide to go for the PhD. Zegonc, thank you very much for all your incentive, support and guidance. Our supervision meeting at a bar eating crabs were memorable. You are an example of leadership and inspiration earned by respect instead of by imposition.

Besides the support from the University of Twente, this short stay in the Netherlands would not have been possible without the participation of Giancarlo and Renata. Thanks so much for having me at your home and for everything else you helped me with in Brazil, in Italy and in the Netherlands. Without you none of this would have been possible.

The path from Brazil to the Netherlands was not straight, though. Before ultimately moving to Enschede, we took a 2-year *"detour'* through Trento, Italy. When we moved to Trento, we met Marco and Maira, who became our friends and helped us to cope with the adaptation to a new country and life circumstances. Gladly we are still in touch. I'll never forget our *almost* trip to Switzerland, the parties, the asparagus harvest, and all the help you gave us. Another friend we kept from this period is Yudistira (Yudis) Asnar. Your permanent smile and positive attitude towards life, together with the talent for cooking delicious Indonesian food will always mark our time in Italy.

We have also enjoyed a lot the international parties and the trips to nice places in the north of Italy with the colleagues from the University of Trento, and the wine tasting and dinners with the people from LOA. I have fond memories of the work at LOA. The philosophical discussions and the endless search for the most suitable and intuitive term to express a concept, and the constant pursuit of the intrinsic nature of things made me see the world in a different way.

From this period in Trento I would like to specially mention John Mylopoulus, Paolo Giorgini, Nicola Guarino, Laure Vieau, Claudio Masolo, Laurent Prevot, Stefano Borgo, Roberta Ferrario, Anna Perini, Maria Luisa Guerriero, Truong Thu Huong, Robert Trypuz, Mariana Doria, Ismênia Galvão, Michele Armano and

G.R. Gangadharan.

Then we moved to Enschede and the years I worked at the University of Twente for my PhD were simply great. Besides the period when our ASNA group was (unfairly, to say the least) dismantled, I cannot recall any down point. And a major part of having enjoyed so much is, certainly, due to the people I spent this period with. Therefore, I have a long list of people to thank for making my PhD time so enjoyable. From the ASNA group I would like to thank Professors Chris Vissers and Bart Nieuwenhuis, my supervisors Luís and Marten, the staff members Val Jones, B. van Beijnum, Dick Quartel (who I have the pleasure to work together again at BiZZdesign), I. Widya, Maarten Wegdam, our great secretary Annalies Klos, our software engineers Richard Bults and L. Helthuis, and, last but certainly not least, my PhD colleagues Patrícia Dockhorn Costa, Ricardo Neisse, Laura Daniele, Eduardo Gonçalves da Silva, Tom Broens, Hailiang Mei, Teduh Dirgahayu, Rodrigo Mantovaneli Pessoa, Kamran Sheikh and Pravim Pawar.

The major part of these over four years at the University of Twente were shared with my two paranymphs, friends and office neighbors Laura and Eduardo. Your friendship and support is important for me. Thank you for everything.

Besides the research work, the PhD has been an opportunity for social activities, like the funny conversations during and after lunch, the conference trips and group *uitjes*. I should thank the *tchês* Ricardo Neisse, Tiago and Ricardo Schmidt, as well as Giovane, Rafael, Laura, Eduardo, Anna Sperotto, Anna Kolesnichenko, Ramon, Idilio, Zambon, Ramin, Aiko, and other members of the DACS and the Formal Methods groups for these enjoyable moments.

After the ASNA group has been dismantled, Luís, Laura, Eduardo and I joined the Software Engineering group. I would like to thank all the members of the Software Engineering group for welcoming us in your group. *"Professore"* Ivan, thank you for the cheerful (and probably loud) chats and for sharing your metamodeling knowledge. Jeanette, thank you for all your help, support and kind words, the arrangements for the defense would not have been possible without you.

I have also been very lucky with my officemates Patrícia and Vikram. I could not expect to share so many hours with better people. Patrícia, you have been an inspiration for your competence, hard working and research skills. Vikram, your incentive and help have been valuable for my work. The delicious dinners at both Patrícia and Vikram houses together with their spouses

João Paulo and Asmita were memorable.

Family is important. The ties that have been established by genetic similarities have enduring consequences, easing the proximity of the relatives. I have been blessed by the family I was given. They were always by my side supporting me in whatever I was pursuing. I should start with my parents Martha and Luiz, and my stepfather Tarcísio. Unfortunately they are not here anymore to witness the end of the phase, but without their support, incentive, invaluable lessons, and inestimable love, I could never have started. A would also like to thank my brother Ricardo, his wife Fernanda, my sisters Barbara and Julia, and my grandparents Zelzy and Ricardo. I also consider mine the family of my wife and, therefore, would like to thank Marcelo, Agar, Ricardo, Ana, Carlo, Tina, Carlos, Rafael, Felipe, Célio, Lumena and Fernanda. This families are incredible. A special appreciation goes to my sister Barbara and my cousin Liana Mara. I know how hard it is to travel all the way from Brazil to the Netherlands. And you did it just to attend to my defense.

For all their support, I would also like to thank our extended *"European"* family Lothar, Laura, Eduardo and Nayeli, Ricardo and Kasia, Giovane, Rafael and Sanjka, Pablo and Flávia, Marcia and Ralph, Tiago and Liga, Idilio and Suen, Raida and Rob, Jailza and Henk, Fran and Henk, Lia and Robin, Sharon and Dick, Ramon and Rita, Zambon, Ricardo, Mariana, Mayra and Rocco, Ismênia and Riemer, Mariana and Anton.

I also would like to share this moment with my friends in Brazil and other parts of the world, specially Cesar, Juliana, Giancarlo, Renata, Gustavo Varejão, Rodrigo, Luiz Sergio, Fernanda, Zegonc, Ricardo Machado, Tarek, Talib, Tamer, André Bona, Marcelo Vasconcelos, Patrícia, João Paulo, Dede, Gustavo Demoner, Cynthia, Pablo and Flávia. Even far away your support and friendship are very important.

Among all these people, there is one single person with the biggest share of responsibility and participation during this journey, my wife Luciana. Your love, patience, support, incentive and well-placed observations were fundamental to give me the emotional balance I needed to overcome all obstacles. When things were difficult, you made them easy, when I thought things were easy, you made me realize that nothing is so simple. Not only this adventure would not have been possible without you, but also a big part of the accomplishments of these last 20 years we are together. And you gave me the best gift of all, our daughter Anna Martha. This book is for you.

Contents

List of Figures

Chapter 1

Introduction

This thesis contributes to the area of Semantic Service-Oriented Computing by proposing an integrated solution for semantic service provisioning in Pervasive Computing environments. The main objective is to provide dynamic service discovery, selection, composition (if necessary) and invocation based on end-users' needs. In this work we aim at providing abstractions that foster the expression of users' needs in a way that is closer to their intuitive perception (regarding non-technical end-users) than to technical terms such as data types, document format, etc. Much of this work is based on the use of ontologies that are applied at two distinct levels of abstraction, namely, to provide the conceptual model underlying a domain specification language (upper-level ontology) and to provide shared semantics throughout specific domains (domain ontology).

Moreover, by associating the target usage scenarios of our approach with Pervasive Computing environments, the issue of managing a large number of services and computing devices emerges. To tackle this issue we propose the integration of context-aware components aiming at transparently gathering users' contextual information to be used on service discovery, selection and composition as well as for supplying input information for service executions.

This chapter is organized as follows: Section 1.1 presents the background of this work; Section 1.2 presents the motivation for the work proposed in this thesis; Section 1.3 states the problem and the research questions, and presents the approach adopted in the research; Section 1.4 elaborates on the scope of this thesis and on the non-objectives, and finally Section 1.5 presents the structure of this thesis.

1.1 Background

Service-Oriented Computing (SOC) is a paradigm for distributed systems' architecture, design and deployment. The vision of SOC is that services represent distributed pieces of functionality that can be combined (composed, in SOC terms) to generate new (and more complex) functionality [Papazoglou et al., 2006]. In an idealistic scenario based on this vision, a service requester expresses requirements and a software infrastructure automatically discovers, selects and invokes services, without the need for further human interaction. Non-functional properties such as cost, trust, privacy, quality of service (QoS), and availability should also be taken into account and resolved automatically.

In business environments, the SOC vision translates into automatic cooperation among enterprises. One enterprise seeking business interactions with another enterprise can rely on its software infrastructure to automatically discover, select and invoke the appropriate services relying on selection policies and corporate objectives. For instance, one enterprise seeking alternatives for its supply chain partners could discover and select in a supply chain service directory new partners that provide the supplies it needs, based on its policies regarding supplier's availability, price, environmental certification, labor conditions, etc.

Although this vision is the ultimate goal of SOC, more work still has to be done to realize this vision. For instance, in scenarios with significant numbers of available services, service providers and service service requesters, issues related to semantic interoperability may rise. Below we present a (non-exhaustive) list of open questions in Service-Oriented Computing that are closely related to the objectives and interests of our research.

1. How to express service requests and, specially for non-technical service requesters, how to express these requests in an intuitive way?
2. How to tackle semantic interoperability issues among services and service requests that use different conceptual models and, therefore adopt concepts and terms with different meanings?
3. What sort of tooling is necessary to realize the SOC vision? This question includes:
   (a) How to supply service providers with appropriate tools to support the specification of (semantic) description of their services in a way that facilitates service discovery and invocation?
   (b) How to supply service requesters with tools to support

the expression of service requests and provide service discovery, selection, composition, invocation and adaptation?

(c) How to supply domain specialists and designers with tools to support the definition of domain-specific conceptual models? These domain-specific conceptual models aim at providing a shared knowledge about specific domains. Service providers and service requesters can share this knowledge, helping tackle the semantic interoperability issue.

Characteristics of Pervasive Computing can help achieve the SOC vision. In his seminal paper about Pervasive Computing (also known as Ubiquitous Computing) [Weiser, 1991] Weiser foresaw that computing, sensing and communication capable devices would be transparently embedded in our surrounding environment. These computer-enriched environments would grant access to information and services everywhere, anywhere. This easily available information can contribute to the realization of the SOC vision, specially by allowing a software infrastructure to gather information related to service execution without needing direct user interaction.

In recent years, the adoption of mobile communication and computing devices had an unprecedented increase and has moved us closer to the computer-populated environments described in [Weiser, 1991]. These devices evolved from rudimentary and limited (in terms of computing power, memory, battery life and communication capabilities) mobile phones and personal digital assistants (PDAs), to the current crop of smart phones, PDAs, laptops and netbooks. Moreover, it is not uncommon that one person uses more than one of these devices. Add to this scenario other computing-empowered appliances and accessories, such as Internet-enabled TV sets, digital pens, refrigerators, home media centers, network-attached storage (NAS), sensors, etc., and we have users facing difficulties to interact with, and to control and manage this plethora of devices and services. One possible solution for this issue is to provide infrastructural support to facilitate the access and management of these services and devices.

In this thesis we are particularly interested in scenarios where non-technical users are surrounded by computer-enabled devices and sensors, as well as have access to a large number of services. In such scenarios, additional support should be provided to the users to help them deal with the (possibly) overwhelming amount of decisions and interactions regarding service provisioning steps, namely, service request specification, service discovery, selection, agreement, composition and invocation.

1.2 Motivation

Since the emergence of Service-Oriented Computing, the availability of a large number of services has been identified as a challenge. In a scenario where a user has to search and select among a few (dozens of) services it is still feasible to carry out this search and selection manually. However, as the number of available service increases, additional support becomes necessary. Among the support possibilities for this issue we can mention the addition of semantics to service requests and service descriptions, which allows semantic reasoning [Domingue et al., 2005, Farrell and Lausen, 2007, SOA, 2008, Ope, 2010], and software support to discover, select and invoke services on behalf of the user [Fensel and Bussler, 2002, A-M, Ami]. Merging these two support possibilities, a software platform can perform service discovery, selection and invocation using semantic queries and/or semantic inference. This enriches the search and increases its accuracy, while automating the process.

However, adding semantics is not enough if the semantics for the terms in different artifacts (e.g., service requests and service descriptions) is not shared. For instance, if the semantics for the term *bermuda* in a service request refers to the set of islands in the Caribbean and the same term in a service description refers to a knee-length walking shorts, the resulting service discovery will be erroneous, resulting in a semantic interoperability problem. To tackle this issue, ontologies have been used in SOC to provide shared semantics [Berners-Lee et al., 2001].

The increasing adoption of mobile communication and computing devices together with the availability of a large number of services, raised issues regarding information overload and management. To tackle this problem, Weiser [Weiser, 1991] suggests that these computing-enriched environments should transparently act on users' behalf, by gathering and processing information, and should use this information to pursue the users' objectives. In order to achieve this suggested transparency, computers should be able to *gather the users' needs*, understand them and find means to satisfy these needs.

In the realm of Service-Oriented Computing, service users' needs are satisfied by executing services, while in order to properly gather the user's needs service requirements have to be explicitly expressed by the service requester. In order to understand these requirements we need to answer two questions: *(i)* how to semantically interpret requirements?, and *(ii)* how to relate the requirements to service offering descriptions?

Currently, many approaches, ontologies and languages are being developed to tackle requirements gathering and understanding [Rolland et al., 2007, Bresciani et al., 2004, Domingue et al., 2005, Van Lamsweerde et al., 1995], as well as providing platform support for service provisioning activities [Haselwanter et al., 2006, Ami, A-M, U-C, Goncalves da Silva et al., 2011]. However, they either lack a clear and comprehensive definition of service and its related concepts, do not provide an integrated solution or ignore the relations between computational services and services in the real world. These limitations hinder the applicability of these approaches in complex real world scenarios.

Following the increasing adoption of the service paradigm in Computer Science, several research efforts aim at providing reference architectures [Arsanjani et al., 2007, Laskey et al., 2009, Arsanjani et al., 2009], or providing shared conceptualization for services [de Bruijn et al., 2006, Martin et al., 2004]. These approaches consider the services as computer-executable processes (computational services), which are the realization of business processes. In this view we have a division between the social level, represented by the business processes, and the computational level, represented by the computational services. However, in some cases, business processes cannot be mapped onto computational services, but can be mapped onto services provided and executed by humans. For instance, the process of a heart transplant can only be mapped onto a service provided and executed by a heart transplant team, including the heart surgeon, anesthesiologist, assistant surgeons, nurses, etc.

In this heart transplant example, while the main service of actually performing the transplant cannot be automated by a computational service, some activities and events related to this service can be automated, such as booking the surgery (e.g., through the hospital's information system), being informed of the availability of the new heart (e.g., through the national organ donor's information system) and retrieving the patient's medical history (e.g., through an electronic medical record system). From this example we can conclude that there is a need for means to distinguish social and computational services, as well as to define the relations between the services at these two levels.

The widespread of Service-Oriented Computing leads to services being disseminated in environments where their users are not always technically literate. For instance, in the Ambient Intelligence and Home Health Care areas, devices and services are deployed in home environments, where the inhabitants generally do not have a technical background. In these environments, the service users would need a way of expressing their needs (that should be satisfied by services) closer to their expression capabilities than, for instance, a service request written in a technical language like the Web Services Description Language (WSDL).

The concept of goal has been used in different fields of study as a way of expressing what someone wants to be achieved. In the context of Service-Oriented Computing, this concept is also being used as a way to express service requirements. However, each approach defines the concept of goal and uses this concept in its own way. Therefore, a more thorough investigation of the concept of goal, its nature and its use is needed. In the scope of this thesis, this investigation is particularly necessary to evaluate the suitability of the concept of goal to be used (directly or indirectly) as a way to express service requirements by service users.

1.3 Research Design

In the work reported in this thesis, we have adopted the principles and guidelines of Design Sciences [Hevner et al., 2004]. Our work has been carried out following the steps of *(i)* defining the research problem, *(ii)* designing our research approach, and *(iii)* evaluating the research.

From the discussion of the motivation of our work (see Section 1.2), we have identified that the problem we intended to address is the support to service provisioning. From this problem we have defined that the main objective of this thesis is to *"create facilities to service provisioning"*. More specifically, we intend to facilitate the service provisioning in an environment with multiple services and devices targeting a class of service users with poor technical knowledge.

Since we consider complex environments with a (potentially) large number of services and devices to interact with these services, we also consider the distinction between social and computational services. Therefore, our objective of facilitating service provisioning is not limited to computational services but also encompasses social services and the relationships between computational and social services.

1.3.1 Research questions

By defining non-technical service users as the main target group in our objective, we have assumed that, to reach our objective of facilitating service provisioning, we could not be restricted to service requests being expressed in terms of technical solutions such as WSDL, SAWSDL or OWL-S, but should allow these users to express their requirements in terms that are closer to their common conceptualization. We have observed that the concept of goal is used in several areas of Computer Science to represent requirements of system users. We have extrapolated this observation to service provisioning, and considered that the concept of goal should be useful to express service requirements because it represents an abstraction that is closer to the purposes of the users as opposed to the technical solutions used to express service requirements.

The goal concept has been used in different areas of Computer Science, and it has been conceptualized, used and defined in different ways. Therefore, the following research question has been raised:

> **RQ1:** What are the best practices in using the concept of goal to capture user requirements?

Having an adequate conceptualization and usage strategy for goal is only part of the solution. From our objective, the service provisioning support should facilitate not only the service request but also other activities and events related to service provisioning. A comprehensive solution to service provisioning would require a set of components to provide the intended support. To adequately design such solution, the following research question has been raised:

> **RQ2:** Can we devise components to provide facilities for service provisioning in multiple domains?

Our solution includes a software platform to automate activities related to service provisioning and, therefore, facilitating service users to access services. Software platforms can support service users in discovering, selecting, negotiating and invoking services. In order to be more effective in supporting these activities, the software platform should have the means to understand the service user's requests and the service descriptions. This understanding allows the software platform to reason about the terms included in the user requests and service descriptions, and achieve not only syntactic matches between terms in these artifacts, but also semantic matches, which, in most cases, improves the success rates of the matches. Semantic matching and reasoning can be realized by means of semantic annotations on the terms contained in service requests and service descriptions.

Our proposed software platform performs reasoning based on semantic annotations in the service requests and service descriptions. These annotations are based on semantic domain specifications, expressed in terms of domain ontologies. The need for semantic annotations and semantic domain specifications raised the following research questions:

> **RQ3:** Which techniques are capable of supporting and enabling domain specifications and semantic annotations?

> **RQ4:** How to structure a platform to support dynamic service provisioning using domain specifications and semantic annotations?

Finally, after designing and implementing our solution, we need a strategy to evaluate our framework and the knowledge obtained from this research. We should define whether our solution properly and satisfactorily solves the problem and analyze what we have learned from this process. These issues lead to our last research question:

> **RQ5:** How can we assert whether our proposed framework satisfies the objectives of our research?

1.3.2 Approach

We have defined a research approach to answer our research questions, and, by answering them, reach the objectives of this thesis. To answer the research question on the best practices in using the concept of goal to capture user requirements (RQ1), we have conducted a literature investigation on how the goal concept is defined and used. This investigation aimed at learning and clarifying the conceptualization for goal and providing inspiration for how we should define and use this concept in the scope of our work and objectives. This study has been conducted on several areas of Computer Science such as Artificial Intelligence, Agent-Oriented Computing, Requirements Engineering and Service-Oriented Computing.

The research question on the components to facilitate service provisioning (RQ2), has been answered through a literature study on the areas of Service Ontologies, Service Supporting Platforms, Semantic Service Discovery and Composition, Context-Awareness and Pervasive Computing.

From our service-related study, we observed that a comprehensive conceptualization for services is still missing. Moreover, the study showed us that current approaches in the Service-Oriented Computing area mainly focus on defining services as computer-executable processes, and the relations between computational services and social services are either not defined or are unclear.

From the understanding gathered in these literature studies, we could devise a high-level solution for the problem of facilitating service provisioning. In this solution we have designed a framework for semantic service provisioning. We use the term *framework* as a *conceptual structure to define a solution for a problem*, i.e., the framework defines boundaries as to what is suitable for addressing particular problems. More specifically, our proposed framework defines a set of technologies and techniques to cope with dynamic service provisioning.

To guide the design of the framework we have defined a set of usage scenarios involving dynamic service provisioning in environments containing a large number of services and devices. From the analysis of current solutions in the literature and of our usage scenarios, we have defined the requirements for the framework and justified its architectural design.

To answer the research questions on the techniques to support semantic annotations (RQ3), we have proposed a domain specification language. To provide the conceptual model underlying the domain specification language, we have proposed a foundational ontology by extending an existing one. Our foundational ontology defines primitives necessary for domain specification, such as agent, goal, resource, action and event, as well as primitives necessary for service specification, such as service, task, service provider, service client and service commitment. This domain specification language also distinguishes between social and computational services and defines the relations between these two types of services.

From the literature study and the framework requirements, we have defined the architectural design of a software platform, answering the research question on the structure of a platform to support dynamic service provisioning (RQ4).

To answer the research question on how to assert the framework compliance with our objectives (RQ5), we have defined a case study based on a set of usage scenarios. In this case study we have evaluated whether the domain specification language is capable of defining domain specifications, and whether the prototype of our software platform supports service provisioning.

1.4 Scope and Non-Objectives

In this thesis we focus on the architectural design of a framework for semantic service provisioning and on the definition of the service-related concepts of the foundational ontology and its derived domain specification language. The acknowledgment of two different classes of services whose related tasks are performed by computational devices and services whose tasks are performed by humans, the understanding of the relationships between these two classes of services, and the support for modeling domains with this distinction explicitly and clearly defined has also been a focus of our work.

Regarding the software infrastructure for the realization of the proposed framework, we focus on the architectural design of the infrastructure, reusing when possible existing software components and technologies.

In this thesis we did not extensively addressed implementation issues of Web services or any other middleware technology, such as concurrency, quality of service, conflicts, etc. Since the proposed domain modeling language aims at supporting the definition of domain ontologies for annotation purposes, we did not focus on the underlying foundational ontology formalization. Although a significant part of the foundational ontology has been formalized, the event and social aspects of the ontology still lacks a thorough formalization in terms of theories and axioms.

1.5 Structure

The structure of this thesis reflects the approach we followed in this research. The remainder of this thesis is structured as follows:

- *Chapter 2 - Goal Definition and Usage.* This chapter investigates the definition of the goal concept and its usage in different areas of Computer Science. This chapter discusses the goal definition based on the evaluation of the different approaches and their applicability to the objectives of this thesis.

- *Chapter 3 - Goal-Based Service Framework.* This chapter motivates and presents our framework for service provisioning support. The chapter discusses the frameworks architectural components, related techniques and methods, and the relationships among the components.
- *Chapter 4 - Goal-Based Service Ontology.* This chapter discusses the Goal-Based Service Ontology concepts and relations in details, and introduces and justifies their ontological foundation.
- *Chapter 5 - Context-Aware Service Platform.* This chapter presents and discusses the architectural components of the Context-Aware Service Platform.
- *Chapter 6 - Case Study and Evaluation.* This chapter evaluates the applicability of the the Goal-Based Service Framework by means of a case study. In the case study we modeled scenarios in the scope of the Health Care application domain, and defined service provisioning support to the service clients in this domain.
- *Chapter 7 - Conclusions and Future Work.* This chapter concludes this thesis by discussing the main results of this work and the drawbacks we have encountered. Finally, we identify topics that require further investigation as part of future work.

Figure 1-1 depicts the structure of this thesis, and it shows in which chapters the research questions are answered.

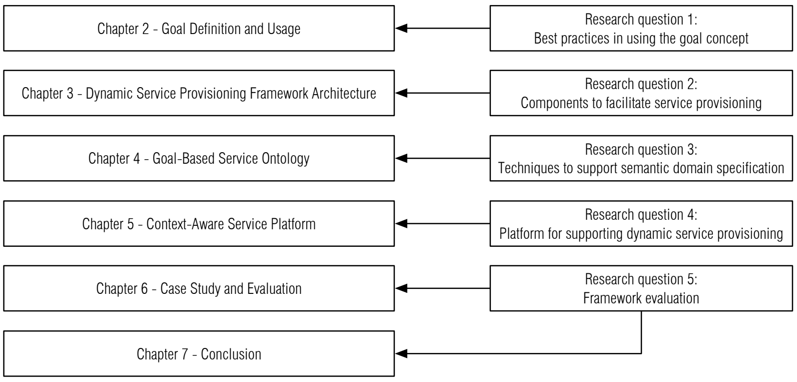

Chapter 2 - Goal Definition and Usage
Research question 1: Best practices in using the goal concept
Chapter 3 - Dynamic Service Provisioning Framework Architecture
Research question 2: Components to facilitate service provisioning
Chapter 4 - Goal-Based Service Ontology
Research question 3: Techniques to support semantic domain specification
Chapter 5 - Context-Aware Service Platform
Research question 4: Platform for supporting dynamic service provisioning
Chapter 6 - Case Study and Evaluation
Research question 5: Framework evaluation
Chapter 7 - Conclusion

Figure 1-1 Thesis structure and relation between chapters and research questions

Chapter 2

Goal Definition and Usage

The concept of goal is pivotal for the work reported in this thesis. It has been used as an abstraction to represent what service clients want from the services but also to guide service discovery, selection, composition and execution. During the development of the work reported in this thesis, we have studied, analyzed and discussed several different approaches that are also based on the concept of goal. This process helped us decide which definition to follow or extend based on the usage we wanted to give to the concept of goal in our framework.

This chapter discusses the concept of goal, its various definitions in different areas where the concept has been applied and the usage of the concept in these areas. This chapter begins by briefly introducing the concept of goal in Section 2.1. After this introduction, the concept of goal and its usage are discussed in the areas of Artificial Intelligence in Section 2.2, Agent-Oriented Computing in Section 2.3, Requirements Engineering in Section 2.4 and Service-Oriented Computing in Section 2.5. Finally in Section 2.6 we conclude the chapter by analyzing the suitability of the different definitions and uses for the concept of goal in the scope of this thesis objectives.

2.1 Informal definition

The concept of goal has several different definitions varying from *"the result of scoring"* and *"the physical structure that defines where the score is achieved"* in some ball games to *"a statement of intent for the direction of the business"* in business administration.

The definitions strongly depend on the application area to which this term is applied. Narrowing down to the Computer Science domain, a variety of definitions of the goal concept can also be found. The concept of goal has been used in Computer

Science in many different areas such as Artificial Intelligence, Requirements Engineering, Formal Ontologies and, more recently, Service Oriented Computing. Goal-based analysis has been used in different areas of Computer Science to identify stakeholder's objectives, determine requirements for software systems and guide system's behavior. This generic approach ranges from techniques to accurately identify goals as presented in [Bresciani et al., 2004, Anton, 1996] to the use of goals for service composition presented in [Padgham and Liu, 2007, Vukovic and Robinson, 2005a, Zhang et al., 2006].

In the following sections we present and discuss different approaches defining and using the concept of goal. The approaches discussed have been selected based on their relevance in their respective fields and the relation they have with the main objective of this thesis, namely, to facilitate service provisioning. Moreover, the sequence of the research areas in which the presented approaches are related follow a historical, chronological and incremental evolution. For instance, the approaches presented in the area of Artificial Intelligence were used as foundations for the work carried out in the areas of Agent-Oriented Computing, Requirements Engineering and Service-Oriented Computing. In some of the presented approaches we have included parts of the formalization of the defined concepts and relations. However, since some of these formalizations are extensive and complex, we have included only the parts that are relevant for our analysis and discussion.

2.2 Artificial intelligence

In the Artificial Intelligence (AI) realm, goals are considered in problem solving research. The main motivation for problem solving research is that in order to achieve their goals, agents frequently need to act on the world. Considering computational agents, approaches and techniques were necessary to support the generation of appropriate actions. The AI community has developed two complimentary approaches to generate these actions: planning and situated actions [Weld, 1994]. Planning is mostly used when a number of actions must be executed in a coherent way (pattern) to achieve a goal or when there are complex interferences of one action on others. Conversely, situated actions are used when the best possible action can be easily computed from the current state of the world, i.e., when no look ahead is necessary because actions do not interfere with each other. These approaches differ, since planning uses a partial description of the state of the world while situated actions use the complete state.

Artificial Intelligence planning focuses on automated techniques for determining plans by combining several operators for solving complex problems. A plan is defined as a sequence of actions that transforms the world from an initial state of a problem description into the final desired state [Nau et al., 2004, Russell and Norvig, 2009].

Planning can be divided in two branches, namely classical planning and neoclassical planning [Nau et al., 2004]. The former defines planning techniques within environments that are fully observable, deterministic, finite, static and discrete, while the later define planning techniques within partially observable and stochastic environments.

The classical planning approach considers the initial state of the world, a set of actions and their (deterministic) effects, and a sequence of actions (a plan) to achieve a certain goal state. Forward and backward chaining can be applied as the underlying inference mechanisms for classical planning algorithms. Forward chaining iteratively applies a possible operator O to a set of input parameters and preconditions defined by the initial state aiming to reach the goal G. If applying the operator O does not solve the problem, i.e., the desired goal state is not reached, then a new query can be computed from G and O', and the whole process is iterated. Backward chaining starts from the desired goal state G (the final state) and iterates back to the initial state using the operator O_2 to provide at least one of the required parameters. Applying the operator O_2 may result in new parameters being required, which can be formalized as a new goal G' and, once more, the process is iterated until a solution is found. The solution is a plan defined in terms of a sequence of actions.

In these approaches, goal is defined as a *"description of a world state that is expected to be realized"* [Russel and Norvig, 2002]. More specifically, in planning a goal is defined as a partially specified state, represented by a conjunction of positive ground literals [Fikes and Nilsson, 1971]. In other words, a state S satisfies a goal G if S contains all the atoms in G (and possibly others as well) [Russel and Norvig, 2002]. For example, the state $Rich \wedge Famous \wedge Unhappy$ satisfies the goal $Rich \wedge Famous$.

2.3 Agent-Oriented Computing

Agent-Oriented Computing deals with systems in which autonomous computational elements called agents sense their environment and act on it by pursuing their own agenda [Franklin and Graesser, 1997]. The research in this field started in the early 1970's and aimed at developing a new paradigm for system design along with related technologies incorporating socio-psychological insights [Shoham, 1993, Wooldridge et al., 2000]. Mimicking the problem solving behavior of humans, a software agent acts autonomously on its environment and collaborate with other agents, if this collaboration and acting is beneficial for achieving the agent's individual objectives.

2.3.1 Agent properties

The operation of agents is based on the following properties defined according to models from socio-psychology on human behavior [Wooldridge and Jennings, 1995]:

- *Autonomous* - an agent acts self-directed and controls its own actions;
- *Social ability* - an agent interacts by communicating with humans or other agents for collaborative problem solving;
- *Reactivity* - an agent observes its environment and reacts to changes in this environment;
- *Pro-activity* - an agent performs its tasks in a goal-driven manner, i.e., it acts according to its goals.

In the agent-oriented research, intelligent agent architectures have been developed using artificial intelligence techniques to simulate human behavior in software agents. In this context, a goal-based agent model is used to determine multiple agent's behavior by means of goals as the desired final state to be reached. Moreover, by relating actions and goals, these models also provide knowledge about the effects of applicable actions.

2.3.2 BDI Model

From the approaches developed based on goal-based agent models, one that is relevant to the scope of this thesis is the Belief-Desire-Intention (BDI) model. The BDI model is a philosophical theory on the motivation and behavior of rationale action by humans [Bratman, 1987]. In summary, beliefs represent information about the world that an agent considers to be true, desires represent the objectives that the agent wants achieved, and intentions represent desires for which the agent is committed to have achieved by means of actions. These three concepts are mental states whose interrelations determine the rationale of the agent's behavior. In other words, by understanding the beliefs, desires and intentions of an agent, one can comprehend this agent's behavior and the reasons for this behavior.

One relevant feature of the BDI model for rationale action is that an agent does not determine a complete (and fixed) resolution plan for a goal starting from its initial state and ending at the desired final state represented by its desire. Instead, at each performed action, more knowledge is gained, which may change the agent's beliefs and drive the agent towards different (and not previously planned) actions. For instance, one agent may consider that driving through a certain route is the best way for achieving its desire of fastly arriving at a certain location fast. However, during the trajectory, the agent is informed of a traffic jam in a part of the route he had planned to follow. With this new information the agent may change its beliefs about the fastest route towards the intended location and decide to follow an alternative path which, under normal circumstances would not be the optimal choice.

The BDI approach has been specified through logical formalisms such as Intention Logic [Cohen and Levesque, 1990] and BDI logics [Rao and Georgeff, 1991], providing the basis for practical reasoning about agent structures. As an alternative, practical reasoning is directed towards actions, contrarily to theoretical reasoning which is directed towards beliefs, and is defined in [Bratman, 1987] as *"a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes"*.

Both Intention Logic and BDI Logics adopt possible world semantics. For instance, possible worlds are used to represent that when an agent believes in something, this means that there is a world W accessible to the agent in which his belief holds.

Intention Logic

In Cohen and Levesque's Intention Logic, intentions are considered as a mental state, determining the goal resolution behavior. The Intention Logic's goal resolution behavior has the following properties:

1. Intentions pose problems for agents and the agents need to determine solutions for these problems;

Table 2-1 Atomic modalities in Intention Logic

Operator	Meaning
$(Bel \quad i \quad \varphi)$	agent i believes φ
$(Goal \quad i \quad \vartheta)$	agent i has goal ϑ
$(Happens \quad \alpha)$	action α will happen
$(Done \quad \alpha)$	agent action α just happened

2. Intentions may be used as "filters" for adopting other intentions, and they must not conflict. For instance, if an agent has an intention a, the agent is not expected to adopt intention b such that a and b are mutually exclusive;
3. Agents aim at satisfying their intentions and they can try again if an attempt fails;
4. Agents believe their intentions can be satisfied, i.e., an agent believes that there is at least a course of action in which the intention can be satisfied;
5. Agents do not believe they will not satisfy their intentions. For instance, , i.e, an agent would not adopt an intention if it believes that the intention is not satisfiable;
6. Under normal circumstances, agents believe they will satisfy their intentions.
7. Agents do not necessarily intend all the expected side effects of their intentions. For instance, a patient may intend to treat his tooth but not necessarily intends the pain that may be caused by the treatment.

The three first properties have been adopted from Bratman's philosophical model [Bratman, 1987].

Intention Logic (as well as BDI Logics, presented further in this section) uses modal logics associated with possible world semantics. The *modalities* are used to express constructs that could not be expressed in first-order logic. Table 2-1 shows the core modalities of Intention [Cohen and Levesque, 1990].

For convenience, Intention Logic adopted a set of abbreviations for the several logic expressions, among which we highlight the *Eventually* ($\Diamond$) and the *Always*($\Box$) abbreviations. *Eventually* is defined as $\Diamond\alpha \equiv \exists x(Happens x; \alpha?)$. In other words, $\Diamond\alpha$ is true in a possible world if there is some sequence of events after which α will hold. *Always* is defined as $\Box\alpha \equiv \neg\Diamond\neg\alpha$, and $\Box\alpha$ means that α is henceforth true in the course of events.

For instance, the expression $(Bel \quad john \quad \diamond hasWife(mary))$ denotes that John believes that he eventually will have Mary as his wife. In this example, the primitive *Bel* is a modality representing a belief of an agent, and the symbol $\diamond$, a common symbol for eventuality.

To talk about propositions that are not true now, but will become true, Intention Logic defines $(Later \quad p) \equiv \neg p \wedge \diamond p$ [Cohen and Levesque, 1990]. Moreover, to state constraints on courses of events, Intention Logic defines $(Before \quad p \quad q) \equiv \forall c(Happens \quad c; q?) \supset \exists a(a \leq c) \wedge (Happens \quad a; p?)$. The $Before$ definition states that p comes before q if, whenever q is true in a course of events, p has been true.

Intention Logic also defines a construct that determines rational behavior of agents, namely the *persistent goal (P-Goal).* A persistent goal represent an agent's desire that will be kept until it is achieved or considered unachievable. The expression 2.1, represents the definition of persistent. This definition captures the *"fanatical"* commitment between an agent and its goal, which establishes that an agent will not give up fulfilling its goal until it believes that the goal has been fulfilled, or until it believes that the goal will never be fulfilled.

$$\begin{aligned}(P - Goal \quad x \quad p) \equiv\ & (Goal \quad x \quad (Later \quad p)) \\ & \wedge \quad (Bel \quad x \quad \neg p) \quad \wedge \\ & [Before((Bel \quad x \quad p) \quad \vee \\ & (Bel \quad x \quad \Box\neg p)) \\ & \neg(Goal \quad x \quad (Later \quad p))] \qquad (2.1)\end{aligned}$$

Intention is defined in Intention Logic as a *"a kind of persistent goal".* In [Cohen and Levesque, 1990], the authors present two different definitions of intention. In the first, the subject of the intention is an action and is expressed as $(Intend \quad x \quad a)$, where x is the agent and a is the action. In this definition the agent intends to perform an action and it is committed to perform it. However, to avoid the situations where an agent would perform an action accidentally or unknowingly, the $Intend$ definition entails that the agent commits to believe in performing the action and then performing it.

The second definition of intention has a state of affairs as its subject as is expressed as $(Intend \quad x \quad p)$, where x is the agent and p is the intended state of affairs.

The definition of intention in Intention Logic has been criticized as either not being fully compliant with the theoretical model of Bratman, or for not being able to represent the intended properties and relationships. According to Hoek and Wooldridge [van der Hoek and Wooldridge, 2003], in Intention Logic intentions are reducible to beliefs and desires, only denoting temporal sequences of these two concepts. However, Bratman's model place intentions as first-class mental attitudes influencing rational behavior as much as desires and beliefs.

BDI Logics

Rao and Georgeff's BDI logics aims at overcoming the deficiencies of Intention Logic. BDI logics is based on (branching time) temporal logic (CTL*). Similar to the Intention Logic, the BDI logics define the modal operators *Bel* (agent believes), *Des* (agent desires) and *Intend* (agent intends), and the temporal operators X (next), U (until), F (sometime in the future or eventually) and E (some path in the future or optionally). However, the main difference in BDI logics is that this approach treats intentions as first-class constructs, and that beliefs, desires and intentions are understood in terms of possible worlds.

In BDI logics, the mental states of the agents are represented in structures named *time trees*. A time tree represents the current situation of an agent at a point in time, having one single past (the known behavioral history of the agent) and a branching future [Rao and Georgeff, 1998]. This branching future, also called accessible worlds, represents the set of all possible situations accessible to the agent with its current knowledge. The transition between accessible worlds are represented by events. Therefore, at any point in time, an agent has possibly multiple belief-, desire-, and intention-accessible worlds. Using BDI logics, one is able to express how the beliefs, desires and intentions of an agent evolve over time (or rather over possible time lines).

In BDI logics, a set of relationships holds between an agent's mental states, supporting practical reasoning, e.g., belief-desire compatibility and desire-intention compatibility. While the former states that if an agent desires φ it also believes φ $((Des \quad x \quad \varphi) \Rightarrow (Bel \quad x \quad \varphi))$, the later states that if an agent intends φ it also desires φ $((Intend \quad x \quad \varphi) \quad \Rightarrow \quad (Goal \quad x \quad \varphi))$. Being accessible worlds, when an agent *"believes"* in a desire, it means that the agent believes that the state of the world representing the desire is reachable.

Assuming that in BDI logics, desire is equivalent to a goal [Meyer, 2003], we can say that both BDI logics and Intention Logic consider goal as a state intended to be achieved by an agent. In other approaches within the agent-oriented computing community the term goal does not have a standard definition. In [Moghadasi et al., 2007], a goal is defined as a *"state with highest utility and* *an agent must choose the course of actions to reach that goal".* In [Rosenschein and Zlotkin, 1994], two types of goal models are presented, namely the task-oriented goal model and the state-oriented goal model. The former defines goal as a *"fixed list of tasks"* and the goal satisfaction is achieved when the agent finishes all these tasks. The later defines goal as a *"final state that the agent tries to achieve by moving from its initial state through a defined and finite sequence of intermediary states".*

Goals and tasks

Although these definitions of goal differ, a common agreement in the agent community is that goals are part of an agent's intentional properties and that the goal owner is committed to satisfy the goal. This satisfaction can be achieved either by the agent acting itself to satisfy the goal or by delegating the goal satisfaction to other agents. Agents commonly represent the stakeholders of a domain, being these stakeholders humans, organizations or automated systems. Agents can delegate the fulfillment of a goal or the execution of a task. A goal delegation represents a situation where an agent has a goal, but for some reason it is not capable of fulfilling this goal, and delegates it to be fulfilled by another agent. Task delegation occurs similarly. The main difference between goal and task delegation is that while in the goal delegation the goal owner wants his goal fulfilled no matter how, in task delegation the task should be carried out in the way defined by the task owner [Castelfranchi and Falcone, 1998].

2.4 Requirements Engineering

Requirements engineering (RE) can be understood as *"process of discovering the purpose of a system, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation"* [Nuseibeh and Easterbrook, 2000]. In [Zave, 1997], the author presents the following definition for RE:

> Requirements Engineering is the branch of Software Engineering concerned with the real-world goals for functions of and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.

This definition highlights the relationship between real-world goals system specifications. These goals represent the rationale, or the *why*, for system development and the *real-world* denotes that systems are developed to satisfy goals from stakeholders in the real-world.

In contrast with Requirements Engineering, in the Agent-Oriented Computing area goals are elements that are part of the system, i.e., the agents that comprise the system have their behavior guided by their goals. In the Requirements Engineering area, goals are identified and studied to represent the purpose of the system under development and are translated into artifacts that guide the development of systems that may or may not incorporate these identified goals as their internal elements. For instance, requirements engineering artifacts may be used to develop systems under the agent-oriented paradigm. In this case, the identified real-world goals are translated into goals of the system's agents. Alternatively, the same requirements engineering artifacts may support the development of a system using functional or object-oriented paradigms. In this case the real-world goals are (indirectly) translated into functions or objects, i.e., if one traces the documentation from the code to the requirements artifacts, he should be able to identify that a certain function or object has been defined because it contributes to the fulfillment of a given goal.

From the requirements engineering definition above, several approaches related to area emerged aiming at providing methods and techniques to explicitly represent goals. Among these approaches, we have selected KAOS, i*/Tropos and GBRAM to discuss, because they are the most prominent approaches in this area.

2.4.1 KAOS

The Knowledge Acquisition in autOmated Specification (KAOS)[1] [Dardenne et al., 1993] is a goal-oriented approach aiming at the formal modeling of functional and non-functional system requirements. The KAOS approach is based on three main components: *(i)* a conceptual model and a language for acquiring and structuring requirements models, *(ii)* a set of strategies for elaborating requirements models, and *(iii)* an automated assistant to provide guidance in the acquisition process.

[1]Lately the KAOS acronym stands for Keep All Objects Satisfied

Main components

The KAOS conceptual model is the metamodel for the language used in the approach and includes concepts and relations used in the requirements models, such as goal, agent, assignment relation, etc. The acquisition strategies correspond to the methodological aspect of the approach, and define the steps for creating and for acquiring the components of the requirements models. These components of a requirements model are domain-specific instances of the concepts and relations defined in the metamodel.

An acquisition strategy specifies a way of traversing the metamodel graph to acquire instances of its nodes and links, i.e., it prescribes a way to follow the concepts and relations of the metamodel in order to identify elements of the application domain that match these concepts and relations. For instance, the metamodel can be traversed backwards from the agents available in the system to the objectives to be fulfilled. The strategies can be automated by the acquisition assistant.

Methodology

The KAOS methodology aims at supporting the whole process of requirements elaboration, from the high-level goals to be fulfilled by a system, to the requirements, objects and operations to be assigned to the various agents in the system. The KAOS language supports requirements modeling in terms of goals, constraints, objects, actions, agents, events, etc.

In KAOS, goals are specified at a high-level, and then each high-level goal is decomposed into a set of sub-goals. This decomposition process continues until the goals cannot be further decomposed. KAOS introduces AND/OR decompositions, so that each level of decomposition is either disjunctive (OR) or conjunctive (AND). A disjunctive high-level goal is satisfied when a single sub-goal is satisfied, whereas a conjunctive high-goal is satisfied when all of its sub-goals are satisfied. KAOS also specifies that satisfaction of goals can contribute to or degrade the satisfaction of other goals in the system. These effects occur when agents in the system have different motivations, perceive information differently or when goals interact with each another. The KAOS model allows information to be specified in such a way that enables the system to determine which agents in the system are best capable of pursuing specific goals.

Conceptual metamodel

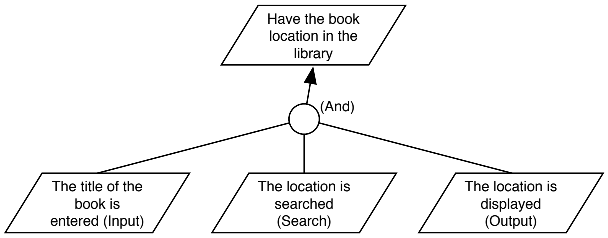
In KAOS, the software under construction is specified as an instance of a conceptual metamodel where abstractions such as goals, requirements, operations, agents, or entities are semantically linked. Moreover, KAOS provides a formal assertion layer (Darimont and van Lamsweerde, 1996) for inferring specifications from requirements (van Lamsweerde and Willemet, 1998) and reasoning about goal satisfaction (Letier and van Lamsweerde, 2004).

A KAOS model (an instance of its conceptual model) comprises four complementary and interrelated views that are iteratively prepared [Letier and van Lamsweerde, 2002b]:

1. A goal model the defines goals to be achieved by the software and their refinements;
2. An object model that defines the domain entities, relationships and attributes that are relevant to goal formulations;
3. An agent model that defines the responsibilities and interfaces of the various agents forming the system (humans, devices or software); and
4. An operation model that defines input-output relationships between operationalizations of requirements and identified objects.

In the goal model, the identified goals are represented as a hierarchical graph using AND/OR-decompositions of discrete high-level goals down to precise leaf-level requirements [Van Lamsweerde et al., 1995]. Figure 2-1 provides an example of a KAOS goal model for a library system where the high-level goal *"Have the book location in the library"* is decomposed into the sub-goals *"The title of the book is entered"*, *"The location is searched"* and *"The location is displayed"*. In this example an AND decomposition has been used.

Figure 2-1 An example of the KAOS goal model


Have the book location in the library
(And)
The title of the book is entered (Input)
The location is searched (Search)
The location is displayed (Output)

Once the goals are identified and modeled, the object and the agent models can be created. The object model collects objects, attributes, and their relationships, while in the agent model, agents are assigned responsibility for achieving the goals [Letier and van Lamsweerde, 2002a]. Agents and objects are used when the operation model is prepared. In this last step, operations that describe the behavior of the system in specific situations are derived from requirements [van Lamsweerde and Willemet, 1998]. Situations in which operations work are determined by events (arrowed rectangle) that cause the operation (oval) and entities (rectangles) that serve as information input. Each operation must be performed by an agent (hexagon). Figure 2-2 provides an example of an operation model.

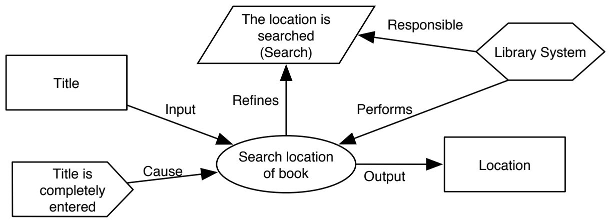


Figure 2-2 An example of the KAOS operation model

Goal concept

The concept of goal is defined in KAOS as a *"nonoperational objective to be achieved by the composite system"* [Dardenne et al., 1993].

The KAOS approach combines different levels of expression and reasoning. A semi-formal level is used for modeling and structuring goals, a qualitative level is used for alternative selection, and a formal level is used for more accurate reasoning [Lapouchnian, 2005]. To support these different levels, the KAOS language combines semantic networks for conceptual modeling of goals, assumptions, agents, objects and operations in the system, and linear-time temporal logic for the specification of goals and objects, and for state-base specifications for operations.

The models in KAOS normally have a two-level structure: the outer graphical semantic layer and the inner formal layer. Concepts are declared together with their attributes and the relationships between the concepts at the graphical layer, while concepts are formally defined at the formal layer. For instance, the formal layer of the graphical layer shown in Figure 2-1 is:

Goal Achieve[Have the book location in the library]
Concerns Title, Location
RefinedTo Input, Search, Output
InformalDef. The location of a book is found if it is displayed
FormalDef.$\forall t : Title, l : Location : Output(l) \wedge$
$SearchedTitle(t) \wedge IsLocationOfBook(l, t)$
$\Rightarrow BookFound(t)$

2.4.2 i* and Tropos

Traditionally, software development methodologies have been inspired and driven by the dominant programming paradigm of the day. Examples are structured programming, which inspired structured analysis and design [DeMarco, 1979] or more recently, object-oriented programming, which originated object-oriented analysis and design [Wirfs-Brock et al., 1990]. Because development methodologies follow predominantly the programming paradigms, a conceptual gap is created between the software system and the operational environment where the software runs. This is because the concepts used to model the environment of the software system being developed are constrained by the concepts of the programming paradigm, and are often distant from the environment itself. This gap follows from the difference between the conceptualization used express the technical aspects involved in computer systems (e.g., classes, objects, tables, functions, among others) and the conceptualization necessary to express the operational environment of organizations (e.g., roles, strategic goals, stakeholders, among others).

Using the same concepts to align requirements analysis with system design and implementation should reduce the aforementioned conceptual gap between the concepts used in artifacts of the different development phases. This also permits the use of coherent toolsets and techniques for system development. For instance, one can choose whether to perform the conceptual alignment based on the concepts used in artifacts of the implementation phase, or based on the concepts used in the artifacts of the requirements phase.

In this scenario i* and Tropos [Yu, 1997, Bresciani et al., 2004] have been created, aiming at offering a requirements-driven development framework based on the concepts used during early requirements analysis.

i* models and concepts

i* is a framework designed for modeling and reasoning about organizational environments and their information systems. The framework comprises two main modeling components, namely the Strategic Dependency (SD) model and the Strategic Rationale (SR) model. The SD model describes the dependency relationships among the stakeholders in an organizational context, while the SR model describes the stakeholders' interests and concerns, and how they might be addressed by systems.

i* focuses on the early requirements phase of Requirements Engineering, and applies the concept of intentional agents from the Agent-Oriented community. In i*, intentional agents are called actors and have intentional properties such as goals, beliefs, abilities and commitments. Actors depend on each other for achieving goals, performing tasks and providing resources.

In 2008, the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T) gave final approval for recognizing the User Requirements Notation (URN) as an international standard [ITU-T, 2008]. The URN is a language consisting of Use Case Maps (UCM) [Buhr and Casselman, 1996], a scenario modeling notation, and the Goal Requirement Language (GRL) [Liu and Yu, 2004], a variant of the i* framework.

Tropos

Tropos is a software development methodology, in which concepts of the agent paradigm are used along the whole software development process. Tropos is built on top of i*, i.e., it uses the i* modeling notation as modeling tool in the methodology.

Tropos follows the agent-oriented approach by qualifying intentional entities with properties such as autonomy and social ability, mental states such beliefs, intentions and goals, among others. These abstractions are claimed to allow a more natural understanding of organizations, their members and the social relationships present in these environments.

In Tropos, the modeling activities start at the knowledge level instead of at the procedural level, like other traditional software development methodologies. Two of the distinctive features of Tropos are:

1. Use of the mentalistic notions founded on belief, desire and intention (BDI) agent architecture in all software development phases, from early requirements down to implementation;
2. Central role of the early requirements analysis phase.

In the early requirements phase, models the organizational environment, in which the system will be embedded, are created. These models convey information like the stakeholders and their strategic goals and interrelationships. This phase precedes the requirements specification of the system under development. By confronting the models of the early requirements phase with the models of the specification of system under development, it is possible to assess whether this system meets the identified organizational goals and why the functionality has been assigned to the system.

The main concepts in Tropos are actor, goal, task, resource and social dependency. Figure 2-3 depicts the Tropos' graphical representation of actor, goal, task and resource.

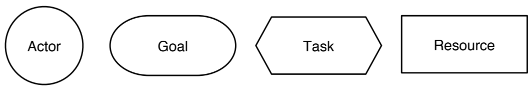


Figure 2-3 The graphical representation of the Tropos main concepts

Tropos actors

The Tropos concept of *Actor* matches the agent-oriented concept of an autonomous and social entity. An actor can represent a physical person, an organization, an organizational role, a type of intentional entities, a software system or component, or any other entity having strategic goals and intentions. In order to distinguish individuals and classes, the concept of actor can be primarily specialized into *agent* and *agent kind*.

An agent represents an individual, a particular instance which cannot have further instances, e.g., John Smith or company Acme. *Agent kind* is a generic classification of agents, which is further specialized into *agent type* and *abstract role*. An agent type is a necessary classification, i.e., it is a rigid classification [Guizzardi et al., 2004]. A type *T* is rigid iff every instance of *T* is necessarily (in the modal sense) an instance of *T*, i.e., if it cannot cease to be an instance of *T* without ceasing to exist.

An abstract role is an abstract class specialized into role and position. A role is an anti-rigid classification [Guizzardi et al., 2004]. A type *T* is anti-rigid when an instance of *T* is not essentially an instance of *T*, i.e., the entity can contingently be of type *T*, but not necessarily. In Tropos, a role characterizes the behavior of an agent within some specialized context or domain, i.e., while an agent is playing a role it has its behavior bounded to the behavior associated with the role. For example, suppose John Smith in a given time of his life plays the role of a teacher. While John Smith is playing the role of a teacher, he has to perform the behaviors associated with the teaching role, such as give lectures, apply exams, correct the exams and grade the students. The important fact here is that there are occasions in his life when he is not playing the role of teacher, and, therefore, not bounded to the behavior associated to this role.

A position represents an aggregation of two or more roles (or positions) played by an agent. For example, suppose that John Smith occupies a position of Head of Department, which includes the role of teacher and some administrative role. In the Tropos terminology, a position covers two or more roles (or positions), an agent occupies a position and, therefore, the agent plays the roles covered by the position. Figure 2-4 depicts the graphical representation of the Actor concept and its specializations.

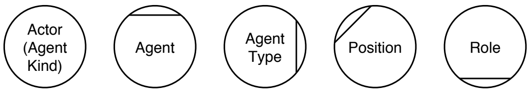


Figure 2-4 The graphical representation of the Actor concepts and its specializations

Tropos goal concept

In Tropos, a *goal* represents an intended state of affairs that an actor desires to be achieved. An actor will act accordingly to have the goal fulfilled. The goal represents actor's strategic interests and can be divided into hard goals and soft goals. The main difference between them is that a soft goal is typically a non-functional attribute or a quality, with no clear-cut criteria as to when it is achieved, while a hard goal precisely defines achievement criteria. Since it is not clear whether a soft goal is fulfilled, one can say that soft goals are *satisficed* while hard goals are satisfied [Chung et al., 1999].

Goals can be fulfilled by means of tasks, resources or a combination of both. A task represents a particular course of action that produces a desired effect. By performing a task, one can totally or partially satisfy a goal, or can totally or partially satisfice a soft goal. A resource represents a physical or an informational entity without intentionality, e.g., data, a file, a computer, amongst others. A resource can be consumed or produced by a task.

Tropos dependencies

Actors can be linked to each other through dependency relations. A dependency relation represents an agreement between two actors. An actor can depend on another actor to fulfill a goal, to perform a task or to deliver a resource. The former actor is named *depender*, while the later is named *dependee*. The object (goal, task or resource) around which the dependency lays is called *dependum*. Every dependency involves a speech act [Winograd and Flores, 1987], meaning that a dependency is an agreement upon a *dependum*. In the goal dependency, the speech act represents the agreement that the *depender* intends the goal (the *dependum*), but is not going to fulfill it by himself, the *dependee* adopts the goal and the *dependee* commits to procure a way to fulfill it.

Figure 2-5 presents an example of the Tropos goal dependency, where a patient (a role) depends on the hospital to fulfill his goal of receiving medical treatment.

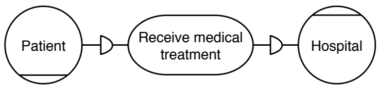


Figure 2-5 An example of the Tropos goal dependency

Regarding tasks, the speech act represents the agreement that the depender is unable to perform the task, the dependee is able to perform it, and the dependee commits with the depender to perform the task. Tropos does not make any assumption whether the depender has the capacity to fulfill a goal or to perform the task, only that it will not fulfill it or perform it for some reason. For example, it may be cheaper if the dependee does it, or it may not be convenient that the depender does it, among other reasons.

Finally, the speech act in the resource dependency represents the agreement that the dependee controls a given resource, the depender needs it and the dependee will provide access to the resource to the depender.

When one actor depends on another actor for a dependum, the depender is able of achieving goals, performing tasks, or using resources that otherwise would not be able to on its own. Furthermore, the depender becomes vulnerable because in the case of the dependee fails to deliver the dependum, the depender would be adversely affected. Figure 2-6 depicts the Tropos' dependency metamodel [Susi et al., 2005].

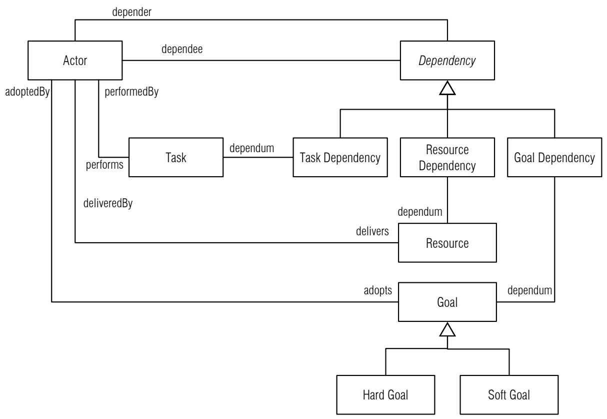


Figure 2-6
The Tropos dependency metamodel

Tropos development phases

Although it focuses on the early requirements phase, the Tropos methodology aims at covering the full range of the software development phases, namely Early Requirements, Late Requirements, Architectural Design, Detailed Design and Implementation [Bresciani et al., 2004]. The first two phases of requirements analysis share the same conceptual and methodological approach. In the early requirements phase, organizational analysis and modeling are performed . During this phase, the domain stakeholders are identified and modeled as social actors that depend on each other for goals to be achieved, tasks to be performed and resources to be made available. The objective of this phase is to understand the organizational context within which the system under development will eventually function. At the end of the early requirements, an organizational model is produced, including the relevant actors with their associated dependencies. The clear definition of these dependencies allows the assessment of the rationale for assigning some functionality to the system. This phase also enables the verification of whether the final implementation matches the initial goals.

In the late requirements phase, the conceptual model is extended. The system is represented as an actor and a number of dependencies with other actors of the environment are defined. These dependencies specify all the functional and non-functional requirements of the system under development, which are the main concerns of this phase.

The design phases, i.e., Architectural Design and Detailed Design, focus on the specification of the software system, and are based on the requirements gathered in the previous phases. The architectural design phase concerns with the system's global architecture, how it is subdivided in sub-systems and how the sub-systems are interconnect through data and control flows. Here, the sub-systems are represented as actors, and dependencies represent the data and control interconnections. This phase also provides a correlation between system actors and a set of software agents, each characterized by their specific capabilities. In short, the architectural design is composed of three steps: *(i)* introduction of new actors (the sub-systems), *(ii)* capability identification and *(iii)* the transformation of some actors into software agents.

The main concerns of the detailed design phase are specifying and detailing the agent's capabilities and interactions. Here, each architectural component is analyzed and the agents are specified in terms of their capabilities, goals, beliefs and tasks. Usually the implementation platform has been already chosen and the platform's particularities can be taken into account to facilitate the code in the next phase. In this phase existing agent communication languages such as FIPA-ACL [Labrou et al., 1999] or KQML [Finin et al., 1994] can be used.

Finally, the implementation phase follows the detailed design specification based on the mapping between the detailed design concepts and the implementation platform constructs.

Tropos modeling activities

The Tropos methodology involves several modeling activities during its phases. However, we discuss only the activities involved in the early requirements phase, as described below:

– *Actor modeling* consists of identifying and analyzing the actors of the environment, and the system's actors and agents. In the early requirements phase, actor modeling focuses on modeling the application domain stakeholders and their intentions as social actors which want to achieve goals. During late requirements phase, actor modeling focuses on the definition of the system under development as an actor, while during the architectural design phase it focuses on the structure of the system under development as an actor, by specifying it in terms of sub-systems (also modeled as actors) interconnected through data and control flows. In the detailed design, the system's agents are defined and all the notions required by the target implementation platform are specified accordingly. Finally, during implementation, actor modeling corresponds to the actual agent coding.

- *Dependency modeling* consists of identifying the goal, task and resource dependencies among actors. During the early requirements phase, dependency modeling focuses on modeling the goal dependencies among social actors of the organizational setting. In the late requirements phase, the focus is on the dependencies raised by the introduction of the system under development. In architectural design, the data and control flows identified during actor modeling activity are modeled in terms of dependencies among the sub-actors of the system under development, providing the basis for the capability modeling that starts in the detailed design together with the mapping of system actors to agents.
- *Goal modeling* consists of the analysis of an actor's goals, conducted from the point of view of the actor, by using three basic reasoning techniques: means-end analysis, contribution analysis, and AND/OR decomposition. In particular, means-end analysis aims at identifying plans, resources and soft goals that provide means for achieving a (hard) goal. Contribution analysis identifies goals that can contribute positively or negatively to the fulfillment of the goal to be analyzed. AND/OR decomposition consists of decomposing a root goal into subgoals using AND/OR decomposition links in order to produce a finer goal structure. Goal modeling is applied to early and late requirement models, aiming to refine them and to elicit new dependencies. During architectural design, it contributes to motivate the first decomposition of the actors of the system under development into a set of sub-actors.
- *Task modeling* consists of an analysis technique complementary to goal modeling. It applies reasoning techniques analogous to those used in goal modeling, namely, means-end, contribution analysis and AND/OR decomposition.

2.4.3 Goal-Based Requirements Analysis Method

The Goal-Based Requirements Analysis Method (GBRAM) [Anton, 1996, 1997] is focused on the initial identification of goals from various sources of information. It assumes that no goals have been documented or elicited from stakeholders and thus one uses existing diagrams, textual statements, interview transcripts, etc. for goal elicitation. In this approach, the concept of goal is defined as *"high level objectives of the business, organization or system"* [Anton, 1996]. A goal captures the reasons for needing the system, and guide decisions across enterprises. Goals can be classified into achievement and maintenance goals.

GBRAM prescribes the following activities: goal analysis and goal refinement. The goal analysis activity is further divided into the exploration, identification and organization sub-activities. The exploration sub-activity carries out the examination of the available input information (diagrams, statements, transcripts, etc.). During the exploration of the input information, the analyst identifies goals and their responsible agents from this information, and organizes these goals, by classifying these goals and grouping them according to goal dependency relations.

The goal refinement activity concerns the evolution of goals from the moment they are first identified, to the moment they are translated into operational requirements for the system specification. This activity is further divided into the refine, elaborate and operationalize sub-activities. In the refine sub-activity, the analyst prunes the goal set by eliminating redundant goals and reconciling synonymous goals. The refined goal set obtained in the refine sub-activity is then analyzed, by considering possible goal obstacles and constructing scenarios to uncover hidden goals and requirements. Goal obstacles are identified by analyzing statements that illustrate an example of a goal being blocked by another goal or conditions which prevent its completion [Anton, 1997]. Finally, in the operationalization sub-activity goals are translated into operational requirements, which specify how a goal can be fulfilled by a proposed system.

During the goal refinement phase, GBRAM requires the identification of goal precedence. This means that it is necessary to identify the order in which goals must be fulfilled. The method suggests answering questions like "What goal(s) must follow this goal?" and so on. Another useful method for determining precedence relations between goals is to search for agent dependencies. For instance, if a physician depends on a nurse to administer medicine to a patient in order to provide a proper medical treatment, there is an agent dependency, and a goal precedence relation can also be identified. Once goal precedence has been established, tables are produced with goals ordered according to this precedence.

Similar to other goal-based approaches, a system and its environment in GBRAM are represented as a collection of agents. Here, agents are defined as entities or processes that seek to fulfill goals within an organization or system, based on their assumed responsibility for the goals.

The GBRAM approach provides guidelines for goal elicitation and refinement by providing requirements engineers with standard questions. For instance, a question to determine if a goal can be classified as a maintenance goal is “Is continuous achievement of this goal required?”.

2.4.4 ArchiMate and its Motivation Extension

Enterprise Architecture (EA) is a set of principles, methods, and models that are used in the design and realisation of an enterprise’s organizational structure [Lankhorst, 2009]. Enterprise Architecture aims at clarifying the relationships between products, processes, organizations, information services and technological infrastructure. To accomplish this, EA relies on guidelines, assumptions, principles and preferences to produce a set of simplified representations of the organization, from different viewpoints.

Enterprise Architecture modeling techniques commonly focus on modeling architectures that have been or are being implemented in terms of information, behavioral and structural model elements. These model elements convey the information about *what* the enterprise does or should do. However, equally important information is the explicit representation of the rationale, or the *why*, the architecture has been defined in some specific way. Principles and techniques of Requirements Engineering can be applied to provide the rationale for Enterprise Architecture.

ArchiMate layers and aspects

ArchiMate [The Open Group, 2009a] is an enterprise architecture modeling language fostered by The Open Group. ArchiMate complements The Open Group Architecture Framework (TOGAF) [The Open Group, 2009b] by defining a modeling language for enterprise architecture. However, in the current version 1.0, ArchiMate does not provide language primitives to support requirements modeling. The Open Group Architecture Framework (TOGAF) [The Open Group, 2009b] acknowledges the need for explicit architecture rationale representation, and prescribes the use of goals and requirements as central drivers for the architecture development process. The TOGAF’s Architecture Development Method (ADM) applies requirements management to all phases of the ADM cycle, as depicted in figure 2-7.

The ArchiMate enterprise modeling language is underlined by a modeling framework that decomposes an enterprise along two dimensions: *(i) layers*, representing successive abstraction levels at which an enterprise is modeled, and *(ii) aspects*, representing different concerns of the enterprise that should be modeled. Fig-

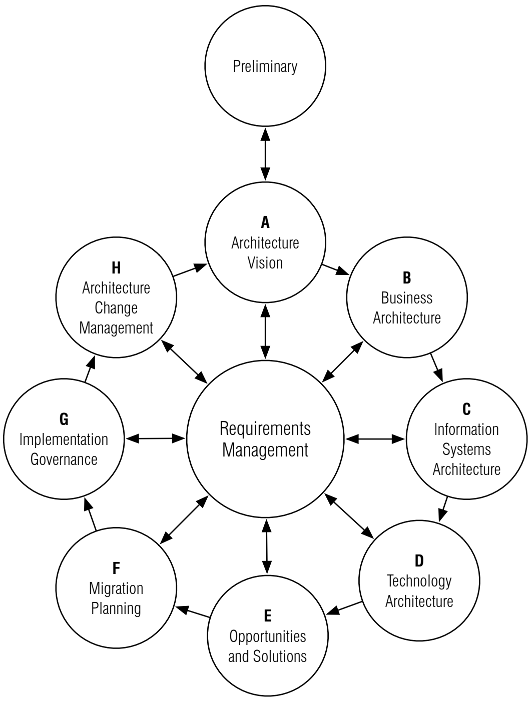


Figure 2-7 TOGAF Architecture Development Method

ure 2-8 depicts the ArchiMate's modeling framework [Engelsman et al., 2011, Quartel et al., 2009].

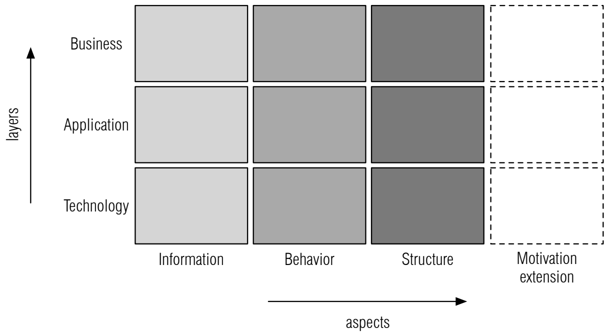


Figure 2-8 ArchiMate modeling framework

The layer dimension consists of the following layers:

- *business* layer, containing products and services to external customers that are realized in an organization by business processes;
- *application* layer, containing application services supporting the business layer that are realized by (software) application components;
- *technology* layer, containing infrastructural services necessary

to run applications that are realized by computer and communication devices and system software.

The aspect dimension comprises the following modeling aspects:

- *structure* aspect, representing the actors involved and how they are related. Actors can be systems, components, people, departments, etc.;
- *behavior* aspect, representing the behavior (processes and services) presented by the actors and how the actors interact;
- *information* aspect, representing the problem domain knowledge used by and communicated between the actors through their behaviors.

Motivation extension

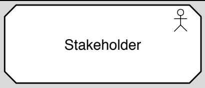
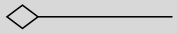
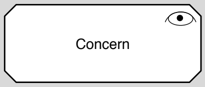
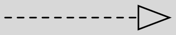
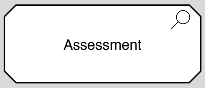
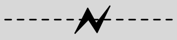
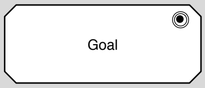
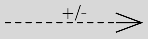
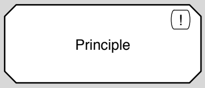
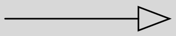
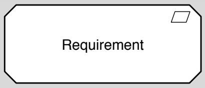
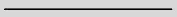
The ArchiMate 1.0 specification [The Open Group, 2009a] focuses on modeling the extensional aspects of the enterprise, i.e., its appearance as an operational entity from an external perspective. To allow the modeling of the intentional aspects of an enterprise, i.e., the business goals, principles and requirements that motivate the design of the enterprise, an extension for ArchiMate, named Motivation Extension, has been proposed [Engelsman et al., 2011, Quartel et al., 2009]. The ArchiMate Motivation Extension supports the modeling of motivational properties by extending the ArchiMate 1.0 framework with the *motivation* aspect.

In the motivation aspect, the intentions of the enterprise are defined in terms of goals, principles and requirements. Intentions are pursued by the stakeholders, having certain areas of interest called *concerns*. These concerns organize the stakeholders' intentions, and *assessments* of the concerns are used to decide whether the intentions need to be adjusted or not. Table 2-2 presents the modeling elements of the Archimate Motivation Extension.

Following the TOGAF specification, the *stakeholder* concept represents an individual, team or organization interested in the outcome of the architecture. The *concern* concept represents a key interest that is important to certain stakeholders, and that determines the acceptability of the system. The analysis of a concern produces as outcome an *assessment*, revealing the strengths, weaknesses, opportunities and threats (SWOT) that affect the enterprise architecture, which are addressed by new or modified business goals.

The *goal* concept represents an *end* intended by a stakeholder, which can represent different concepts such as an effect, a state of the problem domain, a produced value, tasks or a realized system

Table 2-2 The modeling elements of Archi-Mate's Motivation Extension

Concept	Notation	Relation	Notation
Stakeholder		Aggregation	
Concern		Realization	
Assessment		Conflict	
Goal		Contribution	
Principle		Specialization	
Requirement		Association	

property [Engelsman et al., 2011]. The goals *justify* the enterprise architecture choices, by providing the motivations for the choices for the structure and functionality of the architecture. In other words, the enterprise architecture has been defined in some specific way so that it fulfills the business goals.

The *requirement* concept represents "a desired property that must be realized by a system". In this definition, the term system is used with a large scope and may refer to different elements of a enterprise architecture, including information systems, data objects and business actors.

Finally, the concept of *principle* represents a generally desired property that is used to guide the design of systems. Principles are related to goals and requirements. Principles are motivated by goals, i.e., the stakeholder intends some state of the world (the goal), and the means to achieve this state are constrained by the principles. For instance, with the goal of increasing the productivity of its employees, an organization defines as a guiding principle that the software applications used by these employees should by easy to use. This principle is based on the assumption that easy-to-use applications allow the employees to perform better.

The concept of principle is more abstract and broader in scope than the concept of requirement. A principle defines desired properties that applies to any system in a given context, while a requirement usually applies to a specific system or a group of systems. To enforce the conformity of a system to a principle, it is necessary to specialize the principle in terms of a requirement to be applicable to the given system.

Regarding relations, besides the association, specialization, aggregation and realization relations that have similar meaning as their UML counterparts, the ArchiMate Motivation Extension defines the conflict and contribution relations.

The association relation is used to relate stakeholders to concerns and concerns to assessments. The aggregation relation allow the modeling of intention decomposition, i.e., goals, requirements or principle can be decomposed into more fine-grained intentions [Quartel et al., 2010].

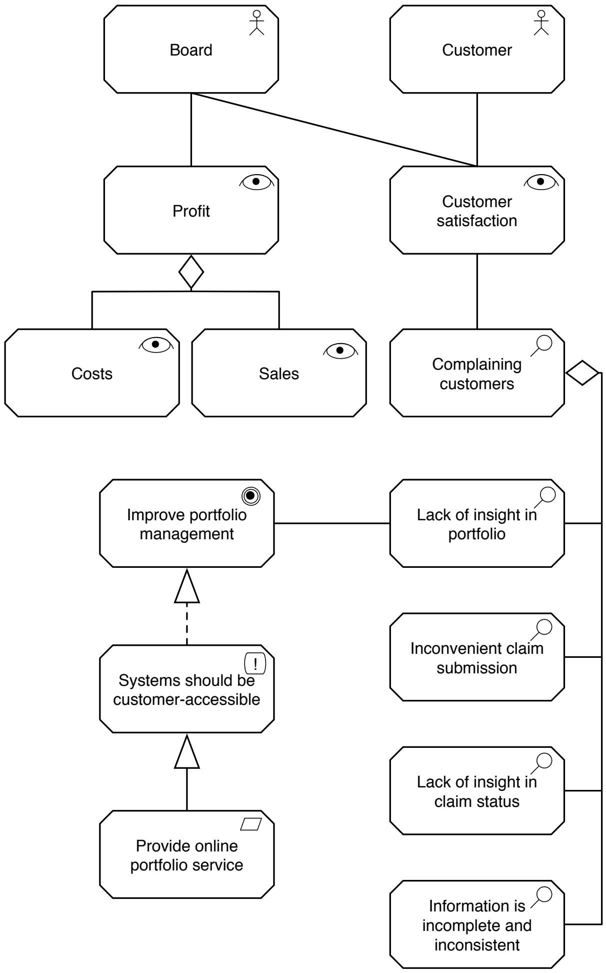
The realization relation allows the modeling that an *end* is realized by some *means*. It is used to denote that a goal is realized by a principle or a requirement, or that a requirement is realized by a system. This system can be represented by an ArchiMate's information, behavior or structure element, such as a business actor, application component, business process or application service [Quartel et al., 2010].

The conflict relation allows the modeling that the realization of two intentions are mutually exclusive. For instance, an airline corporation could model that the goal of increasing seat availability is in conflict with the goal of reducing the number of operated airplanes (assuming a fixed number of seats per airplane).

The contribution relation allows the modeling that the realization of an intention causes a positive or negative contribution to the realization of another intention. For instance, in the airline example, the goal of optimizing flight operations may contribute positively to the goal of increasing seat availability, while reducing in flight entertainment options may negatively contribute to the goal of increasing customer satisfaction.

Figure 2-9 presents an example of model created with the Archimate Motivation Extension. This model has been adapted from the Archimate Motivation Extension whitepaper [Quartel et al., 2010] and depicts an excerpt of the stakeholder's view of an insurance company enterprise architecture. Figure 2-9 shows two stakeholders, namely the *Board* and the *Customer*. Each stakeholder is associated with the concerns *Profit* (which comprises the Costs and Sales concerns) and *Customer satisfaction*, respectively, and the concern *Customer satisfaction* is shared between these two stakeholders. We assume that the analysis of the *Customer satisfaction* concern revealed a degree of customer dissatisfaction and lead to the assessment *Complaining customers*. This assessment can be decomposed into a set of other assessments as depicted in Figure 2-9.

Figure 2-9 An example model using ArchiMate Motivation Extension


Board
Customer
Profit
Customer satisfaction
Costs
Sales
Complaining customers
Improve portfolio management
Lack of insight in portfolio
Inconvenient claim submission
Systems should be customer-accessible
Lack of insight in claim status
Provide online portfolio service
Information is incomplete and inconsistent

Among the (sub-)assessments, the *Lack of insight in portfolio* assessment can be addressed by the goal *Improve portfolio management.* The principle *Systems should be customer-accessible* realizes the goal of improving the portfolio management and is specialized into the requirement *Provide online portfolio service.*

2.5 Service-Oriented Computing

The area of Service-Oriented Computing (SOC) has also been using the concept of goal, particularly to formulate users' requirements to services, and to guide the service composition process. For instance, in [Kaabi et al., 2004], both these objectives are pursued by using a goal-driven approach to elicit functional requirements for inter-organizational processes and to identify which services should be provided by each organization. This approach uses a labeled directed graph with intentions as nodes and strategies as edges between intentions. Strategies are defined as ways to achieve intentions. An edge entering a node defines that the given strategy (the edge) can be used to achieve the corresponding intention (the node). Multiples edges entering a node represent alternative strategies that can be used to achieve an intention. In [Kaabi et al., 2004], *Intention* is defined as a *"goal to be achieved by performing a process"*, and a process consists of a sequence of intentions and strategies that should be followed to achieve a particular intention.

A benefit of adopting the concept of goal in Service-Oriented Computing to formulate user's requirements is the possibility of specifying the user's objectives at the knowledge level without having to deal with their technical solutions. For instance, instead of specifying the service requirements in terms of a WSDL document specifying the intended service, a user could express his goal, i.e., what he expects to reach as a result of the service execution.

To identify and characterize the use of the goal concept in the area of SOC, we discuss below a set of approaches that make use of this concept, namely, the Web Service Modeling Ontology (WSMO), GoalMorph, the goal and task-based approach for service composition presented in [Zhang et al., 2006] and the SM4All project.

2.5.1 WSMO

The Web Service Modeling Ontology (WSMO) [De Bruijn et al., 2005] is an ontology-based conceptual model for describing Semantic web services. WSMO aims at describing all relevant aspects related to services in order to enable (total or partial) automation of the service-provisioning tasks, such as service discovery, selection, composition, mediation, execution and monitoring [Roman et al., 2005].

WSMO is founded on the Web Service Modeling Framework (WSMF) [Fensel and Bussler, 2002] and refines and extends WSMF by means of a formal ontology and a family of languages. Being an initiative in the scope of Semantic Web Services, WSMO aims at integrating principles from the World Wide Web, the Semantic

Web [Sem, 2011], as well as design principles for Service-Oriented Computing. These integrated design principles are:

- *Web compliance*: WSMO inherits from the World Wide Web the concept of URI (Universal Resource Identifier) as the unique identification of resources. Moreover, WSMO applies namespaces for denoting consistent information spaces, supports XML and other W3C Web technology recommendations, as well as the decentralization of resources inherent to the Internet.
- *Ontology-base*: ontologies are used as data models in WSMO, i.e., all resource descriptions and all data interchanged during service usage are based on ontologies. The usage of ontologies in WSMO allows semantically-enhanced information processing as well as interoperability. WSMO also supports the ontology languages defined for the Semantic Web, such as RDF and OWL.
- *Strict decoupling*: in WSMO, resources are defined in isolation, i.e., each resource is specified independently without regard to possible usage or interactions with other resources. This complies with the open and distributed nature of the Web.
- *Central mediation*: mediation addresses the heterogeneity that naturally arise in open environments as a complement to strict decoupling. Heterogeneity can occur in terms of data, underlying ontology, protocol or process. WSMO tackle heterogeneity by making mediators first class components of the framework.
- *Ontological role separation*: users exist in specific contexts which may not completely coincide with the context of the available Web services. For instance, a service user may wish to book a holiday according to preferences for weather, culture and childcare, whereas available travel Web services may only cover airline travel and hotel availability. The underlying epistemology of WSMO differentiates between the desires of users and available services.
- *Description versus implementation*: WSMO differentiates between the descriptions of Semantic Web Services elements (description) and executable technologies (implementation). While descriptions require a concise and sound description framework, implementations are concerned with the support of existing and emerging execution technologies for the Semantic Web and Web services.
- *Execution semantics*: reference implementations such as WSMX [Bussler et al., 2005], should allow verification of the WSMO specification and provide formal execution semantics.
- *Service versus Web service*: WSMO differentiates between a

service and a Web service. The Web service is a computational entity which is able to achieve a user's goal when invoked, while a service is the actual value provided by this invocation. WSMO provides means to describe Web services that provide access to services.

Following the WSMF guidelines, WSMO prescribes four main elements to describe semantic web services, namely ontologies, goals, web services and mediators. Ontologies provide (domain-specific) terminology for the other elements as well machine-processable formal definitions of terms, Web services represent computational entities capable of providing access to services, goals define intentions of the service users that should be solved by web services, and mediators define elements that resolve interoperability problems between other components.

The WSMO conceptual model is supported by a family of languages called the Web Service Modelling Language (WSML) [de Bruijn et al., 2005]. This family of languages consists of WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full, which offer different levels of expressiveness, varying from the minimal level supported by WSML-Core (intersection of Description Logics and Logic Programs [Grosof et al., 2003]) to the maximum level expected to be supported by the WSML-Full (First-Order Logic combined with Logic Programs).

In WSMO, a goal is defined in [Roman et al., 2005] as a description of the service user's desires and in [Domingue et al., 2005] as specifying *"the objectives that a client may have when consulting a Web service, and describing aspects related to user desires with respect to the requested functionality"*. A goal in WSMO is described by non-functional properties, imported ontologies, used mediators, requested capabilities and a requested interface as shown below (in a MOF notation).

```
Class goal
      hasNonFunctionalProperties
            type nonFunctionalProperties
      importsOntology type ontology
      usesMediator
            type {ooMediator, ggMediator}
      requestsCapability
            type capability multiplicity = ↩
               ↪ single-valued
      requestsInterface
            type interface
```

In WSMO, goals and Web services descriptions have similar structures, which facilitates service discovery and matching. In the goal description, the capability section is used to express what the requester would like to achieve, and the interface describes how the requester would like to interact with a web service. Goal descriptions can be viewed as the counterparts of Web service descriptions, in the sense that a goal description states what the service requester wants to get achieved, and a Web service description specifies what a Web service can achieve.

2.5.2 GoalMorph

GoalMorph [Vukovic and Robinson, 2005b] is a context-aware goal transformation framework. The framework uses planning techniques and combines goals and contextual information to derive service compositions. GoalMorph transforms goals into problem that can be solved by the planner. A service composition is obtained in GoalMorph by using goals to represent information about the planning problem (or composition request), and to limit the inference space in the planning process. Reasoning on goals provides criteria for creating a successful plan.

The main contributions of GoalMorph are *(i)* a model that represents context-aware goals; *(ii)* analysis and taxonomy of goal types and corresponding transformations; *(iii)* a model that allows reasoning about partial satisfaction of goals; and, *(iv)* a utility model to reason about goal transformations and the corresponding partial success in achieving one goal against the partial success in achieving another goal.

Goal taxonomy

GoalMorph classifies goal into core and context goals. Core goals arise from a user's request, while context goals are side-effects of the current user's context. The following classes of goals have been identified based on core and context goals:

– *Core goal*: a goal that purely describes the user's task intention. A task intention describes which tasks the user intends to be performed.
– *Base core goal*: the absolute minimal core goal that needs to be satisfied to achieve a solution. Other core goals that are not base core goals can be suppressed from a plan to achieve a partial goal satisfaction.
– *Dependent context goal*: a context goal that can be seen as an attribute of a core goal or is directly related to it. If the core goal is removed from the goal set, the related dependent context goals are also removed.

– *Independent context goal*: a context goal that does not directly affect the user's request and, therefore, is not removed in case a core goal is removed.

An example of this goal taxonomy is a scenario where a service user named Miles has access to a set of services provided by his mobile network provider, and would like to locate and receive the directions to a Spanish restaurant. Furthermore, he wants to have these directions delivered in speech while he is driving to the restaurant.

In this example, Miles' goals have been defined as *(restaurant_found spanish amsterdam)*, *(directions_found current_ address restaurant_ address)* and *(directions speech_ out)*. The goals *(restaurant_found)* and *(directions_found)* can be classified as core goals, since they express the essence of Miles' intention. The goal *(restaurant_found)* can also be classified as a base core goal, because even if he does not get the directions for the restaurant, having the address enables him, for instance, to take a taxi and ask to be taken to that address instead of driving himself. The goal *(directions speech_ out)* can be classified as a dependent context goal, because it relies on the presence of the directions obtained from the core goal *(directions_found)*. Moreover, it would be useful for Miles to add an additional goal condition that lowers the car-audio system while reading out the route directions. Since this added goal does not directly affect the service user's request, it can be classified as an independent context goal.

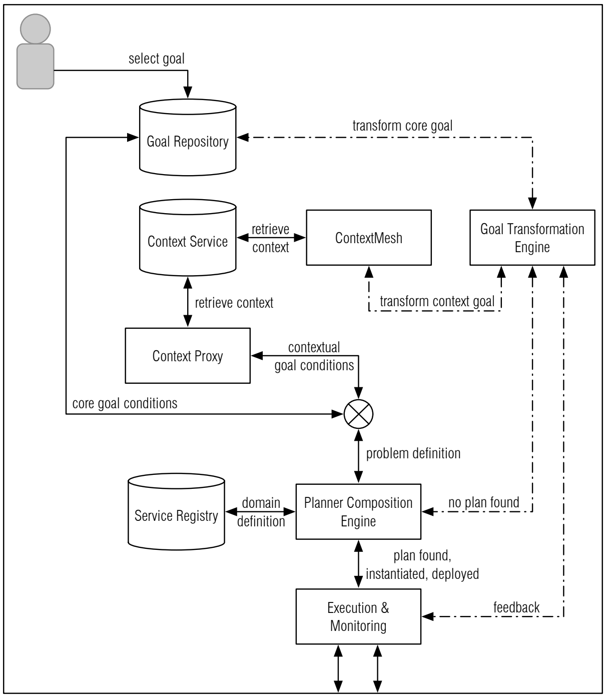
Architectural design

The starting point of GoalMorph is the user's request for a composite service. A user selects a goal from the GoalRepository and this goal forms the basis to obtain a composite service. The GoalRepository stores the available core goal templates that have been defined by domain engineers using the Planning Domain Description Language (PDDL) [Ghallab et al., 1998], which is a planning language to represent goals.

In the GoalMorph's architecture depicted in Figure 2-10, *ContextService* is a general middleware infrastructure for context brokering from a number of different context sources to context consumers. The context source used in GoalMorph to retrieve contextual information is based on the solution proposed in [Lei et al., 2002]. The *ContextProxy* component generates context goal conditions that constrain the composition request based on contextual information of the user provided by the *ContextService*.

Based on the core goal conditions from the *GoalRepository* and

Figure 2-10 GoalMorph's architectural design



the context goal conditions from the *ContextProxy*, the final composition request is assembled. The *Planner Composition Engine* then receives the problem definition (the final composition request) and the domain definition from the *ServiceRegistry*, and uses knowledge about available actions and their consequences to identify a solution, i.e., to create a plan. The creation of the plan fails if the goal cannot be satisfied or if the domain knowledge is incomplete.

If the *Planner Composition Engine* fails to create a plan for the problem definition, control is transferred to the *Goal Transformation Engine*, which transforms the goal into a problem solvable by the *Planner Composition Engine*. The transformation is performed on the core goal(s) and, by interacting with the *GoalRepository*, the *Goal Transformation Engine* attempts to find forms of the original goal that can be satisfied. An ontology consisting of a number of hierarchies for the goals stored in the *GoalRepository* is used to support goal transformation.

When a transformation ends successfully, the transformed goal is passed to the *ContextMesh* for context layering, i.e., for refining the context goal conditions by context unfolding or relaxing the context goal conditions by context folding. Users provide context importance measures through a GUI, and these measures are used as guidelines for the context layering operations. The final step is to feed the transformed goal to the *Planner Composition Engine*, and the resulting composition is evaluated by the user to refine the goal transformation utilities.

2.5.3 A Goal and Task-based Approach for Service Composition

Another approach to goal-based service composition is presented in [Zhang et al., 2006]. This approach starts by building a task-oriented semantic representation model of web services. In this model, application scenarios and task goals are defined at a higher abstraction level. Dynamic service discovery and matching, and goal-based composition are performed to achieve the user's goals. A Task Definition Language (TDL) has been specified. In TDL tasks can be decomposed into sub-tasks, and tasks are defined to achieve goals. An ontology is used to provide an uniform vocabulary for the description of concepts and tasks. A goal consists of activities, and each activity can be defined more specifically (decomposed). A basic activity can be matched directly to a concrete service and does not need to be further specified.

The approach divides the service composition life-cycle into three phases: definition, construction and execution. The definition phase allows designers to define goals and abstract tasks to achieve the goals. The difference between the abstract definition and concrete service composition is that service providers and control information are not specified in the abstract definition. In the construction phase, the task is decomposed, concrete services are matched and bound, and an executable service process is generated. In the execution phase, the concrete Web services are invoked and the process is executed to achieve the user's goal.

2.5.4 Smart Home for All

The Smart Home for All (SM4All) is an European project aiming at investigating *"middleware platforms for inter-working of smart embedded services in immersive and person-centric environments, through the use of composability and semantic techniques for dynamic service reconfiguration" [SM4, 2008-2011].* The project is being developed in the scope of private houses and home-care assistance in presence of users with different abilities and needs.

The SM4All project considers scenarios where users are offered a number of services that are selected depending on the user's goals. Some of these services are completely automated by means of sensors, appliances and actuators, while other services are realized through the collaboration of human agents [Catarci et al., 2009].

Software platform

In the SM4All project, a software platform has been designed to support the integration of heterogeneous home devices, the inference of the home context, and service composition as a response to a user's request or a home event [Kaldeli et al., 2010].

The software platform design has been driven to the solution of handling domestic events. These domestic events can be generated by a user's goal or by some situation in the house that needs to be handled, and the software platform addresses these events by creating service compositions specific to the event and the current house context.

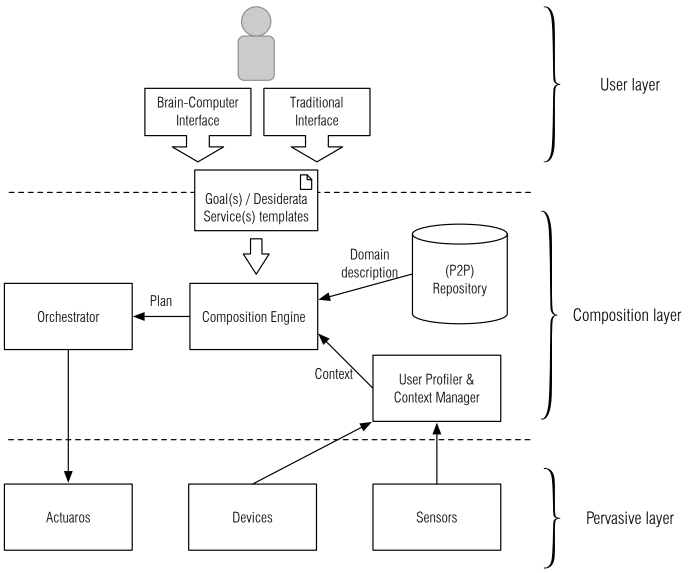
The software platform architecture is composed of three layers, namely the *user layer*, the *composition layer* and the *pervasive layer*, as depicted in Figure 2-11. The user layer provides the interface between the users and the house. Besides the regular screen-based interfaces, SM4All also provides a Brain-Computer Interface (BCI) [Guger et al., 2009]. The composition layer is responsible for inferring the state of the home, by receiving information from the components of the pervasive layer and coordinating these components on behalf of the user. Finally, the pervasive layer consists of heterogeneous sensors, actuators and devices.

The central component of this software platform architecture is the *Composition Engine*. Its task is to compute a plan (i.e., a sequence of actions) that needs to be performed to satisfy a given user goal. The *Composition Engine* first retrieves the home description from the *Repository* and generates the planning domain, by mapping each UPnP action (from the elements of the pervasive layer) to a planning-level action. After that, the generation of the planning domain only needs to be repeated when a new service is discovered, or when an existing service is removed.

The planning-level actions are specified in terms of preconditions and effects [Kaldeli et al., 2010]. For instance, the planning-level action *turn-on-ventilator* is defined as:

```
turn-on-ventilator
        Preconditions:
                ventilator := OFF
        Effects:
                ventilator := ON
```

Figure 2-11
The SM4All software platform architecture



The *Composition Engine* is continuously informed about changes in the house's context, and once new contextual information is received, the state of the house is updated. Whenever a user selects one goal to be satisfied, the *Composition Engine* searches for the services that can be composed, and, as such, can transform the house from its current state to the state represented by the goal. When a composition is found, the *Composition Engine* generates the plan and submits this plan to the *Orchestrator*.

The *Orchestrator* maps the planning-level actions back to the concrete UPnP actions to be executed in the pervasive level. For instance, if the *turn-on-ventilator* planning-level action is used in a plan, the *Orchestrator* maps this action to the appropriate UPnP action from the actuator that switches on the ventilator.

The SM4All approach offers a loose-coupled strategy to encode actions by means of planning-level and UPnP actions. In this way, the planning-level actions can be common to all deployments in a given application domain, needing only to establish the mappings between planning-level actions and the UPnP actions that are particular to a certain deployment. For instance, planning-level actions such as *increase-heating*, *close-garage-door* and *turn-on-lights* are common in the domotics domain and the deployment of the system in a particular house needs to map these common actions to the UPnP actions of the actuators in the house that actually perform the actions.

Goal definition

SM4All uses a goal language to allow the specification of goals by means of numeric variables, temporal constructs and maintainability properties. The goal is expressed in a declarative way, i.e. it specifies what properties should be satisfied for the goal to be fulfilled, and under which conditions. The syntax of a goal specification is defined as [Kaldeli, 2009]:

$$
\begin{aligned}
\textit{goal} &::= \wedge_i(\textit{condition-goal}_i \mid \textit{subgoal}_i)\\
\textit{condition-goal} &::= (\textit{subgoal})\textit{under\_condition}\\
\textit{subgoal} &::= \textit{achieve-maint}(\wedge_i \textit{prop}_i) \mid\\
&\quad\ \textit{achieve}(\wedge_i \textit{prop}_i) \mid\\
&\quad\ \textit{find\_out-maint}(\wedge_i \textit{k-prop}_i) \mid\\
&\quad\ \textit{find\_out}(\wedge_i \textit{k-prop}_i)\\
\textit{prop} &::= \textit{var} \odot \textit{value} \mid \textit{var}1 \odot \textit{var}2 \mid\\
&\quad\ (\textit{var}1 \diamond \textit{var}2) \odot \textit{value}\\
\textit{k-prop} &::= \textit{k-var} \odot \textit{value} \mid \textit{k-var}1 \odot \textit{k-var}2\\
&\quad\ \mid (\textit{k-var}1 \diamond \textit{k-var}2) \odot \textit{value} \mid\\
&\quad\ \textit{k-var\_known} = \textit{true}
\end{aligned}
\tag{2.2}
$$

In the listing 2.2, $var \in (Var \setminus Par)$, where Var is a set of variables and Par is another set of variables that play the role of input parameters. *k-var* stands for the knowledge variables and *value* represents some boolean or numeric constant, depending on the type of the related variable [Kaldeli et al., 2009]. $\odot$ represents relational operators ($\odot \in \{<, >, \neq, \leq, \geq, =\}$) and $\diamond$ represents a numeric operator ($\diamond \in \{+, -\}$. The knowledge proposition $k-prop$ refers only to knowledge variables. Knowledge propositions and knowledge variables are related to sensing actions, i.e., actions that are responsible to gather knowledge by means of the context-aware components.

The $achieve(\wedge_i prop_i)$ subgoal implies that the Composition Engine can try any action having the potential to contribute to the propositions' satisfaction. The $find\_out(\wedge_i k\text{-}prop_i)$ means that a state that satisfies $\wedge_i k\text{-}prop_i$ has to be reached without using any action that include effects that change the state of the world on the variables in $\wedge_i k\text{-}prop_i$. For instance, the goal $find\_out(account\_balance > 100)$ is satisfied if the sensed value for *account_balance* is greater than 100 without allowing any action to alter the variable's value before the sensing action. Contrarily, with the goal $achieve(account\_balance > 100)$, the Composition Engine may invoke the action $transfer\_in$ that transfer a given amount to the account, increasing its balance.

The *maint* (maintenance) suffix used in the syntax presented in listing 2.2 adds the requirement that once the related proposition becomes true at some state, it should remain true in all subsequent states. The *under_condition* primitive imposes that some conditions should be assured before the fulfillment of the subsequent subgoal.

An example of goal defined using the SM4All's goal language is the following (from [Kaldeli et al., 2009]:

achieve-maint ($bookedHotel \wedge bookedFlight$)
under_condition
(find_out-maint($hotelPrice + flightPrice \leq 400$)

In this example, a user wants to book an air ticket and make a hotel reservation. However, the condition imposed by this goal is that the price of the air ticket and the hotel room cannot exceed 400 euros.

The goal concept is defined as the conjunction of condition goals and subgoals, which can be reduced to its constituent propositions. Therefore we can conclude that in SM4All, a goal is considered as a set of propositions that represent a state of affairs intended to be achieved.

2.6 Conclusions

The concept of goal has been used in several areas of Computer Science. Since targeted to different stakeholders, with different objectives, the approaches do not share a common conceptualization and definition. However, it is possible to identify some trends among the several approaches discussed in this chapter. One of these trends is the definition of goal as a state of affairs that is intended by the user. However, in some cases the same approaches that define goal as an intended state of affairs use another concept to explicitly refer to the state of the world. This is the case, for instance, for the approaches in the scope of Artificial Intelligence, as well as in BDI logics (assuming that the concept of desire can be considered as a goal), Intention Logic and KAOS. By having two (or more) concepts referring to the state of the world, it is not clear whether the goal is a special type of of state of affairs, or if it is some kind of relation between an agent, an intention and a state of affairs. In the scope of our work, we believe that this distinction is important and our foundational ontology should clarify the relations between goal, state of affairs, intention and agent.

In some other approaches, we have found that their definition for the goal concept is not precise and, in some cases, inconsistent. For instance, in the seminal paper of KAOS [Dardenne et al., 1993], the term goal is frequently used interchangeably with the term objective. Moreover, the goal definition includes the term objective, making the definition self-referenced. In [Zhang et al., 2006], it is unclear how a goal should be defined, and the terms goal and task are used interchangeably. The approach presented in [Kaabi et al., 2004] considers that goals and intentions as equivalent concepts. In these two approaches, a designer may not be sure of which concept to use and under which situations one concept should be chosen and not the other.

In the ArchiMate Motivation Extension there are several different concepts collapsing onto the concept of goal. For instance, a goal represents an *end*, and this *end* can represent different concepts such as an effect, a state, a produced value, a task or a realized system property. Regarding the entities that are related to a goal, the difference between stakeholder and actor is not clear, and there is an overload on the realization relation because the relation can relate either a goal with a principle or a requirement, or a requirement with a core concept like, for instance, a system.

In WSMO, there is a clear issue on the goal concept being closely related to a service, so that one could consider that a WSMO goal description corresponds to an abstract service description.

After the analysis of these several definitions for the goal concept and how the concept is used in the related approaches, we can conclude that the conceptualization for goal is not a consensus. The lack of concept comes from the fact that the goal concept has been defined according to the objectives of each approach, which are different among themselves and most of them different also from the objectives of our work. However, we could get inspiration from these approaches on the conceptualization for goals. Moreover, the relation between goal, intention and state of affairs should be clarified as well as causality consequence of adopting a goal, i.e., when adopting a goal, what are the consequences in terms of commitments that are inherited by the adopter.

Chapter 3

Dynamic Service Provisioning Framework Architecture

In this chapter we present our framework to support dynamic service provisioning, by giving an overview of the framework's elements and justifying their purpose. This overview aims at positioning the framework's elements and facilitating the understanding of the framework as a whole and the role of each individual element. This chapter is structured as follows.

Section 3.1, presents and discussess use case scenarios. These use case scenarios characterize situations of service provisioning in different domains and were used to identify the stakeholder's roles presented in Section 3.2. The use case scenarios consist of situations in which service clients may need support to service provisioning activities such as service discovery, selection, composition and invocation. This support mediates the interactions between service clients and service providers. Section 3.3 discusses service mediation and motivates the software support to realize service mediation.

Based on the analysis of the use case scenarios and the discussion on service mediation, Section 3.4 presents a set of requirements for a software-based solution to dynamic service provisioning. These requirements motivated the development of our service provisioning framework. Section 3.5, presents the architectural design of our service provisioning framework, which starts from a software platform to mediate service provisioning, and has been incrementally enhanced with the other framework's elements that were needed to comply with the identified requirements.

3.1 Use Case Scenarios

The main objective of our framework is to support dynamic service provisioning in Pervasive Computing environments, i.e., environments surrounding users and populated by a multitude of services and computing devices. To identify classes of stakeholders involved in service provisioning and their interaction patterns, we have selected a set of use case scenarios. These use case scenarios have been selected from the ones identified, investigated and validated in the scope of the Amigo [Ami], A-Muse [A-M] and U-Care [U-C] projects.

In our work we have focused on the areas of Ambient Intelligence and Health Care. These areas offer application scenarios that are appealing and generally seen as useful. For instance, it is generally acceptable that a higher degree of automation (associated with ease of use) in home, working and health care scenarios may improve quality of life. Moreover, contrarily to specialized areas where one needs specific knowledge to understand the issues and scenarios of these areas, Ambient Intelligence and Health Care, for their wide-spread reach allow that more people can perceive the usefulness of their application scenarios. The chosen use case scenarios represent situations that can be realized with the currently available technologies, situations that should be possible with the technologies and techniques that are being researched or developed and are expected to be launched in the next few years.

The following use case scenarios have been used to identify the service provisioning stakeholders and the initial requirements for dynamic service provisioning:

Scenario #1: *Nomadic ambient comfort*

John is a professional who lives in a city apartment but also owns a beach house and a mountain cabin. He has ambient comfort preferences such as light intensity, light color, temperature and humidity that he expects to be automatically applied on all houses and working environments he uses. Moreover, everyday John receives several phone calls, e-mails, SMS and instant messages. However, in some situations some types of messages are more suitable than others. For instance, when he is in a meeting, voice messages should be transcribed and delivered in textual form on his smartphone, or when he is driving text messages should be delivered in an audible form through his car's audio system. Therefore, John would like that incoming messages are delivered according to his current activities. This use case was identified in the Amigo project [Ami].

Scenario #2 *Automatic adjustment of weekly menu based on diet requirements*

Anna lives in a smart home. Every morning, when she wakes up and washes her face in the bathroom, sensors on the floor in front of the sink weight her and measure the percentage of fat, water and muscles in her body. With this information, a weekly menu is set according to her needs for more or less calories and the availability of groceries in her house. If an adequate menu could not be established with the groceries in stock, a shopping list is compiled. The shopping list is either automatically sent to a supermarket that provides delivery services or uploaded to Anna's smartphone so she can shop herself. Additionally, more recipes should be gathered to vary the weekly menus based of Anna's eating preferences. This use case was identified in the Amigo project [Ami].

Scenario #3: *Receive emergency assistance for epilepsy*

Maria has a chronic epileptic condition. However, she wants to carry on with her life as normally as possible. As a daily routine, every morning she runs in the park near her house. Nowadays it is possible to detect an imminent epileptic seizure based on body signals. Therefore she wants to be warned if her body signals reach a critical point so she can stop running and try to put herself in a resting position. Concurrently, the relative or friend who is closest to her location and available to help should be warned of her potentially incoming seizure and head on to her whereabouts. The assigned or on-duty caregiver should also receive information about her body signals, her location, which relative or friend has been warned and when and how far he/she is from Maria's location. This information is used by the caregiver to decide to send an ambulance (depending on the severity of the body signals) or to contact the warned relative or friend to provide further assistance instructions and receive extra situation assessment. This use case was identified in the A-Muse project [A-M].

Scenario #4: *Control the use of medicine by home-bound elderly patients*

Peter and Sofia are an elderly couple living alone at their home. Peter suffers from high blood pressure and has to take some controlled medicine. His medicines have fixed and determined frequency and schedule to be taken. Sofia suffers from a mild diabetes that can be normally managed through a controlled diet and only in exceptional cases she needs to take insulin shots. Both of them are in the initial phase of senility, presenting occasional memory lapses. However, it is preferable to keep them at home.

For them it is advantageous to stay at home instead of in a health care facility because they prefer to stay as long as possible in their well-known environment. For their health insurance provider, keeping them at home is cheaper than transferring them to a nursing facility or to a hospital. Therefore, the health insurance provider hires a home health care specialized company to support the couple with their medical needs. To allow them to remain living in their house, the home health care company provides mechanisms to control their body signals such as his heart rate and blood pressure, and her glucose level. Her diet is controlled and, according to what she eats, the need for an insulin shot is evaluated by monitoring her glucose level. In case her diet can not be controlled, for instance, when she eats out, her glucose level is assessed using the glucose meter. Moreover, both of them are reminded of the time to take their medicines and only the correct amount is made available on their automatic medicine dispenser. This use case was identified in the U-Care project [U-C].

Scenario #5: *Nomadic medical consultation*

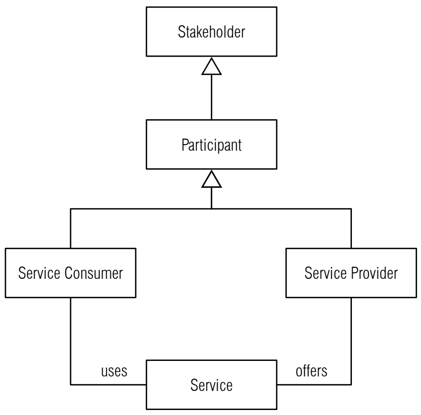
Lucia is a professional who is frequently required to go on business trips. In one of these trips, a health-related event occurs and she needs a medical consultation. However, since she is abroad and, consequently not near her general practitioner, she needs to find a nearby doctor so that the consultation and treatment costs are covered by her health insurance. During the consultation the doctor abroad should be able to assess her medical history. Moreover, during the consultation, the doctor abroad requested further medical tests and prescribed some medication. Again, Lucia needs to find laboratories that can perform the requested tests and pharmacies that provide the prescribed medicines, and are covered by her health insurance. We devised this scenario by combining elements of the former scenarios.

3.2 Service Stakeholders' Roles

Taking into account the above use case scenarios we have identified services to fulfill the needs of the presented subjects. Some of the identified services were directly extracted from the use case scenarios' descriptions, such as the home health care service implied in the use case scenario # 4. Other services were identified because the needs of the subjects presented in the scenarios could have been fulfilled through services (and have been fulfilled by services developed in the referred projects), such as the message adaptation services and ambient comfort adjustment services extracted from use case scenario #1. The stakeholders' roles that we have identified from the analysis of the services should be general in the sense that they are applicable not only for a particular service instance but also for other service provisioning instances.

In a simple service provisioning activity, two types of stakeholders interact. On one side there is a stakeholder that plays the role responsible for defining and offering services. On the other side another stakeholder plays the role responsible for requesting the delivery of a service. While the former is commonly called service provider, the later is often called service client. In the related literature, besides the terms service client and service provider [Papazoglou and Georgakopoulus, 2003] these two roles are also referenced with different terms, such as requester and provider entities [Booth et al., 2004], service requester and service provider [Burbeck, 2000], service consumer and service provider [Laskey et al., 2009], or service customer and service trustee [Ferrario and Guarino, 2008]. Regardless the used terms, the conveyed concept is one stakeholder offering (or providing) the service and another stakeholder requesting and using the service. Figure 3-1 presents an excerpt of OASIS' Service-Oriented Reference Architecture [Laskey et al., 2009] modeling the relations between Service Consumer, Service and Service Provider.

Figure 3-1 OASIS' simplified service participants



Following we list and describe the stakeholders' roles involved in service provisioning that we have identified from the use case scenarios presented in Section 3.1:

- *Service Client.* Responsible for requesting services. The Service Client also deals with possible negotiations over the service provisioning terms. For example, a frequent traveler can negotiate with an airline for discounts on a bulk purchase of tickets or a company can get a faster delivery of supplies after negotiating a transport service. With the service contract, commitments are established between the Service Client and the entity providing the service. These commitments define the Service Client's rights and obligations in the scope of the contracted service. Examples of Service Client's obligations are the payment of a specified amount for the service delivery or the availability of some information required for the service execution. Our definition of Service Client is similar to the concepts of service customer in [Ferrario and Guarino, 2008], service client in [Papazoglou and Georgakopoulus, 2003], service consumer in [Laskey et al., 2009] and service requester in [Booth et al., 2004, Burbeck, 2000].

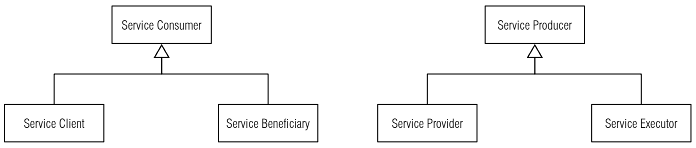
– *Service Beneficiary.* In our work we distinguish a Service Client, which requests and contracts a service, from a Service Beneficiary, which receives the benefit of the service delivery. The Service Beneficiary may or may not be the same entity as the Service Client for some service provisioning instances. For example, a parent contracts the education services of a school for his child while the direct beneficiary of the service is the child. In our use case scenario #4, although the health insurance company (in this instance acting as a service client) hires the home health care company, the direct beneficiaries of the service are Peter and Sofia. Entities can benefit from service delivery even if the Service Client is unaware of it. For instance, if one resident of a street contracts a service for cleaning his street from snow, all the inhabitants of this street benefit from the service. However, in this example the Service Client also directly benefits from the service delivery. The benefit perceived by his neighbors is more like a side-effect of the delivery of his contracted service. In other words, the householder that contracted the snow cleaning service aimed at having a personal benefit from the service although his neighbors also perceived some benefits. In our work, we categorize as Service Beneficiary the entities benefiting from a service as explicitly intended by the Service Client, i.e., the Service Client requests a service with the explicit intention of benefiting the Service Beneficiary.
– *Service Provider.* We define the Service Provider as the stakeholder accountable and liable for the service offering. The Service Provider advertises its offered services and commits with the performance of the activities described in the service advertisement once a service is contracted by the Service Client.

Our definition of Service Provider is similar to the concepts of service trustee in [Ferrario and Guarino, 2008] and service provider in [Booth et al., 2004, Burbeck, 2000, Laskey et al., 2009].

– *Service Executor*. Similarly to the differentiation between Service Client and Service Beneficiary, we also differentiate between Service Provider and Service Executor. The former is the actual responsible (and liable) entity for the service from the point of view of the Service Client. The later is the stakeholder that actually performs the activities that allow the service to be effectively provided. For instance, suppose a cleaning company X (Service Provider) offers corporate cleaning services and has been contracted by bank Y (Service Client) to clean its headquarters. Due to internal personnel problems of company X, the cleaning of the bank's headquarters is being performed by freelancers (Service Executor) temporarily hired by company X. The freelance cleaners are the service executors from the bank Y's point of view, since the cleaning service has been offered, negotiated and contracted by company X. At the same time, from the company X's point of view, the freelance cleaners are the service providers of temporary cleaning services. In this example we have two different services, with the stakeholders playing different roles in each one. The first is the corporate cleaning service that bank Y (the service client and the service beneficiary) contracted company X (the service provider) to perform, but is actually performed by the assigned freelance cleaners (the service executors). The second service is the temporary cleaning service that company X (the service client) hired the freelance cleaners (the service providers and service executors) to perform at the premises of bank Y (the service beneficiary). The distinction between these two services allows the clear separation of responsibilities and obligations between Service Providers and Service Executors. Our definition of Service Executor is similar to the concept of service producer presented in [Ferrario and Guarino, 2008].

In our work we consider that the term service consumer conveys an intuitive notion of a stakeholder that contracts a service and/or benefits from the service's execution. Since in our conceptualization we have these two roles separated, we refined the OASIS' classification by introducing our specializations to the Service Consumer, namely the Service Client and the Service Beneficiary. To be consistent with the hierarchy of generalization and specialization, we also refined the service's offering counterpart by introducing the concept of Service Producer that can be specialized into Service Provider and Service Executor. Figure 3-2 depicts the hierarchy of the Service Consumer (Figure 3-2a) and of the Service Producer (Figure 3-2b).

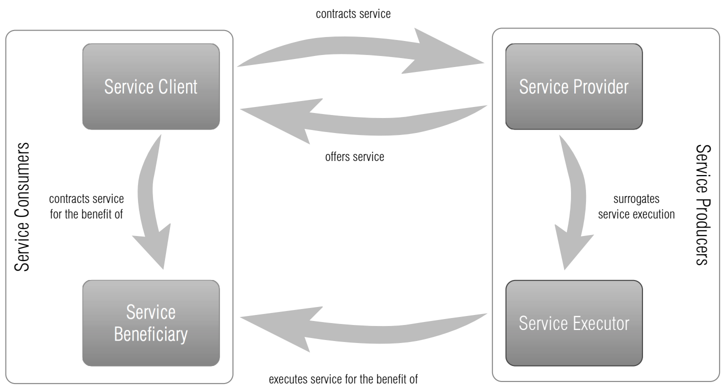
Figure 3-2 Main service stakeholders' roles



(a) Service consumers (b) Service producers

Figure 3-3 depicts the relationship among the identified types of stakeholders' roles. These roles are played by individual actors circumstantially, i.e., one actor can play the role of a Service Provider in one service provisioning instance and play the role of a Service Client in another service provisioning instance. The same applies to Service Beneficiary and Service Executor.

Figure 3-3 Service stakeholders' roles



3.3 Service Mediation

In scenarios with a small amount of services and service providers, or when the need for a service can be fulfilled by a previously used and known service, a service client can select the service by himself because *(i)* of the small amount of available services and providers; *(ii)* the service request is known and has been previously applied, or *(iii)* there is a previously established relationship between the client and the provider(s) facilitating the interaction. An example of this situation is a company that has an established contract with a provider of transportation services. In this case, only the types of transportation services offered by the contracted company are available to the employees of the delivery department. Moreover, the parameters of these available services (e.g., cost, pickup time and delivery time) have been pre-determined in the transportation service contract, so that the possible choices of services for the employee are limited and, therefore, can be made manually. Another example is the selection of energy supplying services. Normally there are only a few available companies providing this service in a given geographical area, easing the service selection.

However, whenever there is a need for a new service, and one requires the selection of the most appropriate service (based on a given criteria), and/or there is a large amount of available services and providers to consider, some sort of service mediation becomes necessary.

In the context of Service-Oriented Computing (SOA) [Erl, 2005] the term service mediation is often used as a reference to transformation components that tackle heterogeneity issues (e.g., semantic, process, data, etc) between services or between services and service clients [Barros et al., 2005, Confalonieri et al., 2005, Cimpian et al., 2006]. In this thesis we use service mediation in a broader sense, following the definition presented in [Laskey et al., 2009] of service mediator as *"a participant that facilitates the offering or use of services in some way"*. Examples of service mediators are service registries that allow the registration of service descriptions by service providers, facilitating the discovery of the available services, or proxies that actively stand for some party in an interaction. The yellow pages of the telephone directory can be considered as a service registry since it facilitates service discovery by organizing service providers in categories.

3.3.1 Software-based service mediation

The Web services architectural model (a SOA implementation) [Huhns and Singh, 2005] has three main parts: a provider, a client and a registry as depicted in Figure 3-4 . The registry is where service providers publish their service descriptions and where service clients search for services. A service client queries the service registry for services matching some criteria. Once found, the registry returns a list of matching service descriptions. It is the responsibility of the service client to selected a service from this list and to establish a connection with the service provider, so that the selected service can be provided. In other words, the service registry supports the client on the service discovery activity, leaving other activities, such as service selection, negotiation and activation, for the service client. This pattern of a service registry to store descriptions (service offers) was already prescribed in ODP, in the specification of the ODP trading function [ISO/IEC-ITU/T, 1998]

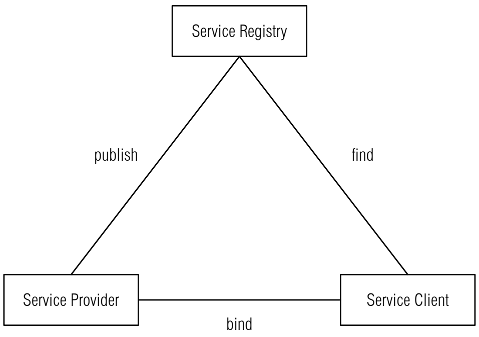


Figure 3-4 Web services architectural model

Figure 3-5 depicts the mediator in the Oasis' Service Reference Architecture. In this reference architecture, a mediator is mainly considered as an awareness facilitator, i.e., a participant that facilitates the awareness of service descriptions by service clients (or service consumers in Oasis terminology) This awareness is achieved by means of registry and repository facilities as depicted in Figure 3-6. The registry contains links or pointers to the service descriptions stored in the repository. Moreover, the Oasis approach considers the possibility of federated registries.

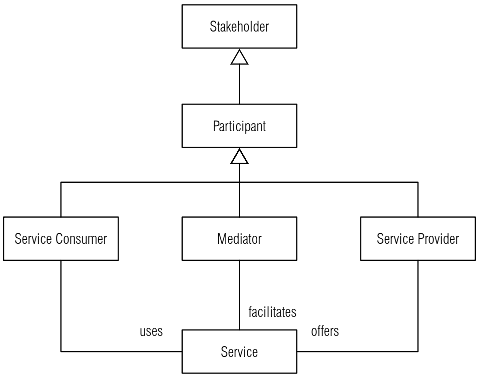


Figure 3-5 OASIS' service participants

In summary, a service client should be able to search for services in the available registries, decide which service best fits its needs, and invoke the service using the interface definitions defined in the service description. To facilitate these tasks, mediation entities can play a role supporting the interactions between service

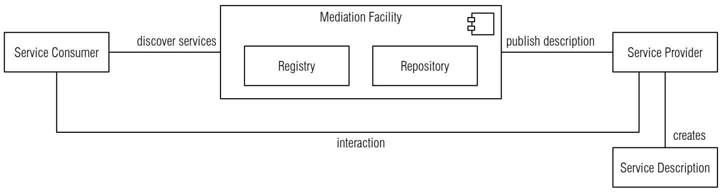


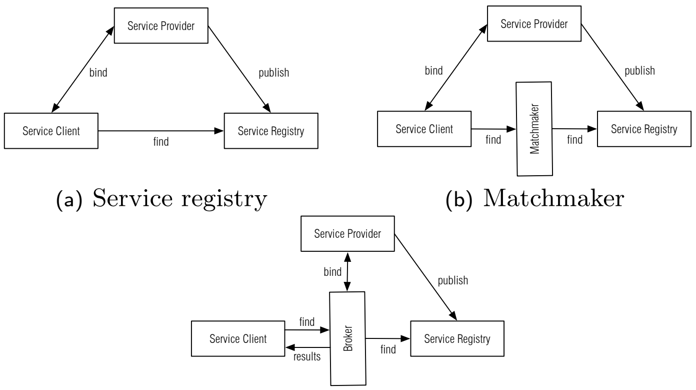
Figure 3-6 OASIS' mediation facility

clients and service providers. On the provider's side, a mediation entity can be beneficial by providing mechanisms for rapid creation, deployment and advertisement of services, such as in the examples presented in [Agarwal et al., 2005] and [Srinivasan et al., 2005]. On the client' s side, the mediation entity can support discovery, selection, composition and invocation of services, amongst others [Preist, 2004].

The characteristics of a service mediator varies according to the functionality it offers. The architecture of the mediation entities can be defined in terms of the type of interaction they offer to their users (in our case, service consumers and service producers). Following this approach, the level of support offered and the choice of the mediation functions determine the mediator's requirements. Here, we define as level of support the subset of the service provisioning activities offered by the service mediation entity (e.g., service discovery, selection and invocation).

Among the alternative types of roles for a service mediator we focus on two types, namely broker and matchmaker. The matchmaker provides a service discovery function by trying to match the client's service requirements with the service descriptions available at service registries. The broker can provide not only service discovery but also service selection, composition, invocation, transformation, amongst others, offering a higher level of support to the client [Bonino da Silva Santos et al., 2007].

Figure 3-7 shows different interaction patterns applied in three service architectures. In the basic service architecture depicted in Figure 3-7a the service provider publishes the service descriptions in a registry, the client queries the registry for the descriptions of available services and, after selecting an available service, the client invokes the appropriate service. The service architecture depicted in Figure 3-7b places a matchmaker between the client and the registry (or the set of accessible registries). In this way the client sends the criteria of the desired service to the matchmaker, which searches the available registries for service descriptions matching these criteria. In the case of a positive match, the matchmaker returns the description of the discovered service to be invoked by the client. The service architecture depicted in Figure 3-7c presents an example of the broker role. In this example, the software platform not only provides matchmaking facilities but also invokes the discovered services on behalf of the client. If necessary, the platform playing the role of a broker also performs transformations on the results received from the invoked services to comply with the data format requested by the client and only then sends the results to the client.



(a) Service registry (b) Matchmaker (c) Broker

Figure 3-7 Service mediator roles

3.3.2 Matchmaker

In service matchmaking we identify three distinct roles: a requester (service client), a matchmaker and a provider (service provider) [Decker et al., 1996]. The requester aims at finding services that offer the capabilities dictated by criteria provided in terms of the desired service interfaces and properties [Vasudevan, 1998]. The matchmaker has access to a set of services descriptions made available by providers, and provides facilities to discover services based on the requester's criteria.

Early computational directories offer matchmaking facilities that provide mappings between names and addresses similarly to the white pages in telephone directories. Later on, a more advanced form of matchmaking emerged that supports search based on entry's attributes, allowing matches based on certain desired characteristics. This form of matchmaking resembles the yellow pages in telephone directories. One shortcoming of this approach is that the selection criteria are completely supplied by the requester, providing an asymmetric form of selection [Facciorusso et al., 2003].[Hoffner et al., 2000] suggests the introduction of symmetry in the selection process, in which the requester provides a description of the requested service and its capabilities as a service client. The provider specifies its demand to the clients of its services, thus limiting the range of his service's potential clients. This allows the provider to select clients just as clients select services, fostering the establishment of relationships between compatible business partners. Otherwise, a client could select a service and, during the negotiation phase, find out that he is not a compatible client according to the provider's restrictions, wasting time and effort that could have been spared if the client-provider incompatibility issues have been addressed in earlier steps.

A matchmaker acts like as if it were a provider of services of different providers. The requester does not have to interact with several providers or several service registries querying them for the descriptions of their services and then try to match the descriptions with the needed criteria. In a matchmaking environment, the requester sends the criteria to the matchmaker, which searches services descriptions for a positive match. Having found services that comply with the given criteria, the matchmaker sends the service descriptions back to the client. The client then analyzes the services descriptions and selects suitable ones. After that, the client directly interacts with the services by invoking the service operations and receiving results. Figure 3-8 depicts an example of a sequence of interactions between a client, a service provider and a matchmaker platform.

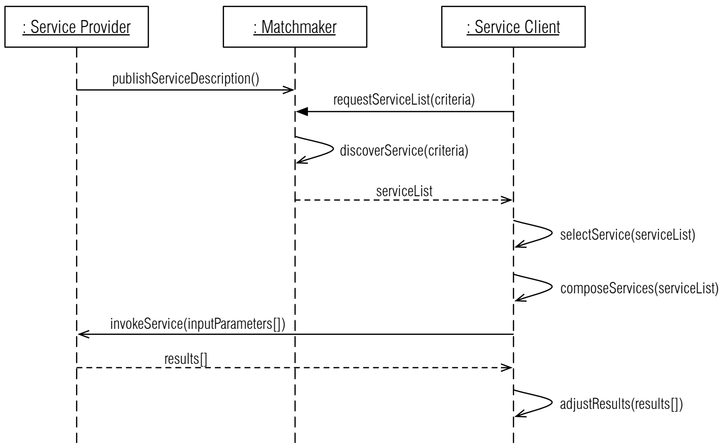

: Service Provider
: Matchmaker
: Service Client
publishServiceDescription()
requestServiceList(criteria)
discoverService(criteria)
serviceList
selectService(serviceList)
composeServices(serviceList)
invokeService(inputParameters[])
results[]
adjustResults(results[])

Figure 3-8 Interaction pattern for the service matchmaker role

The level of support of the matchmaker indicates the architectural components of the platform. In case some functionality is not supported by the matchmaker, e.g., selection or composition, other applications or the client application itself have to supply this functionality. Figure 3-9 depicts a possible architecture for both the matchmaker and the client [Bonino da Silva Santos et al., 2007] complying with the sequence diagram presented in Figure 3-8. The matchmaker supports service publishing by the service provider through the Service Publisher component. The Service Publisher sends the received service description to the Content Manager component, which is responsible for storing it in an available service registry. The Content Manager shields the internal components from interactions with registries. The Content Manager can have access to several registries and can also act as a client to other matchmakers.

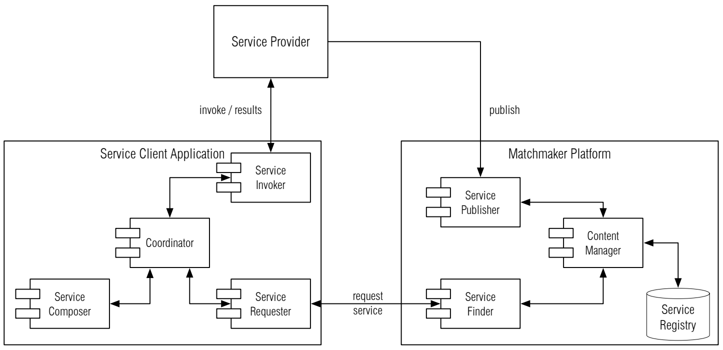


Figure 3-9 Example of a matchmaker platform architecture with service composition on client's side

In the architecture depicted in Figure 3-9 to illustrate the matchmaking process, a service client (through its client application) requests the discovery of services by calling the Service Finder component. Using the search criteria provided by the service client, the Service Finder requests a list of candidate services to the Content Manager. The Content Manager queries the available service registries for services matching the given criteria. If positive matches are found, the service descriptions of the discovered candidate services are sent back to the client.

Since in this architecture the matchmaker does not support service composition, this task is expected to be performed by the client. Therefore, it is possible that the matches have been obtained by a partial satisfaction of the criteria, i.e., some of the properties given by the service client have been satisfied but not completely by a single service. In this case, the service client needs to perform service composition. In Figure 3-9, the service composition task is performed by the Service Composer, which is a component internal to the service client application. The Service Composer component is responsible to find a service composition that fully matches the service client's criteria.

We consider now another example of matchmaker with a higher level of support than the one in Figure 3-9. In this example, the service composition functionality is provided by the matchmaker. Figure 3-10 shows that the Service Composer component can be moved from the client application to the matchmaker platform but its functionality remains the same. The Service Composer still tries to compose the services that partially match the service client's criteria into a service composition that fully matches the criteria. In case additional component services are required to complete the composition, the Service Composer needs to request the new services by providing other criteria. In the example where the service composition is performed by the client, the Service Composer requests the additional services to the Coordinator component. The Coordinator component forwards the request to the Service Requester component, which calls the Service Finder of the matchmaker with the new criteria.

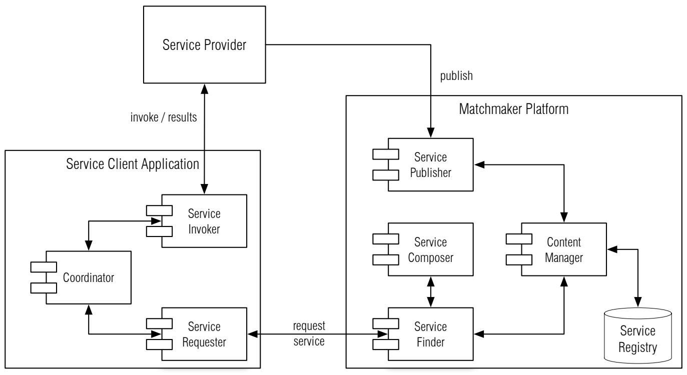


Figure 3-10 Example of a matchmaker platform architecture with service composition on matchmaker's side

In Figure 3-10, the composition is performed by the matchmaker and the criteria of the additional services to complete the composition are passed by the Service Composer directly to the Service Finder. Once the Service Finder discovers services that comply with the criteria, it returns their service descriptions to the Service Composer, which creates the composition and returns its description to the client.

The flexibility to assign functionality to the matchmaker or to the client shown in the examples in Figures 3-9 and 3-10 also holds for other functional components, such as the Content Manager or the Service Finder. Therefore, generic components can in principle be developed to support these functions, and the desired level of support for the service platform dictates where those components should be allocated [Bonino da Silva Santos et al., 2007].

3.3.3 Broker

In this role, besides the discovery functions discussed in the matchmaker role, the broker offers a service selection mechanism, invokes the services on behalf of the client, monitors the service execution and parses the results, possibly translating the output to client's required format. The broker can also perform service composition based on the client's service requirements and the available service descriptions.

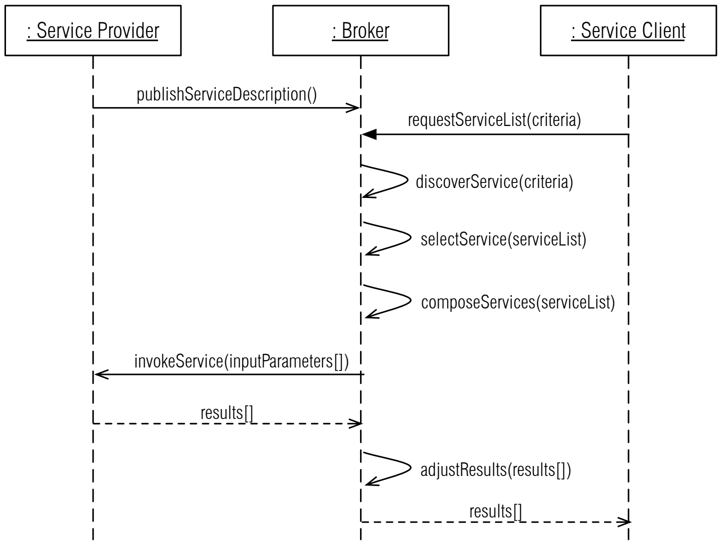
Service brokers may also consider the semantics of the terms present in service descriptions, service requirements and message exchanges. For instance, if the service description contains semantic annotations, the broker should be able to perform a set of complex reasoning tasks [Sycara et al., 2004], which includes interpreting service provider capabilities (service descriptions) and client applications's requirements. Similarly, the interpretation of the terms used in message exchanges can be performed by the broker platform.

Considering the broker role, in an example usage scenario the platform can be available to clients that may request immediate provisioning of a service. In this case, a service has to be discovered and consumed as soon as it is found, as opposed to a service to be discovered immediately and consumed when certain conditions hold at some point in future. The sequence diagram in Figure3-11 shows the interaction pattern between clients, service providers and the broker platform in this example situation. Similarly to the matchmaker, the broker provides facilities for service publishing by the service providers. A critical difference between the matchmaker and the broker is that the latter acts as a surrogate of the service client, i.e., the service client requests the provisioning of a given service and the broker performs the necessary steps until the final result of the service, on client' s behalf. Even if the output request by the service client is somewhat different from the output received by the broker after the invocation of the necessary services, the broker can perform data transformations to comply with the client's requirements. Examples of these transformations are:

1. The client requests a service that returns the postal address for a given point of interest. The broker finds a service that provides the geographical coordinates for points of interest in a given region. Since the client wants the address instead of the geographical coordinates, the broker can perform the output transformation by composing this service with another one that takes geographical coordinates and returns the re-

lated address;

2. The client requests a service that produces a given value in long number format. The broker finds the appropriate service but the output is in integer number format. The broker can perform the transformation of the output by simply parsing the integer value into long and returning the transformed value to the client.



: Service Provider
: Broker
: Service Client
publishServiceDescription()
requestServiceList(criteria)
discoverService(criteria)
selectService(serviceList)
composeServices(serviceList)
invokeService(inputParameters[])
results[]
adjustResults(results[])
results[]

Figure 3-11 Interaction pattern for the service broker role

Figure 3-12 presents an example architecture of a broker [Bonino da Silva Santos et al., 2007]. Here we see the components responsible for the functionality provided by the platform, namely the Service Publisher (service publishing), the Content Manager (service registry access and semantic repository access), the Service Finder (service discovery), the Service Composer (service composition) and the Semantic Mediator (semantic mediation).

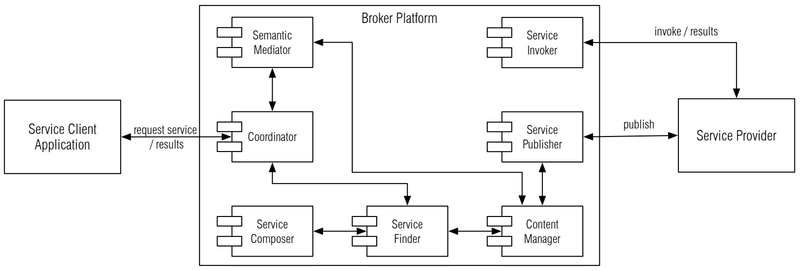


Figure 3-12 Example of a broker platform architecture

Additional characteristics of the broker platform role can be identified, such as:

1. Fault tolerance and robustness: if a service becomes unavailable, the platform can try to find another suitable provider for a similar service;
2. Privacy, security and billing: since we assume that clients and service providers have agreed to trust the platform, the platform is the trusting central point for these entities. Therefore, clients and providers do not have to directly interact, and the platform can provide anonymization for both parties.

Nonetheless, being a central point, the platform can become a single point of failure as well. Techniques, such as redundancy and clustering, among others, can be used to increase the platform's availability.

Unlike the matchmaker, since the broker role isolates clients from providers, it has less opportunities for exchanging functionality between the client and the platform. Although some auxiliary functionality, such as transformation of results could be placed on the client side, most of the functionality should remain on the platform side in order to preserve the essential purpose of the broker as a surrogate.

3.4 Service Provisioning Support Requirements

Based on the use cases and roles defined so far, we have identified a set of requirements for an approach aiming at supporting dynamic service provisioning, which are discussed in the following subsections. Although this is a non-exhaustive set of requirements for service discovery, these requirements are relevant for the purposes of our work in the areas of ambient intelligence and remote health care.

3.4.1 Domain independence and interoperability

The use case scenarios presented in Section 3.1 refer to different domains, namely, Ambient Intelligence, Health Care and Home Health Care. Other application domains in which Service-Oriented and Pervasive Computing can be applied should also be supported by the service mediation platform. Therefore, the proposed software platform should be able to support different domains while keeping the same functional and non-functional properties, and providing the same level of support for its users.

To provide service provisioning support in different domains, the software platform needs to be able to handle the specific characteristics of each of the required domains and also the aspects

aspects that are common to all domains. Issues such as a common vocabulary, types of stakeholders and the roles they play are among the domain-specific aspects that the supporting platform should be able to handle. To support different application domains, the software platform requires access and understanding to the knowledge necessary for the platform to properly operate in the domains. Domain-specific knowledge should be represented in the form of artifacts that are intelligible to the platform.

Moreover, the conceptualization of each supported domain should be shared among the platform's stakeholders to foster interoperability. To allow inference and reduce issues related to semantic interoperability, the interactions between the supporting software platform and its users should be semantically-enriched, i.e., the messages and description artifacts used by the platform's users should be semantically annotated with the concepts defined in the domain specifications. Furthermore, the internal operation of the platform is expected to benefit from the provided semantics. For instance, when searching for a service using a set of parameters, the platform can find a candidate service whose parameters are not exact matches but are similar, by applying subsumption [McGuinness and Borgida, 1995]. In order to interpret the parameters in a service offering, service providers should be able to use these conceptualizations to semantically annotate the terms in their service descriptions [Cimpian et al., 2008] enabling machine-driven service matchmaking. In other words, it is necessary that conceptualizations for each supported domain are available to the platform and these conceptualizations should be shared among the platform's stakeholders [Gruber, 1993].

3.4.2 Abstract service request

By targeting our service mediation support to non-technical end-users, we impose a restriction on how the users request service provisioning. We claim that non-technical end-users would have difficulties specifying service requests using current computer-based service technologies such as WSDL [WSD], WSMO/ WSML [de Bruijn et al., 2006, Haselwanter et al., 2006] and OWL-S [Martin et al., 2004]. It is fair to assume that non-technical users are not capable of using technical elements, such as data types, XML formatting, URLs, URIs, ports, among others, to specify a service request and interact with the discovered services [Rolland et al., 2007].

Moreover, we aim at facilitating the provisioning of services in a broader sense and not only of computational services. Therefore, users should be able to express requests that can be satisfied by either computational or social services.

3.4.3 Limited user interaction

Since we are considering Pervasive Computing and Services scenarios, constant requests for user interaction when devices and services need some information would lead to undesirable disruptions of the users' routine. Limited user interaction is therefore required because end-users (service consumers) should not have to frequently and obstructively interact with the supporting service platform to have services provisioned. Therefore, mechanisms to automatically and transparently gather information required by service discovery, selection, composition, triggering and execution should be offered by the framework, relieving the users of constant and disruptive interactions. The objective of this requirements is to limit the need and the number of direct end-user's interaction allowing the service provisioning to occur as transparently as possible.

3.4.4 Service provisioning automation

Our service mediation approach should be supported by a software platform that fully or partially automates service provisioning steps such as service discovery, selection, composition, negotiation, activation, triggering, execution and delivery. This software platform should support heterogeneous hardware and software, allowing connectivity from different devices so users can seamlessly interact with the platform using their desktop computers, notebooks, netbooks, PDAs, smartphones or other computing-capable devices.

This requirement relates to the requirement of limiting user interaction in Section 3.4.3 in the sense that it aims at facilitating end-users in having (fully or partially) automated service provisioning. By automating some of the steps of service provisioning, a limitation of the number of interactions the end-user need to have with services and service providers can be achieved.

3.4.5 Stakeholder support

The platform should support all its stakeholders with interfaces, APIs and tooling according to each stakeholder's objectives. Service Client stakeholders should be supported according to their technical expertise and based on typical needs of users in the application domain. For instance, a service client in the home health care domain, such as Peter and Sofia from the use case scenario

#3, could interact with the supporting platform through their TV screen, facilitating the visualization of the interface items. Since they may have some sight impairments, a simple and familiar GUI is advisable. In contrast, Maria (from the use case scenario #2) who is technologically savvy, interacts with the supporting service platform through a web interface on her computer as well as through her smartphone.

Service Provider stakeholders require supporting tools for tasks such as service description publishing and maintenance (insertion, update and deletion) and semantic annotation of the service descriptions.

Besides the provided user interfaces and tooling, the supporting platform should also define APIs to allow third-party client applications to interact with the platform. For instance, if in one deployment setting the provided service client's user interface is not appropriate, third-party developers can implement another user interface that interacts with the supporting platform through its defined APIs.

3.4.6 Maintainability and customization

To support maintainability and customization, the platform should have a modular design [Baldwin and Clark, 2000]. The modular design allows the substitution of particular elements according to evolution of the requirements, changes in the available technologies and specificities of applications domains. For instance, the platform's service composition component can be transparently upgraded for another component that implements a more efficient composition algorithm as long as it keeps the same interfaces and APIs.

Regarding customization, the modular design supports the use of different components based on the technical characteristics of a specific deployment setting. For instance, in a particular setting, the organization responsible for the service support platform chooses to use OWL as ontology representation language. Therefore, the domain ontologies defined using our supporting languages should be represented using OWL and stored in an OWL-compatible repository without affecting the operation of the rest of the platform.

3.5 Framework's Architectural Design

The objective of our work is to facilitate service provisioning to non-technical service clients. In order to accomplish this objective and comply with the presented service provisioning requirements, we started by defining a conceptual framework.

To define the framework architecture and its components we have analyzed the requirements for service provisioning discussed in Section 3.4 and identified elements that allow these requirements to be fulfilled. Since the main objective was the development of a concrete software platform, we adopted a bottom-up approach starting from this software platform. Then we moved up towards enabling technologies and techniques that could support the operation of the software platform and comply with the discussed requirements.

3.5.1 Software Platform

To realize service provisioning support, we propose a software provisioning platform to facilitate the interactions between service consumers and service producers. From the service consumer's perspective, the platform facilitates service provisioning by (fully or partially) automating service discovery, composition, invocation and monitoring, among others. From the service producer's perspective, the platform supports the publication of service descriptions. Figure 3-13 depicts the role of the software platform in supporting service producers and consumers.

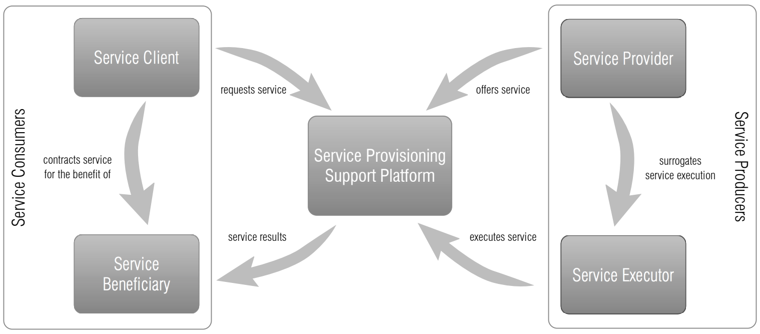


Figure 3-13 Platform support for service producers and consumers

The requirement discussed in Section 3.4.3 states the need for reduced user interaction. To have services provisioned, service consumers would need to interact with a service supporting platform in order to provide to the platform information such as criteria for service discovery, selection and composition, triggering conditions and inputs for service execution. Some of this information relates to contextual information, i.e., information characterizing the service client's context. For instance, when John (from use case scenario # 1) is at his beach house, the contextual information of his current location is used to activate the comfort services available at his beach house and not the services available at his city house or mountain cabin.

By including contextual information in our service supporting platform we have identified a new stakeholder role, namely, the *Context Provider*. A service that provides contextual information can be seen as a specialized kind of information service and, therefore, one could use contextual information services as component services in a service composition. This composition would use the outputs of the contextual information services as inputs for the other non-contextual services. However, in our work we chose to distinguish Context Provider from the more general-purpose Service Provider and consequently distinguish contextual information services from other services. This choice is related to the role that contextual information services play in our service supporting platform. We consider that non-contextual services are selected, contracted and executed to fulfill the service client's request. Therefore, we assume that contextual information services are used not to directly fulfill the service client's request, but as auxiliary services for other services and for the service platform.

In our service supporting platform, contextual information is used in three ways: *(i)* as input information for service execution (e.g., a temperature regulation service requires the temperature of a room to regulate the room's heating); *(ii)* as triggering of a service execution (e.g., a fire extinguishing service is triggered when a fire notification is received); *(iii)* as criteria for service discovery and selection (e.g., a service that lists the restaurants in a given region should only be discovered and/or selected if the service consumer is located or has interest in that region).

Whenever possible, information that would be directly given by the service consumer should be transparently gathered from the service consumer's context. Therefore, the Context Provider stakeholder role has the responsibility of supplying mechanisms that allow the supporting software platform to request and gather contextual information of service clients. These mechanisms should gather digitalized information from service client's software-based data such as profiles, calendar events, appointments, travel bookings, etc., or from sensor devices such as location (from motion detectors, GPS receivers, etc.), blood pressure, heart rate and weight, among others.

Since the use of contextual information characterizes the platforms, we have named our service provisioning support platform the *Context-Aware Service Platform (CASP)*. Figure 3-14 depicts the CASP interactions with service consumers, service producers and the Context Provider.

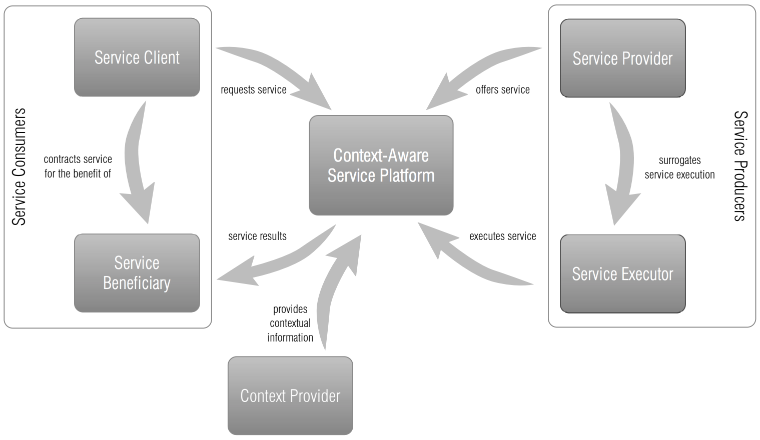


Figure 3-14 Service supporting platform with Context Provider

3.5.2 Domain knowledge sharing

According to the requirement presented in Section 3.4.1 the framework (and consequently, the CASP) should support dynamic service provisioning in different application domains. Operating in different domains implies that the supporting platform and its stakeholders should share the specific knowledge of each of these domains. To tackle this issue, the CASP and the involved stakeholders (service consumers, service producers and context-aware providers) should agree on the content of the shared knowledge [Gruber, 1995].

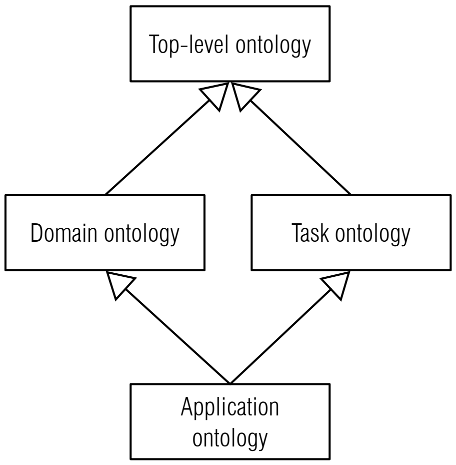
The objects, concepts, relations and other entities that exist in a given universe of discourse represent the domain knowledge and are formalized based on a *conceptualization* of that domain [Gruber, 1991]. A conceptualization is a set of concepts used to articulate abstractions of state of affairs in a given domain [Guizzardi, 2005]. However, these conceptualizations are abstract entities that only exist in the mind of its user(s). To be documented, communicated and analyzed, these conceptualizations must be captured in terms of concrete artifacts [Guizzardi, 2005].

Ontologies allow shared conceptualizations to be represented and manipulated [Gruber, 1993, Colomb, 2007, Guarino, 1998].

An ontology can be defined as an explicit specification of a conceptualization [Gruber, 1991] and has been applied in several areas in computer science. In [Mena et al., 1998, Hruby, 2005, Chandrasekaran et al., 1999, Falbo et al., 2002] the authors claim that ontologies can play a critical role in supporting semantic information brokering. In [Guarino, 1998] ontologies are classified in the following four categories according to their generality level (Figure 3-15):

– *Top-level ontology*, also referred to as upper-level ontology, core ontology or foundational ontology, describes general and domain-independent concepts and relations such as time, matter, endurants (objects), events, part-whole, among others, which can be used to construct models of specific domains [Guarino, 1998, Guizzardi, 2005].
– *Domain ontology* can be defined as a set of objects and the relationships among these objects of a particular domain of discourse [Gruber, 1993]. The domain knowledge captured in this ontology is reflected in the representational vocabulary that the domain ontology offers [Guarino, 1998].
– *Task ontology* describes domain-independent generic tasks or activities through the specialization of the terms defined in the top-level ontology [Guarino, 1998].
– *Application ontology* specializes the terms of both domain ontologies and task ontologies to describe concepts that depend of a particular domain and of a particular task. Normally, these concepts refer to roles played by a domain entity while performing a given task [Guarino, 1998].

Figure 3-15 Categories of ontologies



We use ontologies in our approach in order to provide sharable knowledge to the CASP's stakeholders and enable computer inference. Therefore, to support different domains the framework must have access to domain ontologies related to each domain intended to be supported. Having access to domain ontologies, the CASP is able to understand the concepts defined for each domain and reason about them.

The existence of domain ontologies implies that a stakeholder is responsible for defining and specifying these ontologies and supplying them to the CASP. This stakeholder gathers relevant knowledge of a particular domain and represents this knowledge in terms of a domain ontology. In the context of our framework we call this stakeholder role *Domain Specialist.* Figure 3-16 depicts the CASP interactions with service consumers, service producers, the Context Provider and the Domain Specialist.

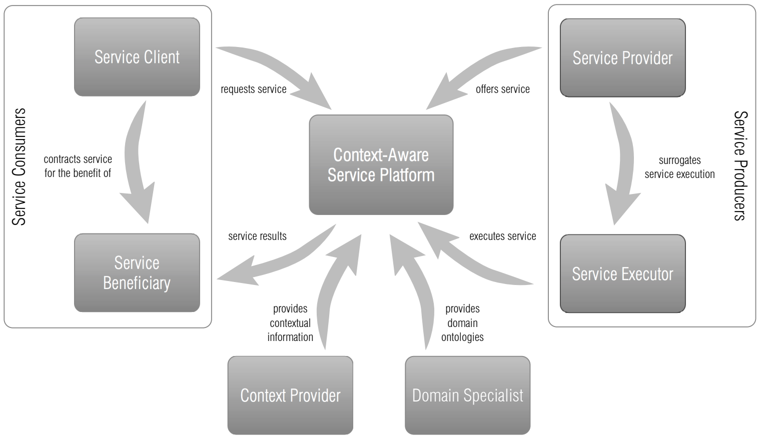


Figure 3-16 The CASP stakeholders including the Domain Specialist

In the requirement discussed in Section 3.4.5, some of the user support is given through semantic annotation of service descriptions and exchanged messages. Semantic annotation refers to assigning links to the terms in some text that point to semantic descriptions related to these terms [Kiryakov et al., 2004]. Since domain ontologies can be considered as semantic descriptions of the concepts in a given domain, we can use these ontologies to semantically annotate the terms in service descriptions and exchanged messages.

Similar terms may be used in different domains for different concepts or different terms may be used in different domains for similar concepts. For instance, in the geography domain Bermuda refers to a British archipelago in the North Atlantic Ocean while in the clothing or fashion domain the same term *bermuda* refers to knee-length walking shorts. Therefore, in our framework semantic annotation of service description, service requests, contextual information descriptions, etc. is performed in the scope of each domain ontology. For instance, a *location* context information may refer to geographical coordinates in a mapping domain and to an office number in a ambient intelligence domain. This domain dependency avoids semantic ambiguity issues.

Figure 3-17 depicts a layered view of the service provisioning framework in which the CASP supports semantically annotated services.

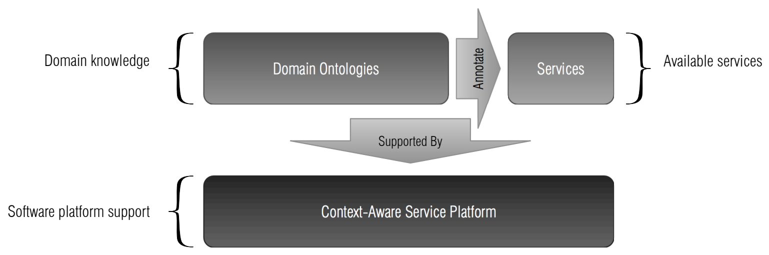


Figure 3-17 The 2-layer architecture of the service provisioning framework

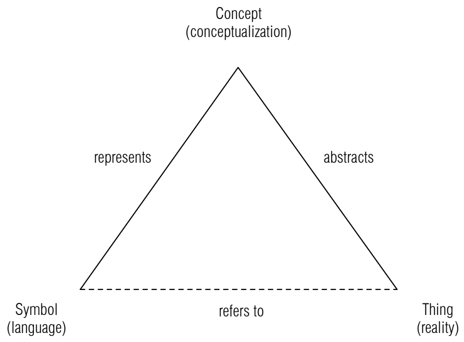
Semantic annotation is also beneficial for supporting Context Providers and for the use of contextual information by the supporting platform. Context Providers are expected to register the description of their offered contextual information to the platform. Similarly to the service descriptions, the contextual information descriptions are semantically annotated with the concepts defined in the domain ontologies. The semantic annotation of the contextual information description is used by the CASP for inferencing. For instance, a Context Provider offers GPS positioning information about selected service clients. When registering this offering to the platform, the Context Provider selects a domain ontology describing a domain that is applicable to his localization information and annotates the context information description with the concepts defined in this domain ontology. In the domain ontology that the Context Provider used to semantically annotate his context information description, a section containing concepts and relations related to contextual information defines GPS positioning as a specialization of location. Therefore, if the supporting software platform needs location information about a service client, GPS positioning services could be selected by applying subsumption.

3.5.3 Domain specification language

When modeling a domain, domain specialists construct in their minds an abstraction of that domain, i.e., a simplification of the reality aimed at representing the part of the real world that is of interest for the modeler's purpose. The set of relevant concepts used to create abstractions of a given universe of discourse is a *conceptualization* [Guizzardi, 2007]. However, conceptualizations and abstractions are immaterial entities that only exist in the mind of a user or a community of users. These immaterial entities should be represented in terms of concrete artifacts if they are intended to be communicated, analyzed and documented. In order to represent these mind-related entities in concrete artifacts, a language is necessary.

Figure 3-18 presents the *Ullman's triangle* [Ullmann, 1972] that depicts the relationships between a language, a conceptualization and the portion of reality that the conceptualization abstracts. The *represents* relation is related to the real-world semantics of a language, i.e., the interpretation of the language's primitives given by their connection with real-world entities. The relation between a language and the reality is depicted as a dotted line to indicate that this relation is intermediated by a certain conceptualization. In other words, the relation between a language's primitives and the real-world entities to which these primitives should be related is indirectly given through the conceptualization of the reality created by the language's designer.

Figure 3-18 Ullman's triangle



In the scope of SOC, standardized languages such as WSDL (for service description) and SOAP (for exchanged messages) do not provide semantics for the described operations and messages, but focus on the syntax of the messages. However, to use a service the client must understand the semantics of the service's operations and messages, their intended purpose and the intended content of all elements of their parameters [Verma and Sheth, 2007].

To address these issues, the Semantic Web Services (SWS) initiative builds upon Semantic Web and Web services technologies to provide semantic interoperability for services by allowing semantic annotation of service descriptions. Among the most prominent Semantic Web Services technologies are the Web Ontology Language for Services (OWL-S) [Martin et al., 2004] and the Web Service Modeling Ontology (WSMO) [De Bruijn et al., 2005]. OWL-S was proposed as an ontology for semantic markup of Web services with the objective to automate the discovery, invocation, composition and monitoring of services. WSMO was proposed with similar objectives but focuses on application integration problems [de Bruijn et al., 2006]. Some other efforts focus on more pragmatic solutions, considering the benefits of lightweight approaches, such as the Semantic Annotation for WSDL (SAWSDL) [Sheth et al., 2008] and the Web Services Semantics (WSDL-S) [Akkiraju et al., 2005]. SAWSDL and WSDL-S provide the mechanisms for semantic annotation of service descriptions assuming the availability of referral service ontologies.

However, since the objectives of these SWS approaches are to either provide mechanisms for semantic annotation on existing Web service interfaces (e.g., SAWSDL and WSDL-S) or to provide a ontology for services (e.g., WSMO and OWL-S), they lack capabilities to describe the concepts related to the domain in which the annotated service should operate. In these approaches, the semantics of the domain concepts used in service descriptions and exchanged messages are assumed to be defined somewhere else. This decoupling between domain and service-related semantics can lead to issues such as:

1. *Semantic conflict.* With two distinct semantics basis (one for the service-related concepts and other for domain-related concepts), the annotated service descriptions and exchanged messages may raise semantic conflicts [Ram and Park, 2004]. For instance, the service-related concepts may be based in a logical formalism that is incompatible with the formalism of the domain-related concepts preventing proper reasoning.
2. *Design effort.* The domain ontologies are generally defined by domain experts as a completely disjointed effort from the development of service ontologies. These two ontologies may have been defined using different languages and different tools or editors. Therefore, some harmonization efforts should be carried out to annotate services with these two ontologies.
3. *Domain appropriateness.* In some domains services must comply with specific regulations (service compliance)[Weigand et al., 2011]. Example of such domains are health care, defense, military, nuclear, among others. In these domains it is beneficial to specify a domain ontology that includes abstract

services or abstract tasks that can be performed by concrete services. For instance, in the health care domain, health services should comply with several regulations concerning privacy, confidentiality, reporting the occurrences of certain diseases, requirements for service provisioning (e.g., mandatory health insurance coverage), etc. Annotating services with integrated ontologies containing domain-related and service-related concepts fosters the enforcement of such regulations for all services implementing those abstract services or performing these abstract tasks.

As suggested in [Korotkiy, 2009], SWS efforts should consider shifting from ontology-enabled services to service-enabled ontologies. In other words, the SWS approaches should include and integrate service concepts in the domain specifications.

Domain ontologies can be used as concrete artifacts to represent domain abstractions. The abstraction of a domain defined by a domain specialist is based on his conceptualization of the reality. Since our framework is intended to operate in the scope of SOA, this conceptualization should be enriched with service-related concepts and relations. Therefore, the conceptualization includes primitives related to service-orientation such as service, service provider, service client, service description, etc. Consequently, the language that represents this conceptualization and is used to realize the domain abstraction in terms of domain ontologies contains these service-related concepts as primitive constructs. Figure 3-19 depicts the relations between the service-oriented conceptualization, domain abstractions, service-oriented modeling language and domain ontologies.

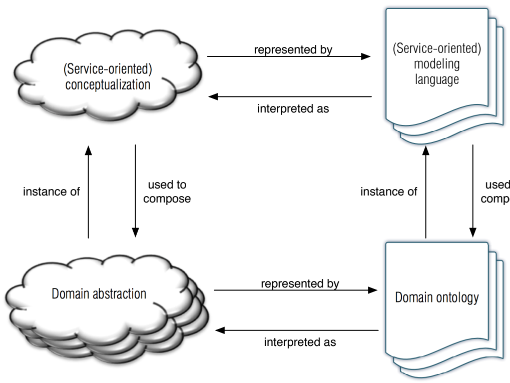


Figure 3-19 Relations between domain conceptualization, domain abstraction, modeling language and domain ontology

The specification of domain ontologies requires that the domain specialists utilize a language having primitives capable of representing concepts and relations of different domains, i.e., the language should be domain-independent. Our framework prescribes ontologies and the associated tooling, and offers a language with which domain specialists can create the necessary ontologies. Figure 3-20 depicts the layered view of our service provisioning framework including the domain specification language.

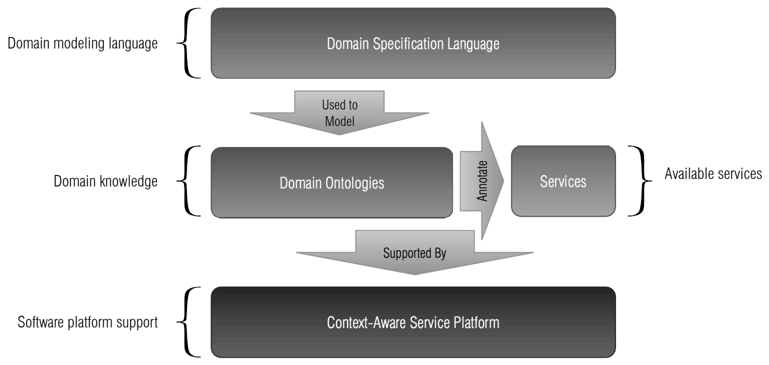


Figure 3-20 Service provisioning framework including the domain specification language

3.5.4 Foundational ontology

An ontology representation language such as the one we proposed for defining domain ontologies has a description of worldview embedded in its modeling primitives, i.e., the description of the conceptual model underlying the language [Guizzardi, 2005]. The conceptual model underlying a modeling language is called the *ontology of the language* in [Milton and Kazmierczak, 2004]. The ontology of a language explicitly describes the ontological commitments of the language.

In [Masolo et al., 2003, Guizzardi, 2005, 2006], it has been argued that ontology representation languages should be grounded on a meta-ontology that describes a set of real-world categories that can be used to talk about reality. In other words, an ontology representation language *should commit to a domain-independent theory of real-world categories that account for the ontological distinctions underlying language and cognition* [Guizzardi, 2005]. This meta-ontology (or domain-independent theory of real-world categories) is called a *foundational ontology* [Guizzardi, 2007]. A foundational ontology is sometimes also called upper-level ontology or top-level ontology.

Our domain specification language (an ontology representation language) is grounded on an extension of the Unified Foundational Ontology (UFO) [Guizzardi, 2005, 2007]. UFO has been developed based on theories from Formal Ontology, Philosophical Logics, Philosophy of Language, Linguistics and Cognitive Psychology. UFO is derived from a synthesis of two other foundational ontologies, namely, the General Formal Ontology (GFO), which underlies the General Ontological Language (GOL) [Heller and Herre, 2003, Degen et al., 2001], and the OntoClean ontology [Welty and Guarino, 2001] and the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [Gangemi et al., 2002]. While GFO/ GOL focuses on natural sciences and OntoClean/ DOLCE focuses on linguistic and cognitive engineering, UFO's main purpose is to provide a foundation for conceptual modeling [Guizzardi and Wagner, 2005].

Our extension to UFO added service-related concepts and relations to this foundational ontology. Therefore, the abstract syntax of our domain specification language is defined in terms of the language's metamodel derived from the extended foundational ontology. In other words, our domain specification language has its abstract syntax defined by its metamodel, and its ontological commitments are made explicit by the foundational ontology of which the language's metamodel is derived.

Figure 3-21 depicts the layered architectural components of our proposed framework, which are summarized as follows:

– *Foundational service ontology.* The foundational ontology defines domain-independent concepts such as service, service client, service provider, goal, task and their relations, among others. The extension of the UFO with service-related concepts allows the specification of service-oriented domain ontologies.
– *Domain modeling language.* Derived from the foundational ontology, the domain modeling language layer is comprised of the Foundational Service-Oriented Metamodel and the Service-Oriented Domain Specification Language. The Foundational Service-Oriented Metamodel represents the concepts and relations defined in the service foundational ontology and defines the abstract syntax of the Service-Oriented Domain Specification Language. The Service-Oriented Domain Specification Language is an ontology representation language that allows domain specialists to define and model domain ontologies, including the services available in a domain.
– *Domain knowledge.* Our proposed framework can be used in different domains, such as Health Care, Ambient Intelligence, etc. For each of these domains a specialist defines a domain ontology, namely the concepts and relations relevant to the domain, goals that users can have, valid tasks in the domain,

etc. The domain ontologies provide shared knowledge about particular material domains.

- *Offered services.* The services offered by Service Providers are semantically annotated with the concepts defined in the domain ontologies.
- *Software platform support.* The Context-Aware Service Platform (CASP) supports the interaction between service consumers, service producers and service clients. From the service producer's perspective, the platform supports the semantic annotation and publication of service descriptions. From the service consumer's perspective, the platform provides mechanisms for service discovery, composition, invocation and monitoring, among others.

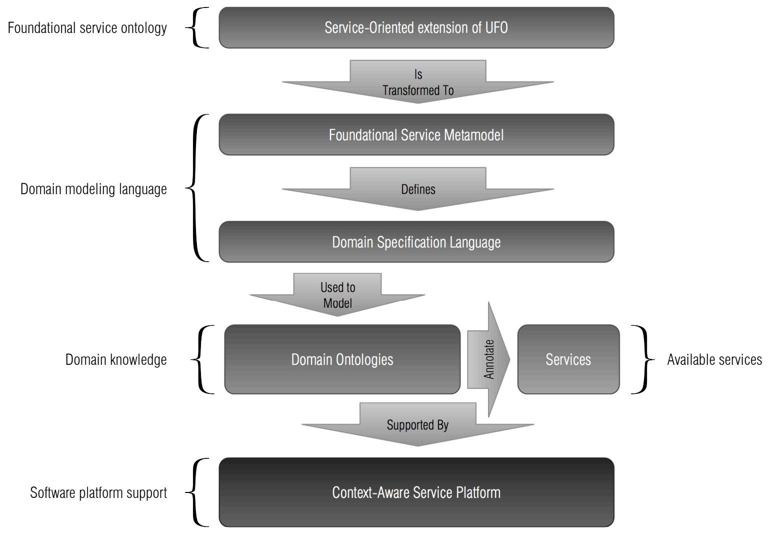

Foundational service ontology
Service-Oriented extension of UFO
Is Transformed To
Foundational Service Metamodel
Domain modeling language
Defines
Domain Specification Language
Used to Model
Domain knowledge
Domain Ontologies
Annotate
Services
Available services
Supported By
Software platform support
Context-Aware Service Platform

Figure 3-21 Main components of the Service Provisioning Framework

3.5.5 Role of the goal concept

In industry, SOC has been seen as an approach to integrate legacy and new systems with a standardized set of protocols and interfaces in a distributed manner. Semantically enriched service client's requirements can be expressed in terms of inputs, outputs, pre-conditions and effects, also known as IOPE. However, end-users, i.e., human service clients, may have difficulties to express such requirements, as they would have to deal with technical issues such as the request's language, and the type, format and coding of the IOPE. Additionally, if we consider the use of services in a Pervasive Computing environment where the end-users would have to deal with a significant number of services, computing devices, sensors, interfaces and actuators, the problem of expressing service requirements and providing the service inputs increases.

To fulfill the framework's requirement of allowing the abstract definition of service requests, we propose the use of the goal concept to express what service clients want to get accomplished by services. The use of goals aims at raising the abstraction level of the definition of service client's requirements and, therefore, facilitating service provisioning for end-users. By expressing service requests by means of goals, a service client defines what he wants to get accomplished, and the supporting service platform tries to discover and select services that could fulfill these goals.

Since the use of the goal concept is a central point in our framework, we have named it Goal-Based Service Framework (GSF). Similarly, the elements in the architectural layers depicted in Figure 3-21 are named stressing the use of goal as a central concept. Figure 3-22 depicts the architectural design of our proposed framework. In Figure 3-22, the framework components have been grouped into layers, which represent their purpose. The software platform support is realized in terms of the Context-Aware Service Platform, the the domain knowledge is realized by the domain ontologies, the domain specification language is realized by the Goal-Based Domain Specification Language and its related metamodel, and the foundational ontology is realized by the Goal-Based Service Ontology.

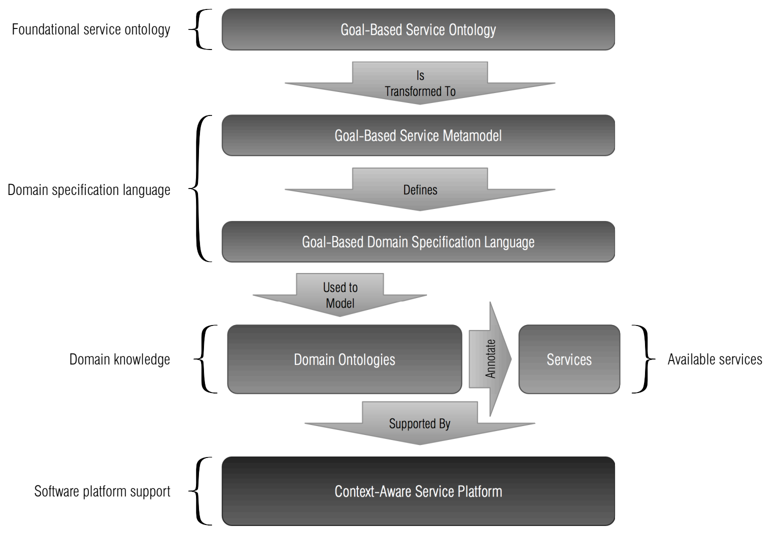

Foundational service ontology
Goal-Based Service Ontology
Is Transformed To
Goal-Based Service Metamodel
Domain specification language
Defines
Goal-Based Domain Specification Language
Used to Model
Domain knowledge
Domain Ontologies
Annotate
Services
Available services
Supported By
Software platform support
Context-Aware Service Platform

Figure 3-22 Components of the Goal-Based Service Framework

Chapter 4

Goal-Based Service Ontology

In this chapter we present the Goal-Based Service Ontology (GSO). The main objective of the GSO is to provide the ontological foundation for a domain specification language (introduced in Chapter 3), which allows domain specialists to define domain ontologies. These domain ontologies are intended to be registered in the Context-Aware Service Platform to provide domain-specific knowledge to the platform. To satisfy this objective, the GSO is defined based on ontologically sound concepts and relations, combining conceptual elements from the Service-Oriented Computing area such as Service Provider, Service Client, Service Agreement and Service, with conceptual elements pertinent to our goal-based approach, such as the Goal, Task, Intention, among others.

Additionally, in this thesis aim at demonstrating that dynamic and automatic service provision requires not only the modeling of the services, but also of the application domain where these services are to be offered and operate. Therefore, to characterize these service-populated domains, additional domain-independent concepts and relations are necessary, such as Agent, Intention, Material Relation, among others.

This thesis aims at defining service-related concepts that allow services to be more precisely described. Instead of defining from scratch the non-service concepts, we chose to reuse general-purpose concepts and relations (non service-related) from an existing foundational ontology. Therefore, GSO extends a foundational ontology, namely the Unified Foundational Ontology (UFO) [Guizzardi, 2005], by adding service-oriented and goal-based concepts and relations.

In Section 4.1 we briefly introduce the UFO, focusing on the concepts and relations that are relevant for understanding our GSO extensions, i.e., concepts and relations that either form the basis of the UFO such as *Universals, Individuals, Moments*, among others, and concepts which GSO directly extends. Section 4.2 introduces the GSO concepts related to the stakeholders identified in Chapter 3; Section 4.3 discusses how goals, tasks and services are related to each other in our ontology; and Section 4.4 justifies our modeling choices for the concepts related to services.

To present the concepts we have used UML class diagrams because of the widespread understanding of UML classes and relations, and their suitability for our purposes of illustrating how the presented concepts are organized, and how they relate to each other.

4.1 The Unified Foundational Ontology

The Unified Foundational Ontology (UFO) [Guizzardi, 2005] is a foundational ontology that has been defined based on a number of theories from Formal Ontology, Philosophical Logics, Philosophy of Language, Linguistics and Cognitive Psychology. UFO combines elements of two other foundational ontologies, namely the General Formal Ontology (GFO) [Degen et al., 2001], which underlies the General Ontological Language (GOL) [Heller and Herre, 2003], and OntoClean/DOLCE [Welty and Guarino, 2001, Gangemi et al., 2002].

UFO is divided into three incrementally layered compliance sets, namely UFO-A, UFO-B and UFO-C [Guizzardi and Wagner, 2005]. UFO-A is the core of UFO and defines concepts related to endurants. An endurant is an entity that is present as a whole at any given point in time, i.e., it does not have temporal parts and persists in time while keeping its identity. Examples of endurants are a house, an apple, and a person. If we say that in a given circumstance c_1 an endurant e has a property P_1 and in circumstance c_2 it has the property P_2, we refer to the same endurant e in each of these circumstances. For instance, we can say that one apple is green in one given day and is red five days later, but we still refer to the same apple even though some properties (in this case the color) have changed over time.

UFO-B builds upon UFO-A and defines concepts related to perdurants. Contrarily to endurants, a perdurant is an entity composed of temporal parts, i.e., its existence extends in time accumulating temporal parts. Examples of perdurants are a meeting, a concert presentation, an anniversary party and a business process. It is not always the case that whenever a perdurant is present all of its temporal parts are also present. With a perdurant, if we freeze time we can only see a limited number of parts of the perdurant and not the whole. For instance, in a "buy product" business process, if we take a snapshot at the point in time when the buyer is having its credit card validated in order to complete the payment, we cannot determine that this activity is part of the "buy product" business process. Finally, UFO-C extends UFO-B and defines social and intention-related concepts.

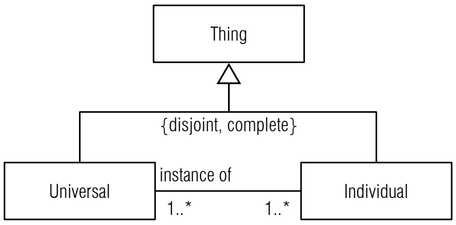
The UFO concepts and relations presented in this chapter are restricted to the ones relevant for understanding the extensions introduced in GSO. The Unified Foundational Ontology is more extensive than what we present here.

4.1.1 UFO-A: Individuals, Universals and Moments

UFO's main objective is to provide a foundation for conceptual modeling, aiming at describing *things* in the real-world. Therefore, the root concept of UFO, and UFO-A in particular, is *Thing*. From this concept, UFO-A defines two sub-concepts, namely *Universal* and *Individual*, as depicted in Figure 4-1. Universals and individuals are represented in conceptual modeling by the constructs of types (classes or classifiers in some modeling techniques) and their instances, respectively [Guizzardi, 2005]. Individuals are entities that possess unique identities, while universals are patterns of features, which can be realized in a number of different individuals [Guizzardi et al., 2008]. Therefore, considering that the symbol :: denotes the instantiation relation, we can define that [Guizzardi, 2005]:

$$\forall x, U \quad x :: U \rightarrow Individual(x) \wedge Universal(U) \tag{4.1}$$

Figure 4-1 UFO-A fundamental distinction between universals and individuals



Being a pattern of feature, a universal can represent types such as an airplane or a dog, attributes such as being fuel-efficient or being loyal, and relations such as flies the route X and belongs to. To distinguish between these sorts of universals, UFO-A defines the concepts of *Substantial Universal*, *Moment Universal* and *Relation*, as depicted in Figure 4-2. Substantials are existentially independent entities and they persist in time while keeping their identities [Guizzardi, 2005, Guizzardi et al., 2008]. Moments represent individualized properties. Moments are existentially dependent of other entities, i.e., they can only exist in other entities, which are their bearers. Examples of Substantial Universals include chair, person or building, while examples of Moment Universals include color, height or electric charge. Relations are entities that connect other entities together [Guizzardi et al., 2008] and are divided into *Material Relations* and *Formal Relations*. Formal relations hold between two or more entities directly, without any further intervening individual. Examples of formal relations include John *is taller than* Mary, and Pluto *is an instance of* Dog. Contrarily, material relations have a material structure of their own and have their relata mediated by individuals called *relators*. For instance, an employment is a relator that connects an employer and an employee, and a marriage is a relator that connects a husband and his wife.

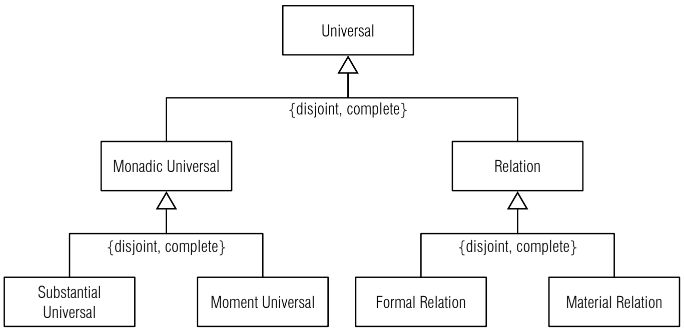


Figure 4-2 UFO-A - Moments and substantials

Given the employment example, let John be an instance of employee and Acme be an instance of employer. In this case, there is an individual relator *emp1* of the type employment that connects John to Acme. When this relationship between John and Acme is established, John acquires a set of moments because he becomes an employee of Acme, for instance, legal liability for disclosing classified information from the company, or his entitlement for the agreed salary. These new acquired moments are *instinsic moments* of John, i.e., these moments are existentially dependent of him. However, these moments not only depend on the existence John but also on the existence of Acme. In UFO-A, intrinsic moments that inhere in a single individual but are existentially dependent on (possible multiple) other individuals are named *Externally Dependent Moments*. Figure 4-3 depicts the *Externally Dependent Moment* concept and its relations with the *Intrinsic*

Moment, the *Relator* and the *Substantial* concepts of UFO-A.

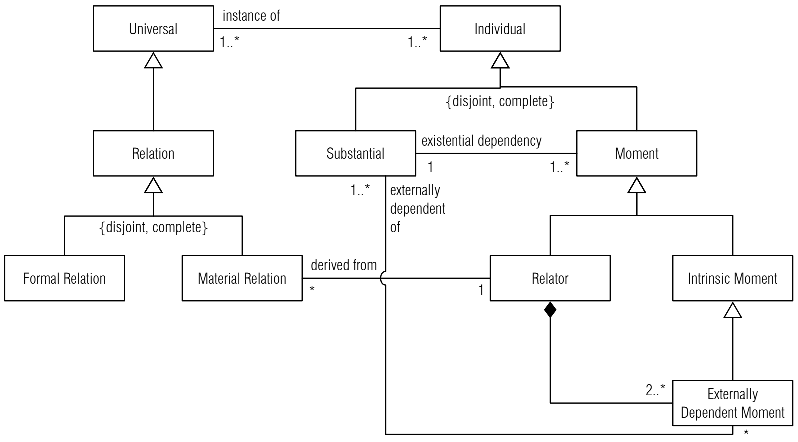


Figure 4-3 UFO-A - Externally dependent moment

For more details of UFO-A, see [Guizzardi, 2005] and [Guizzardi et al., 2008].

4.1.2 UFO-B: Perdurants

UFO-B is an ontology of events and, for that, it defines the distinction between endurants and perdurants. The fundamental point of this distinction is the behavior of these concepts with respect to time. Endurants *are in time*, meaning that they are wholly present whenever they are present even if some of their properties vary in time [Guizzardi et al., 2008]. For instance, a specific person *John* weighted 15 kg when he was 3 years old and weighted 80 kg when he was 20 years old. Although the weight property changed over time, it is the same individual we are referring to. Contrarily, perdurants are individuals said to *happen in time*, meaning that they are composed of temporal parts and they extend in time by accumulating these temporal parts [Guizzardi et al., 2008]. Examples of perdurants are a lecture, a football match, a racing or a concert presentation.

As depicted in Figure 4-4, depending on how they are structured in terms of their parts (their mereological structure), *Perdurants* can be specialized into *Atomic Perdurant* or *Complex Perdurant*. *Atomic Perdurants* have no improper parts while *Complex Perdurants* are aggregations of at least two perdurants (either atomic or complex). A perdurant can transform a portion of reality from an initial situation (pre-state) to another situation (post-state). Perdurants are ontologically dependent entities, i.e., perdurants existentially depend on their participants in order to exist. For instance, a lecture is a perdurant that only exists with the participation of the lecturer, the attendees and the venue (the substantial in the model). Therefore, according to the Figure 4-4, UFO-B's *Perdurants* can be seen from two perspectives: as time extended entities with certain (atomic or complex) mereological structures, and as ontological dependent entities which can comprise a number of different participations [Guizzardi et al., 2008].

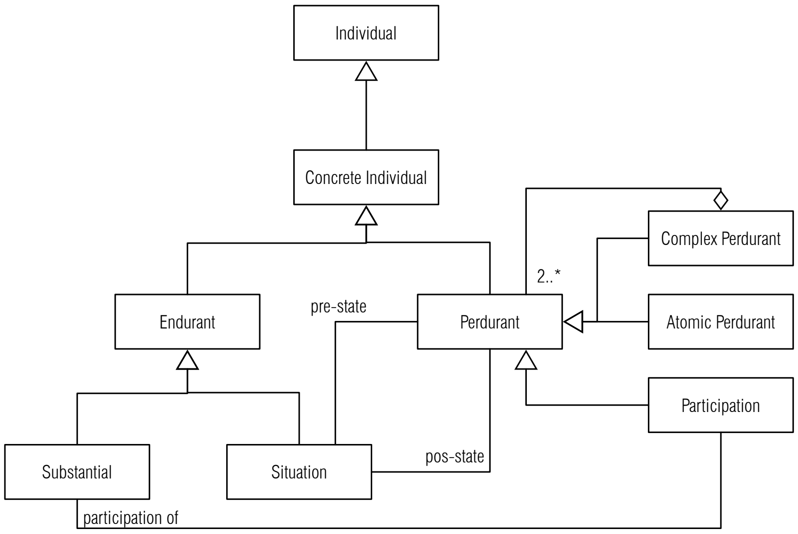


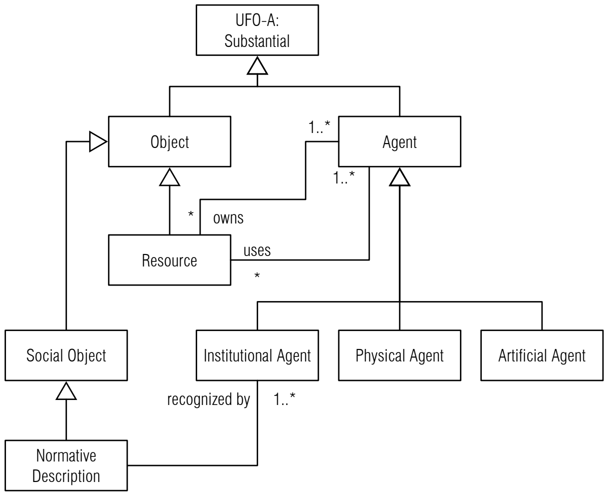
Figure 4-4 UFO-B - The Perdurant concept

4.1.3 UFO-C: Social concepts

UFO-C builds on top of UFO-A and UFO-B, and focuses on social entities in order to distinguish between agentive and non-agentive substantials. As depicted in Figure 4-5, the UFO-A concept of *Substantial* is specialized into *Agent* and *Object* [Guizzardi et al., 2008]. Agents are intentional agents that can be physical (e.g., a person, a pet), social (e.g., an organization, a society) or artificial (e.g., a software assistant application). *Objects* can be further specialized into *Resources* (e.g., a book, a desk, an information) and *Social Objects.* A *Resource* is an object that is used by an agent with specific purpose, and is owned by this or by another agent. For instance, an agent *Peter* uses a resource *pen* which is owned by agent *Mary.* In UFO-C, and consequently in GSO, the use of concepts related to agents does not imply the use of agent technology.

Social Objects include money, language, etc.. A special case of *Social Object* is a *Normative Description*, which defines a set

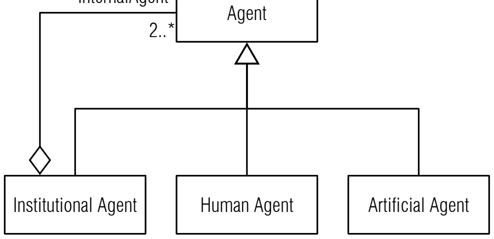
Figure 4-5
The Agent concept from UFO-C



of rules and norms recognized by social agents (e.g., institutional agents). The rules and norms of a *Normative Description* can describe other universals, such as moment universals, social objects and social roles (not represented in the Figure 4-5) [Guizzardi et al., 2008]. For instance, a country's law (a normative description) defines, by means of a set of rules and norms, social roles such as president, senators and governors. Other examples of normative descriptions, which are explicitly described, such as laws, or implicitly recognized, such as social conventions can also define the use of social objects, such as crowns, flags, money, among others.

As depicted in Figure 4-6, the specializations of *Agent* allow the distinction between human agents, artificial agents (software or hardware agents), and agents representing organizations or organization sub-parts, namely, the *Institutional Agent.* An *Institutional Agent* is composed of at least two internal agents, which can be again physical, artificial or institutional agents.

Figure 4-6
The UFO-C types of agents



In UFO-C, the characteristic that differentiates an agent from an object is that agents are the substantials that can bear special kinds of moments called *Intentional Moments*. According to John Searle in [Searle, 1998], intentionality should be understood as the *"...the various forms by which the mind can be directed at, or be about, or of, objects and states of affairs in the world"*. In other words, intentionality is *"the capacity of some properties of certain individuals to refer to possible situations of reality"* [Guizzardi et al., 2008]. As depicted in Figure 4-7, the UFO-A's *Instrinsic Moment* is specialized into the *Intentional Moment* concept, which is further specialized into *Mental Moment* and *Social Moment*.

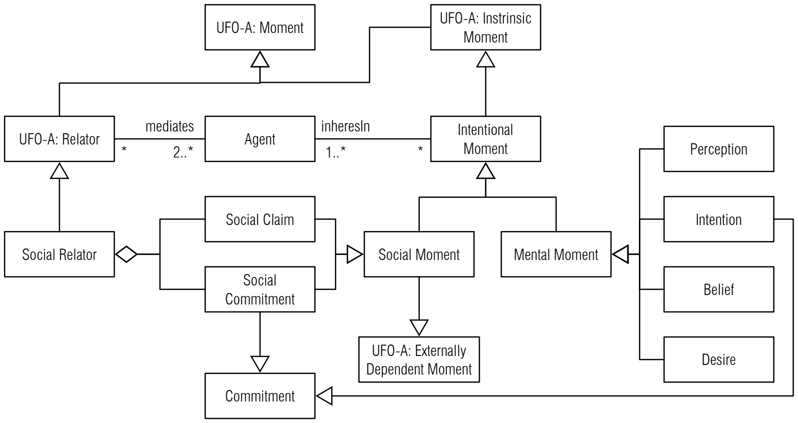


Figure 4-7 Agent and its intrinsic moments

Social Moment is defined in UFO-C as a specialization of the UFO-A concept of *Externally Dependent Moment*. Social moments represent intentional moments that are created as consequence of social interactions between agents and are further specialized into the concepts of *Social Commitment* and *Social Claim* [Guizzardi et al., 2008]. These concepts are used to regulate social relations among agents. As an example, a defendant hires a lawyer to represent him in court. When this social relationship is established (the hiring), the defendant commits to supply his lawyer with relevant information regarding a particular case. This commitment, or the promise to provide relevant information is a social commitment from the client towards his lawyer. This social relationship also creates a claim from the lawyer towards the defendant in which the lawyer can claim the information received from the defendant. A pair of commitment and claim relations constitutes a *Social Relator*, which is a specialization of the UFO-A *Relator*.

The agents in UFO-C follow the Belief, Desire and Intention (BDI) model [Rao and Georgeff, 1991] discussed in Section 2.3.

A *Mental Moment* is defined in UFO-C as an existentially dependent moment of a particular agent, i.e., it is an inseparable part of its mental state and, therefore, inheres in the agent. Mental moments can be further specialized into *Desire*, *Intention*, *Belief* and *Perception*. *Belief* is defined in UFO as any knowledge an agent has about the world. Examples include my belief that the Moon orbits the Earth, and my belief that Paris is the capital of France. *Perception* is a mental moment that expresses the relation between agents and events that are sensed from the environment and from other agents [Silva Souza Guizzardi, 2006]. *Desire* and *Intention* express a will of an agent towards a state of affairs in reality. In UFO-C, the difference between desires and intentions is that by intending something, an agent commits at pursuing it, i.e., an intention represents a desire plus an internal commitment [Silva Souza Guizzardi, 2006, Bonino da Silva Santos et al., 2009].

Every intentional moment has a type (e.g., belief, desire, intention, perception) and some propositional content [Guizzardi et al., 2008]. The propositional content is an abstract representation of the types of situations referred to by the related intentional moment. As depicted in Figure 4-8, situations can satisfy the propositional content of an intentional moment. In other words, a situation in reality can satisfy - in the logical sense - the proposition representing that propositional content [Guizzardi et al., 2008].

In UFO-C, the propositional content of an *Intention* is a *Goal*. Therefore, when an agent has an intention, it means that there is a goal that is the propositional content of that intention and the agent is committed to pursue its fulfillment. Alternative situations can satisfy (in the logical sense) the goal. For instance, if one has a goal of having a vehicle to go to work, a situation where he uses a car satisfies the goal as well as a situation where he uses a bicycle, unless additional constraints such as speed or comfort are defined in the goal. The definition for the concept of goal in UFO-C provides a clear distinction between goal, state of affairs (situation) and intention. As discussed in Section 2.6, the clarification of this distinction was one of the issues we intended to be solved in our goal conceptualization.

Because of the internal commitment that an agent has of pursuing the fulfillment of an intention, intentions cause the agent to perform actions. As depicted in Figure 4-9, an *Action* is a perdurant that is an individual instance of an *Action Universal*, with the purpose of satisfying the propositional content of an intention. As specializations of perdurants, actions can be atomic or complex. A *Communicative Act* is a type of atomic action. An

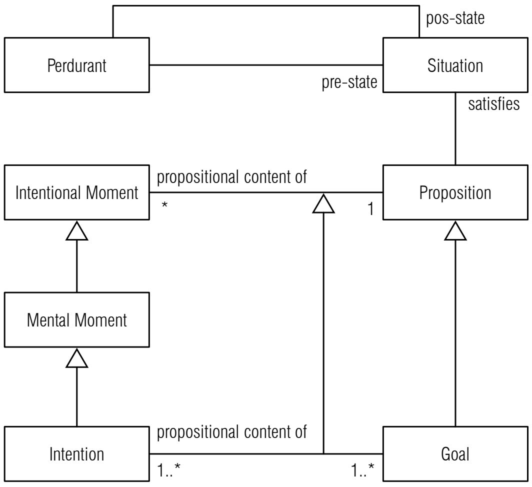


Figure 4-8 Intentional moment and its propositional content

example of communicative act is a speech act, such as a promise or the act of informing something. Complex actions are composed of at least two participations. Participations can be intentional or unintentional events. For instance, the winning of the 1993 Formula 1 Monaco Grand Prix by Ayrton Senna driving a McLaren car includes the intentional participation of Ayrton Senna and the unintentional participation of his Formula 1 car. UFO-C only considers the intentional participation of agents as an action, which is termed *Action Contribution*, while the term *Participation* is used to represent the unintentional participation of objects. When a complex action is composed of action contributions of different agents it is termed an *Interaction*.

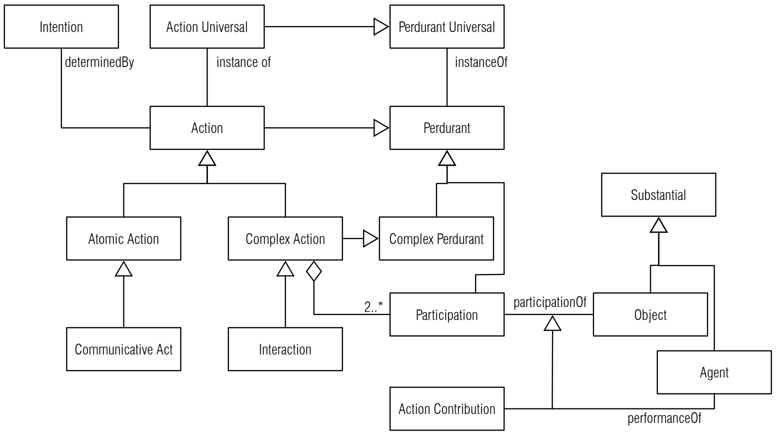


Figure 4-9 UFO-C's action concept

Concerning the principle of identity, agents can be either rigid or anti-rigid types. For instance, if we classify an agent as a person, and since a person is a rigid type, we have that an instance of a person cannot seize to be a person without seizing to exist. Conversely, an agent can be classified as a student (an anti-rigid type) in a period of his life and a non-student in another period without seizing to be the same agent. Therefore, in this last example, although the circumstantial classification of being a student incorporates some characteristics to the agent, it cannot be the main identification of this individual, since it does not persist in his whole existence. To allow the distinction of rigid and anti-rigid types for agents, UFO-C defines two specializations for *Agent Universal*, namely *Agent Kind* (rigid type) and *Agent Role* (anti-rigid type) [Silva Souza Guizzardi, 2006] as depicted in Figure 4-10.

Figure 4-10 Rigid and anti-rigid agent types

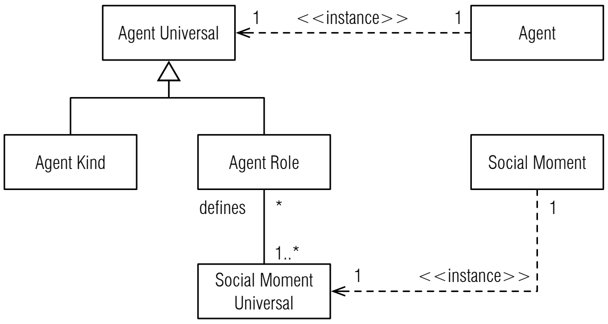


Figure 4-10 also depicts the concept of *Social Moment Universal* associated with the *Agent Role* concept. A social moment type defines the agent's role by describing the set of general commitments and general claims that the agent has when playing this role [Silva Souza Guizzardi, 2006]. For instance, when agreeing to play the role of a "professor", one is committing oneself to "teach courses" and "supervise students".

For more information on UFO-C, see [Silva Souza Guizzardi, 2006] and [Guizzardi et al., 2008].

4.2 Service stakeholders in GSO

In Chapter 3.2 we have identified the stakeholders that participate in service provisioning, namely the Service Provider, Service Executor, Service Client and Service Beneficiary. In real-world situations involving these stakeholders, we can observe that being a service provider, a service executor, a service client or a service beneficiary is circumstantial, i.e, at one point in time an individual could play the role of a service client and, at another point in time the same individual could play the role of a service provider. For instance, when a person who has a job as consultant travels to meet a client, he can be the service client of transportation and lodging service at one moment, and a service provider of the related consultancy services when he meets with his client. Therefore, in our work, we decided to relate these stakeholders to the UFO-C concept of *Agent Role*. Moreover, we have aggregated these stakeholders into two categories, namely, *Service Producer* and *Service Consumer* as depicted in figure 4-11. Service consumers are specialized into *Service Client Universal* and *Service Beneficiary Universal*, while service producers are specialized into *Service Provider Universal* and *Service Executor Universal*.

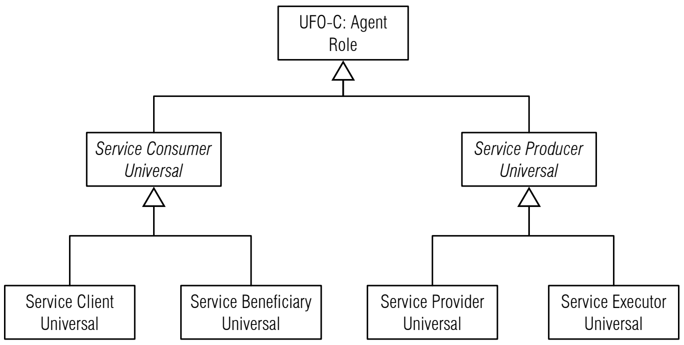
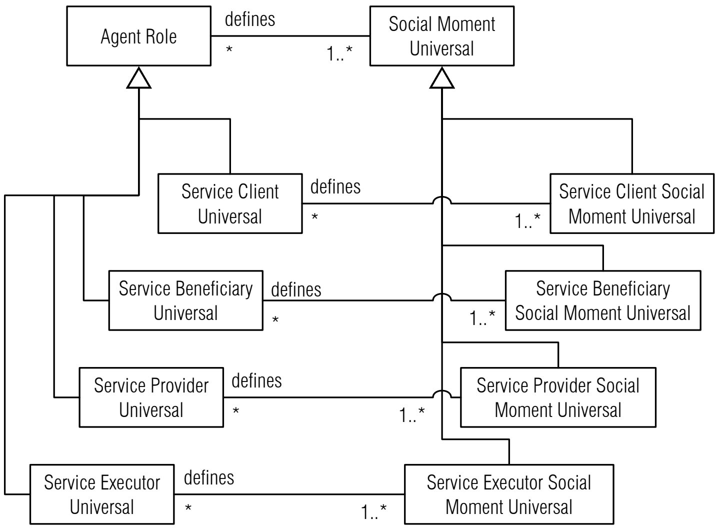

UFO-C: Agent Role
Service Consumer Universal
Service Producer Universal
Service Client Universal
Service Beneficiary Universal
Service Provider Universal
Service Executor Universal

Figure 4-11 GSO's stakeholders types

As discussed in Section 4.1.3, agent roles are defined by a set of general commitments and claims described by an associated *Social Moment Universal*. These commitments and claims define obligations and rights that individuals inhere when playing an agent role. Figure 4-12 depicts that each of the GSO's stakeholders has a social moment type associated to it that defines the obligations and rights of the stakeholder. The *Service Client Universal* is associated with a *Service Client Social Moment Universal* describing that when individuals play this role they are responsible for searching, negotiating and contracting a service. The *Service Beneficiary Social Moment Universal* associated with the *Service Beneficiary Universal* describes that the individual playing this role is entitled to perceive the benefits of the execution of the service contracted by the service client. The *Service Provider Social Moment Universal* associated with the *Service Provider Universal* describes that the individual playing this role is responsible and liable for the service it offers. Moreover, the service provider can negotiate conditions and parameters related to its offered services, such as triggering conditions, costs, quality of service, among others. Finally, the *Service Executor Social Moment Universal* defines that an individual playing the role of a *Service Executor Universal* is responsible for the execution of a service on behalf of the service's *Service Provider Universal.* Consequently, the service executor is liable for the service execution with respect to the service provider that hired it. However, this does not imply that the service executor is liable with respect to the service client, since the service provider is the liable entity in this case. In other words, although a third-party (the service executor) actually executes the service, from the service client's perspective, the service provider is responsible and liable for the services it offers.

Figure 4-12 GSO's stakeholders types


Agent Role
defines
*
1..*
Social Moment Universal
Service Client Universal
defines
*
1..*
Service Client Social Moment Universal
Service Beneficiary Universal
defines
*
1..*
Service Beneficiary Social Moment Universal
Service Provider Universal
defines
*
1..*
Service Provider Social Moment Universal
Service Executor Universal
defines
*
1..*
Service Executor Social Moment Universal

As in UFO-C, GSO also distinguishes human, artificial and institutional agents. However, in our work we consider artificial agents as surrogates of the human or institutional ones. In other words, we adopt the notions of BDI, but extended these notions by considering that artificial agents act on behalf of the individual humans or institutions that own, deploy and use them. Thus, these artificial agents' beliefs, desires and intentions have been given by their designers and/or owners. The consequence of this approach is that we are able to capture the rationale behind the use of an artificial agent, i.e., we can assess and represent the motivations of the humans or institutions behind the artificial agent. Figure 4-13 depicts how *Human*, *Institutional* and *Artificial Service Client Universals* are defined in GSO. In this figure, when a *Human Agent Universal* plays the role of a *Service Client Univer-*

sal, the human agent is classified as a *Human Service Client Universal*. Analogously, when an *Institutional Agent Universal* plays the role of a *Service Client Universal*, the institutional agent is classified as an *Institutional Service Client Universal*. Both the human and institutional service clients can make use of artificial agents for the purposes of their role as service clients. When this happens, the *Artificial Agent Universal* they use is classified as an *Artificial Service Client Universal*.

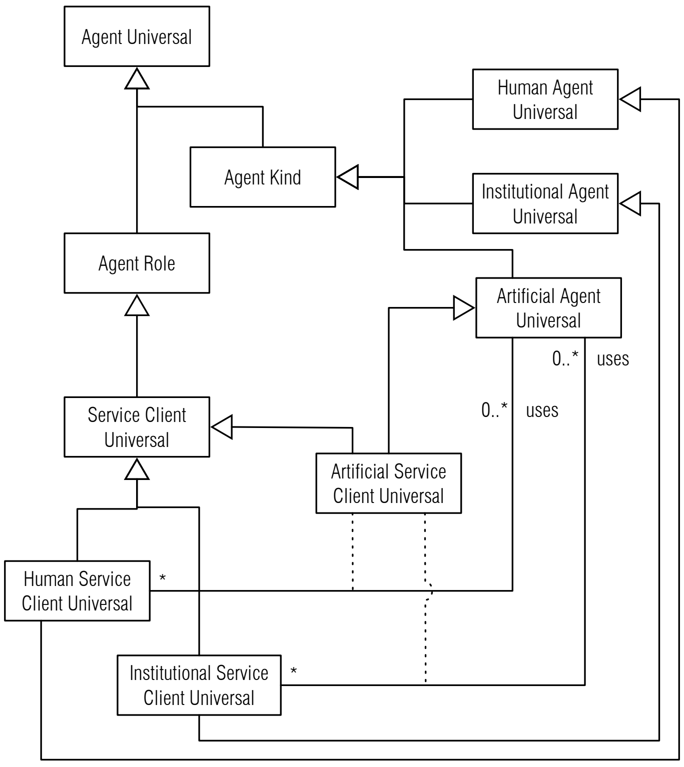

Agent Universal
Human Agent Universal
Agent Kind
Institutional Agent Universal
Agent Role
Artificial Agent Universal
0..*
uses
0..*
uses
Service Client Universal
Artificial Service Client Universal
Human Service Client Universal
*
Institutional Service Client Universal
*

Figure 4-13 Relations between human, institutional and artificial service clients

The same relations we defined for *Human*, *Institutional* and *Artificial Service Client Universals* apply for *Human*, *Institutional* and *Artificial Service Beneficiary Universals*; *Human*, *Institutional* and *Artificial Service Provider Universals*; and *Human*, *Institutional* and *Artificial Service Executor Universals* as depicted in Figures 4-14, 4-15, and 4-16, respectively.

In our work, besides the regular service consumers and service providers, we have defined a special type of stakeholder, namely the *Context Provider*. The *Context Provider Universal* is a specialization of the *Service Provider Universal* that provides informational services. Differently from the *Service Provider Universal*,

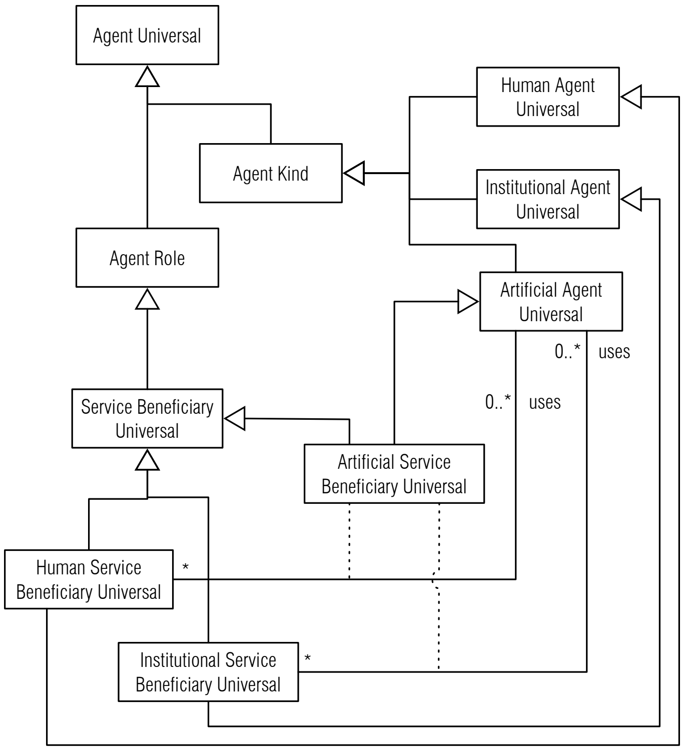

Agent Universal
Human Agent Universal
Agent Kind
Institutional Agent Universal
Agent Role
Artificial Agent Universal
0..*
uses
0..*
uses
Service Beneficiary Universal
Artificial Service Beneficiary Universal
Human Service Beneficiary Universal
*
Institutional Service Beneficiary Universal
*

Figure 4-14 Relations between human, institutional and artificial service beneficiaries

which provides a variety of services that can directly satisfy the *Service Client*'s goal, the informational services provided by the *Context Provider Universal* are used by our proposed Context-Aware Service Platform to support either the discovery, selection, composition and triggering of services or to provide input information for these services. Therefore, the context-aware informational services are not used directly to satisfy a service client's goal but are used to support the provisioning of these goal-satisfying services. Figure 4-17 depicts the *Context Provider Universal* defined as a sub-type of the *Service Provider Universal*. Since the *Context Provider Universal* is a specialization of *Service Provider Universal*, it can also be further specialized into *Human Context Provider Universal* or *Institutional Context Provider Universal*, and both can use an *Artificial Context Provider Universal*.

4.3 Goals, Tasks and Services

Figure 4-18 depicts the *Goal* concept of GSO and how it is related to the concepts of *Task* and *Service*. In GSO, we have extended

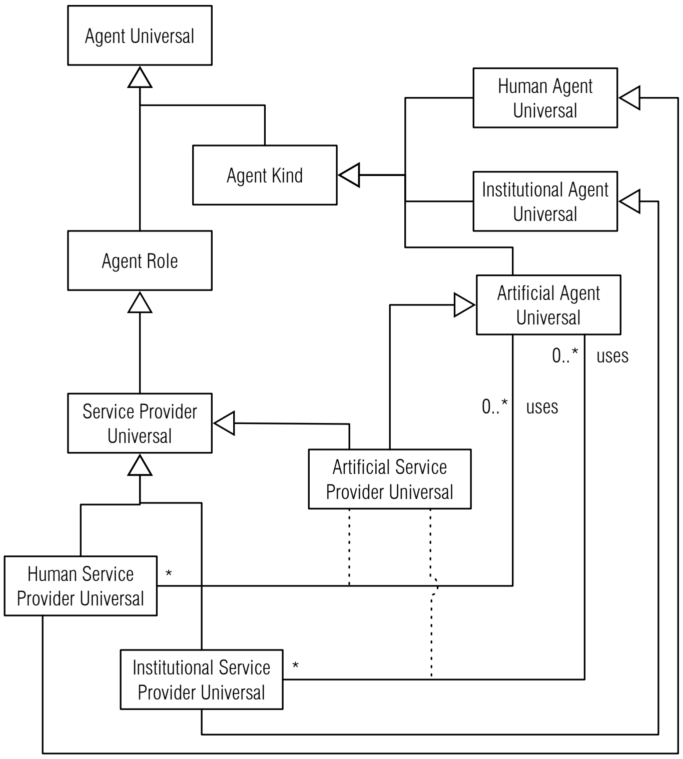

Agent Universal
Human Agent Universal
Agent Kind
Institutional Agent Universal
Agent Role
Artificial Agent Universal
0..*
uses
0..*
uses
Service Provider Universal
Artificial Service Provider Universal
Human Service Provider Universal
*
Institutional Service Provider Universal
*

Figure 4-15 Relations between human, institutional and artificial service providers

the UFO's goal concept to define that a Goal can be owned by a Service Client Universal. This ownership relation defines a meta-commitment, so that individual instances of the Service Client Universal have a goal of a certain type. More specifically, let C be a service client and g a goal, we have that C owns g iff for every instance x of C there is an intention I which is an intrinsic property of x (inheres in x) and g is the propositional content of I. Since the concept of *Goal* is a specialization of the concept of *Proposition*, the distinction between *Universals* and *Individuals* does not apply.

A domain specialist can define goals for different types of service client universals in a domain. For instance, in the health care domain, a *Doctor* (in this example an instance of Service Client Universal) can be specified as owing the goals *BeAccredited-ByHealthInsurance*, *KeepUpdatedWithMedicalAdvancements*, etc., while *Medical Patients* (another instances of service client universals) own the goal *GetHealthier*. In this example, whenever an individual plays the role of a doctor it inheres the goals *BeAccredit-edByHealthInsurance* and *KeepUpdatedWithMedicalAdvancements*.

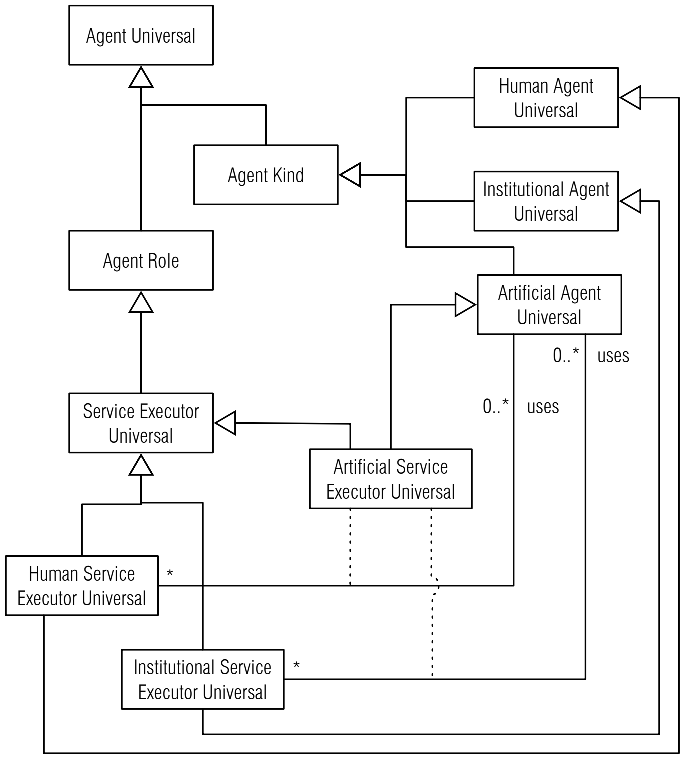

Agent Universal
Human Agent Universal
Agent Kind
Institutional Agent Universal
Agent Role
Artificial Agent Universal
0..*
uses
0..*
uses
Service Executor Universal
Artificial Service Executor Universal
Human Service Executor Universal
*
Institutional Service Executor Universal
*

Figure 4-16 Relations between human, institutional and artificial service executors

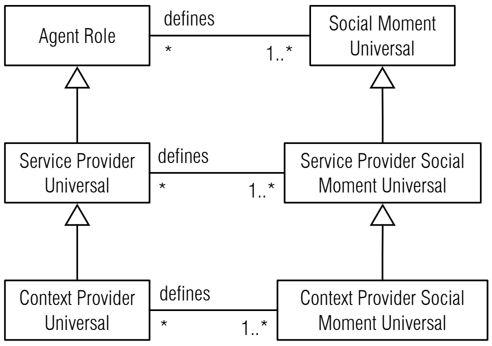

Agent Role
defines
*
1..*
Social Moment Universal
Service Provider Universal
defines
*
1..*
Service Provider Social Moment Universal
Context Provider Universal
defines
*
1..*
Context Provider Social Moment Universal

Figure 4-17 The Context Provider agent role

Similarly, in the same example, whenever an individual plays the role of a medical patient, it inheres the goal *GetHealthier*. Moreover, an individual instance of *Doctor* can be defined as being able to circumstantially play both the roles of a service client and of a service provider. When the doctor performs a medical treatment he is providing a service that fulfills the patient's *GetHealthier* goal and, therefore, is playing the role of a service provider. Conversely, when the same doctor uses the accreditation services of a health insurance company, the company is providing the service that fulfill the doctor's service client goal *BeAccreditedByHealth-Insurance.*

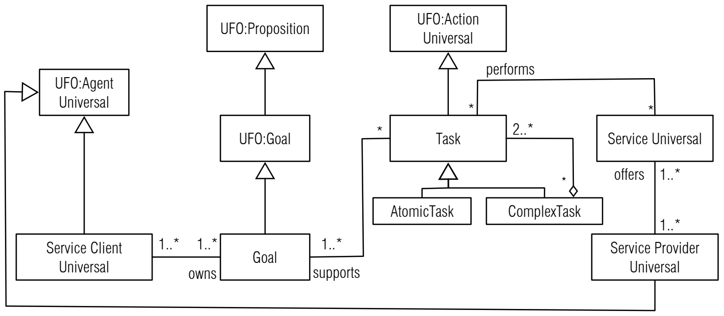


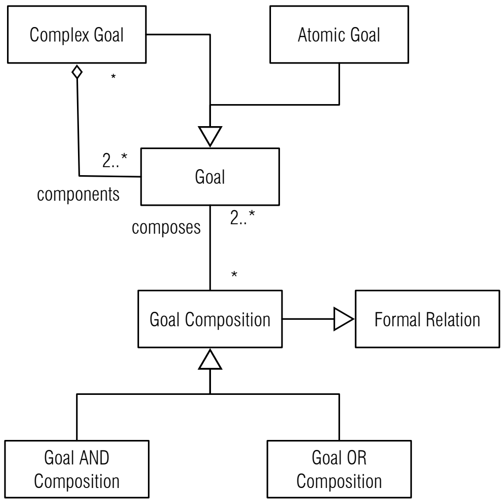
Figure 4-18 Relations between goals, tasks and services

A *Task* in GSO is defined as a specialization of the UFO concept of *Action Universal.* An Action Universal in UFO is an intentional event, i.e., an event performed by one or more agents in order to accomplish a goal. In GSO we adopted the term task for this concept to keep it aligned with the terminology used in i* and Tropos to denote entities that are the means to accomplish and fulfill goals. Therefore, to fulfill a goal an agent performs a task or delegates its execution to another agent. In Figure 4-18, the relation *supports* between task and goal represents that a successful execution of that task satisfies that goal.

In the realm of SOA, we can say that the fulfillment of a service client's goal is achieved by means of a service. In Section 4.4 we argue that a service is more than just the task it performs. However, every service is connected to a task the service provider agrees to be performed when the service execution is triggered. Consequently, in Figure 4-18, the relation *performs* between *Service Universal* and *Task* (Action Universal) represents that instances of *Task* are executed when the associated service is executed. From the SOA point-of-view the term task also complies with the W3C's Service-Oriented Model [Booth et al., 2004], where a service performs a service task.

As depicted in upper half of Figure 4-19, a Goal can be specialized into *Atomic Goals* or *Complex Goals.* A *Complex Goal* is composed of two or more goals that can be also atomic or complex goals. However, the way a goal is composed (or decomposed) has an impact on the goal's fulfillment. For instance, if a given goal *G1* can be composed of two sub-goals *SG1* and *SG2*, we can have two possible situations: *(i)* the fulfillment of *G1* is achieved by the fulfillment of both *SG1* and *SG2*, or *(ii)* the fulfillment of *G1* is achieved with the fulfillment of either *SG1* or *SG2*. To cope with these two situations we have defined the *Goal Composition* relation, which is a specialization of the UFO's *Formal Relation*.

Figure 4-19 Goal composition



As depicted in Figure 4-19, the *Goal Composition* relation can be further specialized into the *Goal AND Composition* and *Goal OR Composition* relations. The *Goal AND Composition* is used to represent relations where the fulfillment of the goal is only achieved with the fulfillment of all sub-goals, while the *Goal OR Composition* is used to represent relations where the fulfillment of the goal is achieved with the fulfillment of one of the alternative sub-goals. In a model expressed using GSO, a mixture of these two relations can be used. For instance, in a GSO model of the medical domain, a high-level goal *GetMedicalTreatment* is fulfilled when its sub-goals *GetMedicalConsult* and *GetMedicinePrescription* are fulfilled, or when either one of the sub-goals *GetHomeMedical-Treatment* and *GetHospitalizedMedicalTreatment* is fulfilled.

Figure 4-20 shows the causal chain of goal satisfaction. An intention (of which a goal is its propositional content) causes an action (an instance of a Task) to be performed, i.e., since the agent is committed to the goal satisfaction, he acts accordingly to pursue its satisfaction. The action creates a situation that satisfies the goal.

The use of situations to characterize goal satisfaction opens the possibility of using Fuzzy Logics mechanisms to assess partial satisfaction of goals (if necessary). Depending on the domain being specified using GSO, the domain specialists can define de-

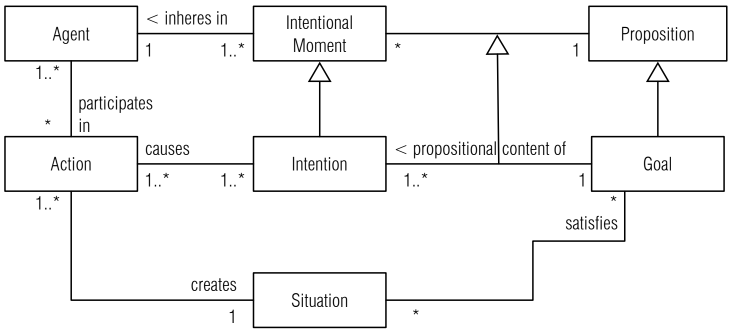


Figure 4-20 Goal satisfaction

grees for the truth value of goals, i.e., goal satisfaction levels. For instance, in a medical domain, cholesterol concentration in blood can be classified as *ideal* if the concentration is below 200, *low risk* if the concentration is between 200 and 240 and *high risk* if the combined concentration of LDL and HDL exceeds 240. In this example, an hypothetical goal of *HavingAGoodCholesterolLevel* can be considered achieved even if the cholesterol concentration stays in the range of the *low risk* level.

In GSO, the ownership relation entitles the owner agent, i.e., an agent instantiating the service client universal, to delegate the fulfillment of the goal to another agent. Moreover, by delegating a goal to an agent, the *delegatee* commits to the fulfillment of that goal. Therefore, the delegation relationship implies also a commitment between the *delegator* and the *delegatee* in relation to a goal. This delegation relationship occurs when a service client delegates the fulfillment of a goal to a service provider by means of the execution of one of its offered services. In the scope of our work we consider both open and close delegation [Guizzardi et al., 2008] of a goal. In an open delegation, a service client delegates the satisfaction of a goal to a service provider but does not prescribe any specific way of reaching this satisfaction. In other words, the service client only wants the goal to be satisfied without caring about how it is going to be satisfied. In contrast, in a close delegation the service provider should satisfy the service clients goal by means of a specific task. The distinction between open and close delegations affects how service are discovered, namely, either by searching for services whose tasks, once performed, create situations that fulfill the service client's goal, or by searching for services that perform tasks similar to the ones defined by the service client.

4.4 Service

In the design of the GSO we extended the UFO to cover the concept of *Service* and its related concepts. However, to justify the modeling choices we made, we discuss definitions of service taken from the literature, starting from the Economics and narrowing down to Computer Science and, more specifically, the definitions found in the fields of Service-Oriented Computing, Formal Ontology and Semantic Web.

When defining our concept of *Service* and its related concepts, we have analyzed the conceptual models of the current praxis in the area of Service-Oriented Computing, confronting them against ontological principles and comparing with the seminal work on service modeling in the area of Formal Ontologies [Gangemi et al., 2003, Ferrario and Guarino, 2008]. As a result, in our definitions we have tried to stay consistent with (at least) the terminology used in the SOC-related conceptual models, but disambiguating, clarifying or extending the concepts and relations, when needed.

4.4.1 Service definitions

The term service has been frequently used in Economics to represent immaterial and intangible products, while the term goods has been used to represent its material and tangible counterpart. In [Hill, 1999] the authors present an historical overview of the definitions of the concepts of services and goods, and after discussing their relevant characteristics provide a taxonomy that differentiates these two concepts, being goods either material and tangible (e.g., a computer and a chair) or immaterial and intangible (e.g., a software program and a musical composition). According to this paper, a service has one fundamental characteristic, namely the mandatory existence of relationships between producers and consumers. The authors claim that the idea of one entity (the service producer as it is called in this paper) acting for the benefit of another entity (the service consumer) is inherent to services. Moreover, contrarily to goods, services are not entities that exist independently of their producers and consumers and are defined as “some (material of immaterial) change in the condition of one economic unit produced by the activity of another unit". Although we agree with the arguments given in [Hill, 1999] we believe that their definition of service does not cover all the aspects defended by the authors, especially the explicit and mandatory relationship between the service producer and the service consumer.

4.4.2 Computer Science definitions

In Computer Science, and more specifically, in the area of Service-Oriented Computing (SOC), several (and sometimes conflicting) definition can be found in the literature. In [Preist, 2004] Preist defines service as *"the provision of something of value, in the context of some domain of application, by one party to another"*. Although we agree that a service is expected to generate value for the requester party it is not clear if the term "provision" is used in the sense of being prepared beforehand or in the sense of the act of providing something. This imprecision brings the ambiguity of two possible interpretations for the definition, namely *(i)* that a service is the actual act or process of providing something of value or, *(ii)* that a service has the potential to provide this value in the future.

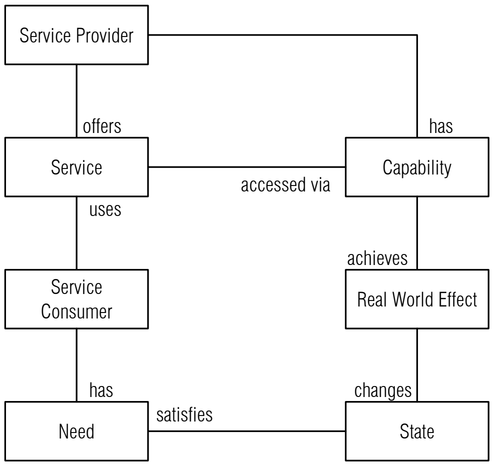
In contrast with the Economics' definitions, SOC focuses on the computational aspects of services, as in [Papazoglou and Georgakopoulus, 2003], which defines services as "self-describing, open components that support rapid, low-cost composition of distributed applications". In [Booth et al., 2004], the W3C defines that a service is *"the resource characterized by the abstract set of functionality that is provided"*. This definition implies that a service is some kind of usable artifact (a resource) that is described by the functionality it provides. Moreover, the W3C considers that different implementations and deployments of services (possibly using different programming languages) that offer the same functionality are the same service. For the W3C, a service is an abstract definition of functionality, and the concrete realization of this functionality is called an "agent". The W3C also distinguishes between the concept of service and Web service, being the later defined as *"a software system designed to support interoperable machine-to-machine interaction over a network"*. This distinction clearly separates a more abstract concept of service from the concept of Web service, which is one of the possible technological alternatives to realize a service.

4.4.3 OASIS definitions

The Organization for the Advancement of Structured Information Standards (OASIS) is a standardization body that has been working on a Reference Architecture Foundation for Service-Oriented Architecture [Laskey et al., 2009]. The reference architecture for SOA aims at identifying many of the key aspects and components present in a SOA-based system. This reference architecture and its related Reference Model for Service Oriented Architecture [Oas, 2006] define service as *"a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description"*.

Figure 4-21 presents an excerpt of the OASIS' reference architecture for SOA taken from [Laskey et al., 2009], depicting that a *Service Provider* has a capability that is accessed by means of a service. The capability is defined as *"an ability to achieve a real world effect"* [Laskey et al., 2009]. This reference architecture considers that a service provider is a stakeholder that is responsible for providing a service. It encompasses not only the enabler that exposes a capability by means of a service but also the provider of this capability, a mediator that translates or manages the relationship between service consumers and the service, or a host that offers support for the service. Since the OASIS reference architecture collapses all of these responsibilities in the concept of *Service Provider*, it is not clear in the models whether a given service provider is always responsible for all of them, none of them or a subset of these responsibilities.

Figure 4-21 The OASIS' Service concept



On the consumer's side, the OASIS' reference architecture defines that service consumers have a need, which is defined as *"a measurable requirement that a service consumer is actively seeking to satisfy"*. A need is expressed as a condition on the desired state of the world and is satisfied by this state. The *Real World Effect* is a change in the state of the world achieved by using capabilities. Therefore, the service consumer uses a service with the objective that the capability associated with this service produces a change in the world state that satisfies its need. As reproduced in the Figure 4-21, the OASIS' reference architecture does not provide cardinality for the relationships between concepts, limiting the clarity of the presented models.

OASIS also defined the Reference Ontology for Semantic Service-Oriented Architectures [Norton et al., 2008]. This ontology formalizes and extends parts of the OASIS' Reference Model [Oas, 2006], such as service description and real world effect. This ontology has been defined to support the specification of relationships among service elements, and defines the concepts of service description, goal description and capability description. Behavioral aspects are represented by the concept of behavioral model and the concepts of orchestration and choreography.

4.4.4 Semantic Web definitions

In the realm of Semantic Web, the W3C has defined an ontology for declaring and describing services named the Semantic Markup for Web Services (OWL-S) [Martin et al., 2004]. In OWL-S, service is defined as *"a Web resource which allows one to effect some action or change in the world"*. The OWL-S ontology aims at being used as a language for describing services by providing a standard vocabulary that can be used with other aspects of the OWL description language to create service descriptions. Since its objective is to provide a language to semantically annotate Web services' descriptions, OWL-S does not provide primitives for domain modeling. Moreover, the OWL-S concepts are closely related to Web services as the service realization approach.

The Web Service Modeling Ontology (WSMO) [de Bruijn et al., 2006] is another Semantic Web research initiative that provides a framework for the semantic description of goals and the functionality of Web Services. As depicted in the excerpt of WSMO presented in Figure 4-22, WSMO defines the concepts of *WSMOElement*, *WebService*, *Goal*, *Ontology* and *Mediator* as its main entities. The WSMO refers to the concepts it defines as elements. Therefore, its root concept is termed *WSMOElement*. Each WSMO element has a set of annotations attached to it. Examples of annotations that can be applied to an element include contributor, coverage, language, date, creator, description, type, version, among others.

The WSMO concept of *Ontology* refers to an artifact that defines an agreed terminology, which is used by other WSMO elements to describe relevant aspects of the domains of discourse. Ontologies are defined using the Web Service Modeling Language

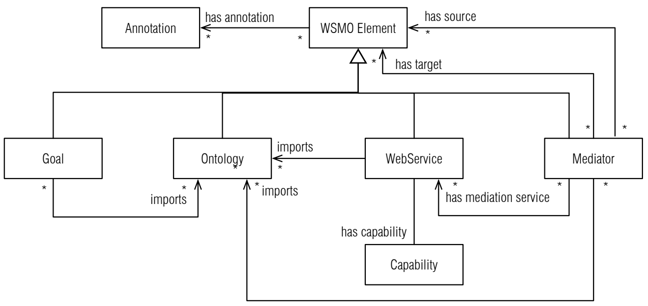


Figure 4-22 The WSMO's Service concept

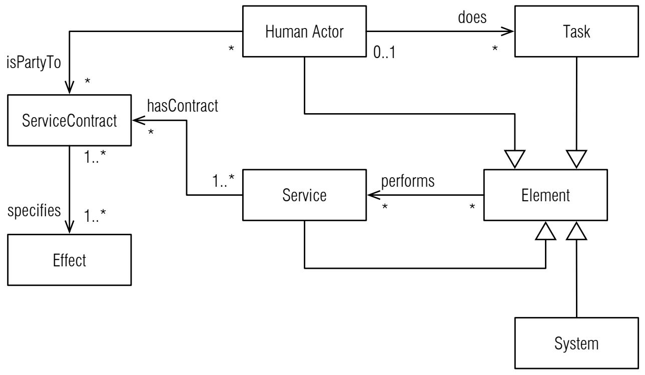
(WSML) [WSM, 2008]. The WSMO documentation is not always consistent with the terminology. For instance, in [de Bruijn et al., 2006], the authors extensively refers to the concept of *Web service*, but in the model diagram this term is replaced with *Service*. The WSMO defines that a *Web service* is *"a computational entity which is able (by invocation) to achieve a user goal"* while *Service* is *"the actual value provided by this invocation"*. Therefore, a commitment is done already in the ontological level concerning the specific technology to realize services.

As discussed in Section 2.5, a *Goal* in WSMO represents user desires. WSMO closely ties goals to Web services since the goal description model includes the interface of Web service that the user could invoke to fulfill this goal. The concept of *Mediator* describes elements designed to overcome interoperability issues between other WSMO elements. A mediator can be introduced between Web services, goals or ontologies to cope with the differences between these elements in other to guarantee their interoperability. WSMO also defines the concepts of function, instance, relation, value, and attribute of Web services, amongst others. WSMO addresses dynamic aspects by relating the concepts of *Goal* and *Interface* with the concepts of *Orchestration* and *Choreography*.

4.4.5 Open Group definitions

The Open Group has specified a Service-Oriented Architecture Ontology (SOA Ontology) [Ope, 2010] and defined service as *"a logical representation of a repeatable activity that has a specified outcome. It is self-contained and is a "black box" to its consumers"*. This definition of service is closely tied to the service behavior or its functionality, reducing the concept of service as an activity representation. Moreover, the Service-Oriented Architecture Ontology does not provide a direct connection between the actors that provide and request or consume the services with the service itself. Instead the service concept is said to be performed by the concept of *Element*. As depicted in Figure 4-23 (an excerpt of the Open Group's SOA Ontology), the concept of *Element* can be specialized into the concepts of *Task*, *System*, *Human Actor* and even by the *Service* concept itself. Although according to [Ope, 2010] a service can be performed by other elements but not by a service, the model depicted in Figure 4-23 does not prevent this situation and, therefore, allows the modeling of these undesirable situations.

Figure 4-23 The Open Group's Service concept



Another issue in the Open Group's SOA Ontology is that, although the definition of service explicitly states that a service has a specified outcome, the model links the concept of *Effect* with the concept of *ServiceContract*. The relationship between *Effect* and *Service* can thus only be established indirectly through the concept of *ServiceContract*. One would expect that an effect is a result of some activity, but the SOA Ontology does not represent this intuitive notion, leaving the concept of *Effect* isolated. In this ontology, the *Effect* concept is only linked to another concept via the *specifies* relationship with the concept of *ServiceContract*. Since the cardinality of the *specifies* relationship (0..n) indicates that a service may not have a service contract, the indirect link between service and effect may not be established, allowing some services to not have any effect.

4.4.6 Formal Ontology definitions

In [Gangemi et al., 2003] and [Ferrario and Guarino, 2008], the authors aim at providing ontological foundations for services and service science, respectively. The work in [Ferrario and Guarino, 2008] provides an initial step towards a rigorous and principled understanding of *Services Science* based on an ontological analysis.

In [Ferrario and Guarino, 2008], multi-level services are also considered, which are services at the social level that have computational services (or e-services, as called in the paper) that ultimately provide social benefits. In [Ferrario and Guarino, 2008], the definition of service is given as: "A service is present at a time t and location l iff, at time t, an agent is explicitly committed to guarantee the execution of some type of action at location l, on the occurrence of a certain triggering event, in the interest of another agent and upon prior agreement, in a certain way". We consider this ontological analysis relevant for our work as it provides a clear understanding of the service concept and the intrinsic characteristics of a service. This definition also enables the formalization of the service concept to be used in computer-readable artifacts.

4.4.7 GSO definitions

From the discussion on the service concept in Section 4.4.1 we can divide the research efforts in SOC into *(i)* proposals for reference architectures [Arsanjani et al., 2007, Laskey et al., 2009, Arsanjani et al., 2009], aiming at providing a canonical architectural pattern for SOC implementations, and *(ii)* proposals for service ontologies [de Bruijn et al., 2006, Martin et al., 2004], aiming at providing a shared conceptualization for services. A commonality among these efforts is that they consider services as computationally-bound building blocks to realize processes defined in the business level (business processes). In these approaches, the relationship between services and business processes are perceived in a limited distinction between social and computational levels. Moreover, the social level is often populated by the business processes and the computational level populated by the (Web) services. A business process (or the part of this process that is to be automated) is then mapped onto a process that can be performed by some process execution engine.

In our work, we aim at giving the concept of *Service* a broader meaning, without the commitment to a specific technology (e.g., Web services) nor even tied to a computational entity altogether. Besides the computation-related definitions, a service is sometimes referred to as an action, a type of action, or the capability to perform an action. As defined in [Ferrario and Guarino, 2008], we consider that a service encompasses a set of commitments that determine that a service provider performs a task for the benefit of a service client under certain conditions. Moreover, a service can be represented as a specialization of the concept of *Perdurant*.

To illustrate the similarities between the concept of service and the concept of perdurant, let us consider the example of the emergency of a service. At some point in time, a service provider starts designing a new service. At this moment there is only a sketch of what the service is and what it should do. As the design of the service progresses, the service provider refines the service description and targets it to a given type of potential clients. This service design process continues until the service is comprehensively specified, a target type of clients is defined and the service is made available. In this example we can consider that the service in the initial design phase has only a few elements, such as a high level definition of what tasks it should perform. As the service is refined, more elements are added, such as planned target clients, service contract and terms of use.

Assuming that the service "exists" from the moment the service provider starts thinking about it until the moment the service provider removes it from the pool of offered services, the service existence throughout these several phases is characterized by different sets of elements and properties. Therefore, similar to perdurants, a service accumulates temporal parts (functionality definition, target service clients, terms of use, contracts, etc.) and are existentially dependent of their service providers. Because of these characteristics, in GSO we chose to define the concept of *Service Universal* as a specialization of the UFO-B concept of *Perdurant* as depicted in Figure 4-24.

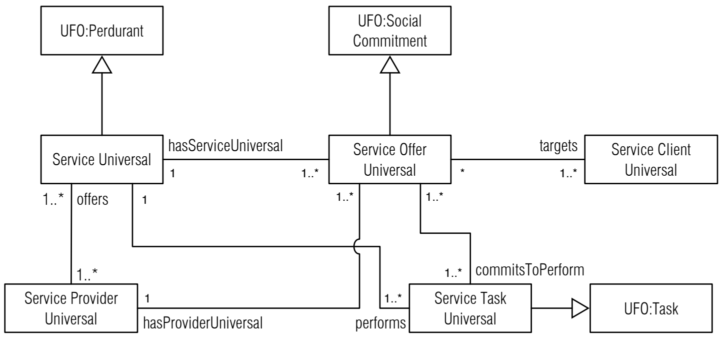


Figure 4-24 GSO's service concept

In GSO, whenever a service provider offers a service there is a social commitment represented by the concept of *Service Offer Universal* in which the service provider commits to perform a *Service Task Universal*. This service offer is targeted to a class of service clients represented by the concept of *Service Client Universal*. For instance, in the real state domain, real state agents who play the role of *Service Provider Universal*, offer the real state brokerage service for property buyers and sellers, which are two different instances of the *Service Client Universal* concept. When offering the real state brokerage service to property buyers, the real state agents are committed to perform the task (or tasks) related to supporting the service client in buying their target properties for the minimum price possible. When the real state agents offer the real state brokerage service to property sellers, they commit to perform the task (or tasks) related to supporting the service client in selling their properties for the maximum possible price.

The *Service Offer Universal* defines also a set of parameters for the service offer, such as cost, service delivery conditions, service triggering conditions, among others. Depending on the service, these parameters can be negotiable. Moreover, even when the parameters are negotiable, only a subset of the parameters may be open for negotiation. For instance, in a commercial airline transportation service, parameters such as date of departure, class and placement of the seat and type of the meal can be negotiated when contracting the service, while other parameters such as the type of aircraft, selection of crew members and flight length are usually not open for negotiation. Contrarily, in a private air charter service, parameters such as the type of aircraft and selection of the crew can be negotiated.

In the GSO, we consider a set of events that occur as part of the service provisioning. Here, we use the term provisioning not in the sense of preparation or contingency but in the sense of facilitating and delivering the service. Therefore, as depicted in Figure 4-25, provisioning a service involves offering the service, negotiating its terms of usage, activating it, triggering its execution and finally executing the task associated with the service.

Figure 4-26 shows that there is a relationship between the concepts of *Service Offer Universal*, *Service Provider Universal* and *Service Offering Universal.* This relationship represents that when a *ServiceProvider Universal* performs the action *Service Offering Universal* it creates an instance of the concept *Service Offer Universal.* Multiple service offering can be created with differences in the offering's parameters as, for instance, to target different types of service clients.

When a service is offered by the service provider, it becomes accessible to service clients. Once a service client discovers and becomes interested in a service, a negotiation over the parameters of the service offer can occur. As depicted in Figure 4-27, the *isPartyOf* relationship between the concepts of *Service Provider*

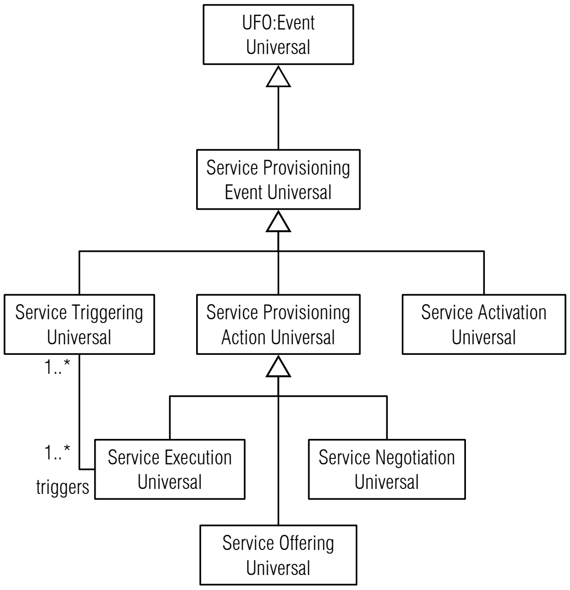

UFO:Event Universal
Service Provisioning Event Universal
Service Triggering Universal
Service Provisioning Action Universal
Service Activation Universal
1..*
1..*
triggers
Service Execution Universal
Service Negotiation Universal
Service Offering Universal

Figure 4-25
Service provisioning events

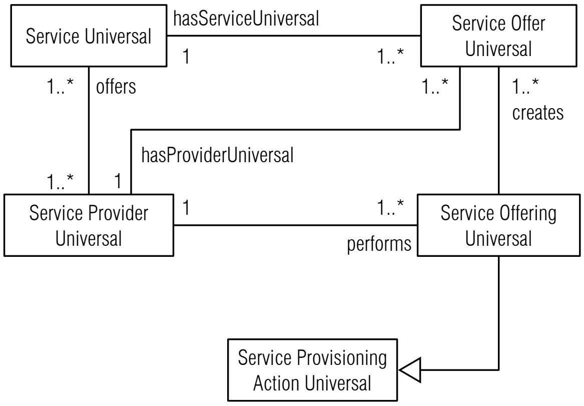


Figure 4-26
Service offering

Universal, *Service Client Universal* and *Service Negotiation Universal* represents the participation of service clients and service providers in a negotiation action, which, when successful, causes the creation of an instance of the *Service Agreement Universal*. This service agreement is a specialization of the UFO concept of *Social Relator* that represents the agreed terms of the negotiation.

The service agreement also establishes the parameters for the service activation. In some situations, a service is agreed upon but it is not available for delivery (active). For instance, in the Netherlands, when someone moves from his current address to another, this person can contract a surface mail rerouting ser-

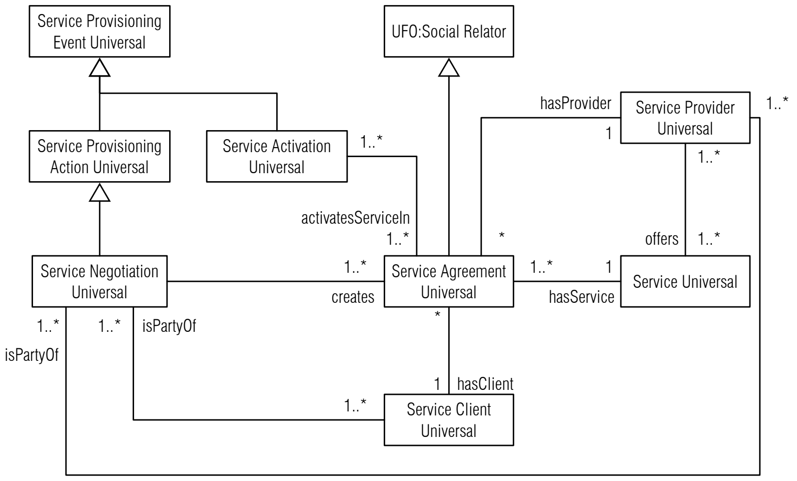

Service Provisioning Event Universal
UFO:Social Relator
Service Provisioning Action Universal
Service Activation Universal
Service Provider Universal
hasProvider
1
1..*
1..*
1..*
activatesServiceIn
1..*
*
offers
1..*
Service Negotiation Universal
1..*
creates
Service Agreement Universal
1..*
1
hasService
Service Universal
1..*
1..*
isPartyOf
isPartyOf
*
1
hasClient
1..*
Service Client Universal

Figure 4-27 Service negotiation

vice. This service reroutes the delivery of surface mail addressed to the client's old address to his new address. When setting up this service, the service client can define a starting date for the activation of the service. In this example, between the date of the service setup (or the establishment of the service agreement) and the activation date, the service is contracted but not active. After the activation date the service becomes active. However, when a service is active it is not necessarily executed or delivered. In this example, the rerouting service is only executed when the surface mail company receives surface mail addressed to the client's old address.

Some other services do not offer or require negotiation. For instance, when a person moves to a new municipality, a firefighting service becomes immediately active without any negotiation. The service agreement is implicit in the sense that when one lives in one municipality he is covered by the local fire brigade under a predefined set of parameters. In this sense, this type of agreement is similar to an adhesion contract, which is a contract between two parties that does not require negotiation.

As depicted in Figure 4-28, a service agreement specifies triggering conditions. In GSO, these triggering conditions are represented by the *Pre-Condition* concept, which is a specialization of the UFO concept of *Situation.* The *Service Triggering Event Universal* responds to these triggering conditions (when they exist), and when these conditions hold, the triggering event is raised triggering the *Service Execution Universal* action to be performed by the *Service Executor Universal.* For instance, in the surface mail rerouting example, the service triggering event occurs when the mail company receives a letter or package addressed to the service client's old address. This event triggers the execution of the surface mail rerouting service, which relabels the letter with the new address and resubmits the letter to the regular mailing delivering service.

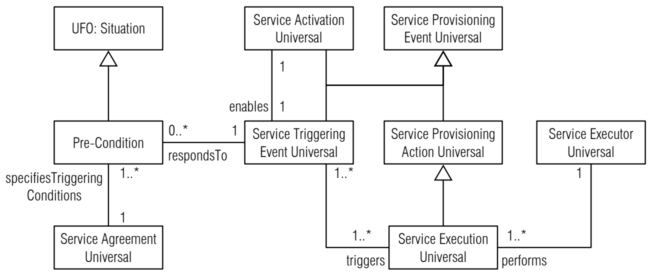

UFO: Situation
Service Activation Universal
Service Provisioning Event Universal
1
enables
1
Pre-Condition
0..*
1
respondsTo
Service Triggering Event Universal
Service Provisioning Action Universal
Service Executor Universal
specifiesTriggering Conditions
1..*
1
1..*
1
Service Agreement Universal
1..*
triggers
Service Execution Universal
1..*
performs

Figure 4-28 Service triggering and execution

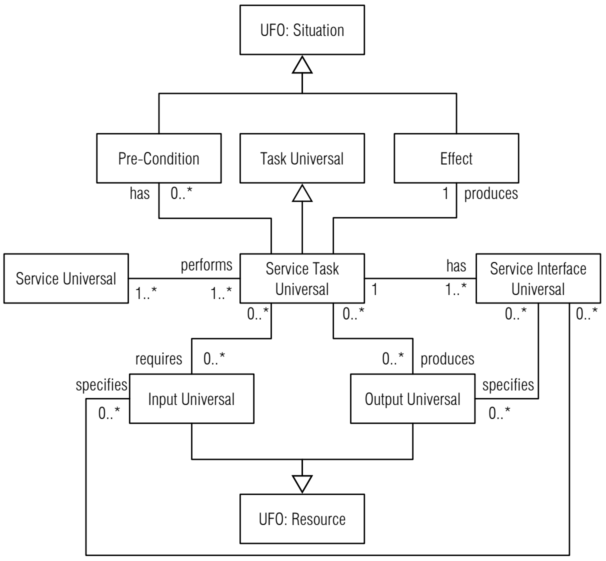
The service task performed by a service must define how it can be accessed, what it requires to be performed and what it produces after its execution. As depicted in Figure 4-29, GSO adopts the input, output, precondition and effect (IOPE) approach. The concepts of *Input Universal* and *Output Universal* represent the informational artifacts that are required for and produced by the service task execution, respectively. These input and output parameters are specified in the *Service Interface Universal.* Besides the input parameters, a service task, requires that a certain situation holds in order to have its execution enabled, namely, the *Pre-Condition* concept. Similarly, the service task execution produces a new situation termed here *Effect.* In other words, the service task execution transforms the world from the initial state defined in the *Pre-Condition* concept into a final state represented by the *Effect* concept.

4.4.8 Service description

In a grocery supermarket, products are displayed in shelves and each one has a label containing its description. Prospective buyers use this information (and the price) to decide whether to buy the product. Similarly, service providers make use of service descriptions to convey information about their offered services, such as the service content, contract requirements, activation and execution requirements, and information about the service task.

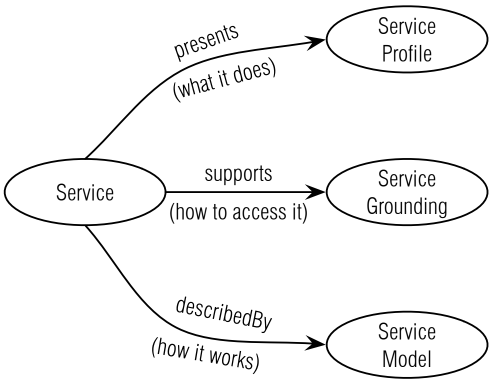
A service is not a monolithic entity, having many facets. Therefore, different types of descriptions are necessary to describe different parts of the service. Our conceptualization regarding service descriptions has been inspired by the work on OWL-S [Martin

Figure 4-29
Service
IOPE



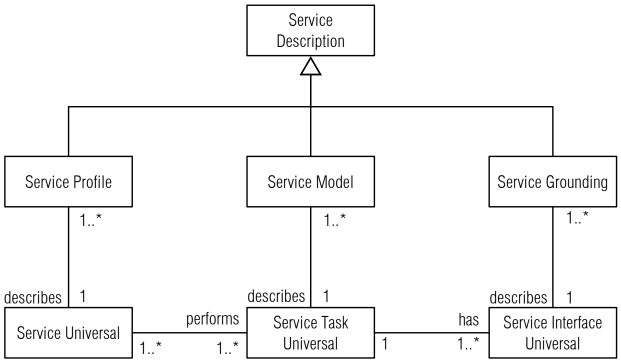
et al., 2004]. Figure 4-30 depicts the top level elements of OWL-S, in which the service profile describes what the service does, the service grounding specifies the technical details (e.g., protocol, message format, transport, addressing and serialization) of how to access the service, and the service model describes how the service works.

Figure 4-30
OWL-S'
top level
elements



Since our concept of service differs from the one in OWL-S, we have adapted the concepts of *Service Grounding*, *Service Grounding* and *Service Model* to match these differences. As depicted in Figure 4-31, the *Service Profile* provides an overview of the service for advertisement purposes. It describes what the service does (normally, in a human readable form), its requirements and conditions. Service-level agreement parameters can also be included, and used in the service negotiation.

Figure 4-31
Service description



Similarly to OWL-S, the *Service Model* in our case also describes how the service works. However, in GSO the behavior of the service is defined by the *Service Task*. Therefore, the *Service Model* describes the *Service Task*, providing information about the activities involved in the *Service Task* execution. In other words, the *Service Model* is used to inform about the service's behavior, describing the set of activities the service performs. The *Service Model* can also be used for service monitoring and orchestration. Moreover, the Service Model can be described at different granularity levels, allowing a more superficial or more in-depth view of the *Service Task*. Although one of the objectives for using services is to hide details of the internal behavior of tasks, in some situations these details interest the service client. This interest is commonly driven by security and trust concerns, such as when the service client needs to guarantee that the service task or part of it is performed by the service provider itself and not by a third-party service executor, or when the service client needs details about the service's internal behavior for auditing purposes by external regulators.

Even when more details of the service's internal behavior is needed by some service clients, the service provider can have one instance of the service model containing the necessary details to satisfy a particular client and other instances of the service model containing less details, targeted to less demanding service clients.

The *Service Interface* defines information necessary to invoke the service, i.e., to trigger service execution. In the case of computational services such as Web services, the *Service Interface* defines the technology-specific information necessary to invoke the service, namely, the communication protocol, the types of input and output parameters, network address, etc. While the *Service Interface* concept defines which functions of the service task are externally accessible and their relationships with input and output parameters (see Figure 4-29), the *Service Grounding* concept represents the informational artifact that describes the *Service Interface*, providing information about how to use the service (through its service task), including message sequence, message dependency, choreography, etc.

The GSO does not commit to any particular language for the realization of the service description's specializations, such as WSDL, OWL-S or SAWSDL. Any of these languages could be used to instantiate the concepts defined in GSO. For instance, we can have a document written in WSDL, and in the domain model where this document is referred to as a Web service description, this document can be considered as an instance of the GSO's *Service Grounding* concept.

4.4.9 Social and computational services

Among the research efforts discussed in Section 4.4.1) and Section 4.4.7 are proposals for reference architectures, such as in [Arsanjani et al., 2007, Laskey et al., 2009, Arsanjani et al., 2009], aiming at providing canonical architectural pattern for SOC implementations, and proposals for service ontologies, such as in [de Bruijn et al., 2006, Martin et al., 2004], aiming at providing a shared conceptualization for services. A commonality among these efforts is that they consider services as building blocks to realize business processes. The relationship between services and business processes are perceived in a limited distinction between social and computational levels. In these efforts, the social level is often represented by the business processes and a computational level represented by the (Web) services. The business process (or the part of it that is to be automated) is then mapped into a computer-executable process.

In our work, we claim that the distinction between services at the computational and at the social level should be further explored, so that the dichotomy between social services and computational services, and the relationships between services at each level should be made explicit and clarified. Our claim is motivated by the assumption that from now on we will be involved in an increasing number of scenarios with a complex mixture of interrelated computational and social services. With the complexity increase in such service-oriented scenarios, an explicit and clear distinction between social and computational services becomes necessary. Clarifying this distinction should enable software provisioning platforms to support not only the provisioning of computational services but also (to some degree) social services, and combinations of computational and social services.

When business processes are mapped onto computer-executable processes, the relationship between social and computational services are hard-coded. In this mapping, the computational service becomes the business process and any change on one implies in a change on the other. Contrarily, if we can manage the distinction between social and computational services and how they related to each other we would be able to establish different automation levels, supporting dynamic reconfiguration, evolution, composition, etc.

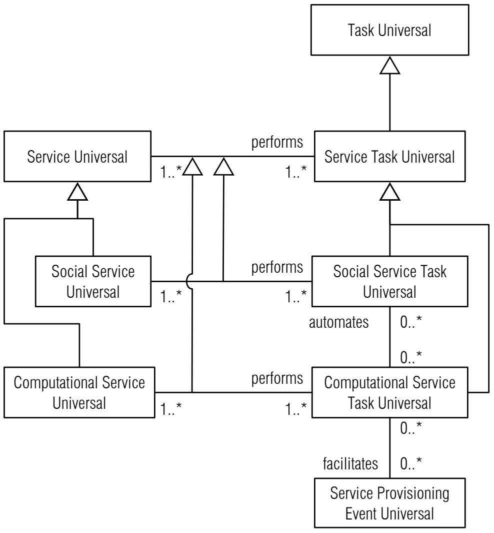
In our work, we aim at distinguishing and clarifying how social and computational services relate to each other. For this purpose, we have identified two main types of relations between social and computational services, namely, *(i)* a computational service (fully or partial) automates a social service, or *(ii)* a computational service facilitates one or more of the service provisioning steps of a social service.

The automation relation between a computational service and a social service derives from the actual automation of the task related to the social service, done by the task related to the computational service. For instance, an online hotel booking service performs a computational service task that interacts with a set of hotel-related databases and returns availability, price and booking confirmation based on given dates, location and room parameters. This computational service automates the social hotel booking service that provides equivalent results by performing its task at the social level.

For some services it may be impossible to automate their core functionality exclusively with computational services. For instance, the air transportation service that moves people or goods from one location to another cannot (yet) be automated by computational services. However, in this case some steps of the service provisioning still can be facilitated by computational services. For example, although the core functionality of the transportation service cannot be automated, some of its provisioning steps, such as discovery (e.g., through flight search services), negotiation (e.g., through online flight booking services) and activation (e.g., through online check-in services), are often facilitated by computational services.

Figure 4-32 shows a GSO excerpt that depicts the relation between computational and social services. The GSO's concept of *Service Universal* is then specialized into the concepts of *Social Service Universal* and *Computational Service Universal*. It is also possible to have a composite service composed of a mix of social and computational services. Similarly, the *Service Task Universal* is specialized into *Social Service Task Universal* or *Computational Service Task Universal*.

Figure 4-32 Social and computational services



Since the *Service Universal* performs the *Service Task Universal*, the *Social Service Universal* performs the *Social Service Task Universal* and the *Computational Service Universal* performs the *Computational Service Task Universal*. Figure 4-32 also shows the *automates* relation between the *Computational Service Task Universal* and *Social Service Task Universal*, representing the possibility that a computational service task automates a social service task. In case a social service task cannot be automated by any computational service task, still computational service tasks may facilitate one or more of the service provisioning events depicted in Figure 4-25

Chapter 5

Context-Aware Service Platform

In this chapter we present the Context-Aware Goal-Based Service Platform (CASP). This software platform aims at supporting the main classes of stakeholders identified in Chapters 3 and 4 in the activities related to service provisioning. Moreover, the CASP aims at reducing the need of direct user interaction, in particular from service clients. For this purpose, the platform contains a set of context-aware components that gather user's contextual information. The contextual information is used by the CASP to perform the discovery, selection and composition of services, identify whether service triggering conditions hold, fully or partially supply the required service input information, and invoke services.

This chapter is organized as follows: Section 5.1 motivates the proposed platform architecture design by identifying the platform's requirements under the perspective of its users; Section 5.2 identifies and discusses the patterns for interactions between the CASP and its users; these interaction patterns justify the choices made regarding the platform's architecture design presented in Section 5.3; Sections 5.4, 5.6 and 5.5 present the detailed design of the CASP's user interfaces, service provisioning and context-aware components, respectively.

5.1 Platform Users and Requirements

The stakeholders' roles supported by the Goal-Based Service Framework (GSF) are the service client, the service beneficiary, the service provider, the service executor, the context provider and the domain specialist. To fulfill one of its main objectives of facilitating service provisioning, the GSF requires a service provisioning support software platform, for which we developed the Context-

Aware Service Platform (CASP). The CASP aims at supporting the GSF's stakeholders, offering facilities to these stakeholders to carry out their activities related to service provisioning.

These stakeholder's roles are played by the CASP's users that use the platform to request and consume services (the service client and beneficiary), provide services and information (the service provider, service executor and the context provider) and to model domains in terms of domain ontologies (the domain specialist). Figure 5-1 presents the CASP (depicted as a black box) and its users.

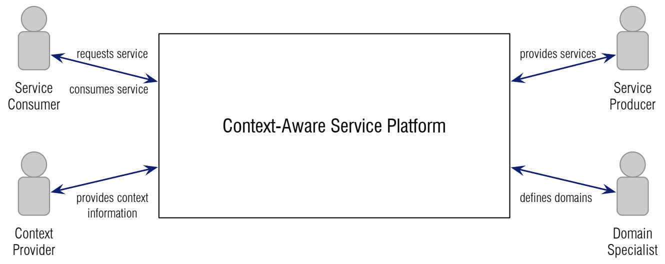


Figure 5-1
The CASP and its users

From the analysis of the stakeholder's roles behavior and their characteristics presented in Chapter 3, we have identified a set of requirements for supporting the interactions of these stakeholders through the CASP. These CASP requirements can be divided according to the four distinct functional areas based on the different users' roles depicted in Figure 5-1, namely service consumption, service provisioning, context provisioning and domain knowledge specification. The requirements, identified from the objectives of the related users within the scope of the GSF's requirements discussed in Section 3.4, are detailed as follows:

– *Service Consumption.* Service consumption is performed by the service client and service beneficiary roles, with the former being responsible for requesting services and deal with possible negotiations over the service provisioning terms, and the later being the entity that perceives the benefits of the service delivery. Service consumers' requirements follow the steps for service provisioning, namely, discovery, selection, composition, negotiation, activation, triggering and execution. Therefore, the service consumers require facilities to: *(i)* express their service requests and constraints, by means of their goals for service discovery, selection and composition; *(ii)* interact with the supporting platform to provide additional information required for service discovery, selection, composition, activation and execution, in case the CASP's context-aware components are not able to provide this required information; and *(iii)* receive the results of the service execution.

– *Service Provisioning.* Service provisioning is performed by the service provider and service executor roles, with the former being responsible for the service itself and the later being responsible for executing the activities related to the service. These users require from the supporting platform facilities to: *(i)* publish service offerings by registering service descriptions in service repositories accessible through the CASP; *(ii)* retrieve domain ontologies to semantically annotate the terms in the service descriptions in order to facilitate service provisioning; *(iii)* receive from the platform information about the fulfillment of the conditions for service activation and execution; *(iv)* receive from the platform information required for service execution (service inputs); and *(v)* provide the results of service execution.
– *Context Provisioning.* Context providers are responsible for designing and supplying contextual information about the service consumers through context sources. The context providers require from the CASP facilities to: *(i)* retrieve domain ontologies to semantically annotate the terms in the contextual information descriptions; *(ii)* register context sources and the contextual information provided by these context sources by means of contextual information descriptions; *(iii)* interact with the platform to provide the offered contextual information through the context sources.
– *Domain Knowledge Specification.* Domain specialists are responsible for specifying domain ontologies. These users require from the platform facilities to add, update and delete domain ontologies.

5.2 Platform Users Interaction Patterns

Based on the use case scenarios presented in Section 3.1 and the service provisioning requirements discussed in Section 3.4, we have sketched a set of interaction patterns between the CASP and its users. These interaction patterns determined the functional requirements of the CASP.

5.2.1 Domain specialists

The CASP operation assumes the existence of domain ontologies to describe a conceptualization of and domain and, as such, provide semantics for the terms used within these domains. Therefore, we prescribe that domain ontologies are registered to the platform by Domain Specialists. Figure 5-2 depicts an overview of the interactions between Domain Specialists and the CASP. First, a Domain Specialist must register itself to the platform (sequence 1 in Figure 5-2), receiving an unique identifier to be used later when registering (sequence 2 in Figure 5-2) and maintaining its domain ontologies (sequences 3 and 4 in Figure 5-2).

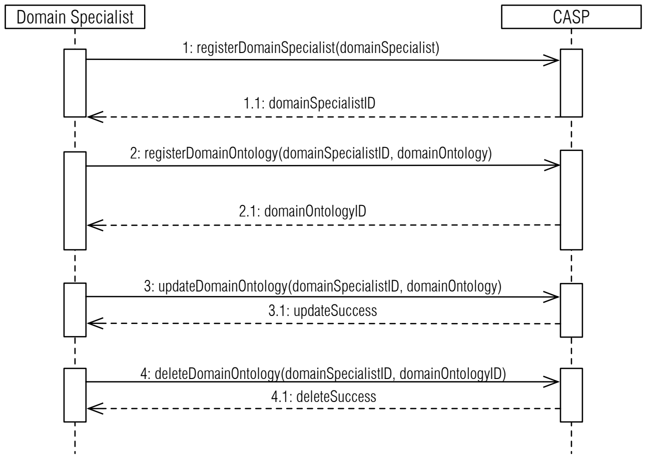

Domain Specialist
CASP
1: registerDomainSpecialist(domainSpecialist)
1.1: domainSpecialistID
2: registerDomainOntology(domainSpecialistID, domainOntology)
2.1: domainOntologyID
3: updateDomainOntology(domainSpecialistID, domainOntology)
3.1: updateSuccess
4: deleteDomainOntology(domainSpecialistID, domainOntologyID)
4.1: deleteSuccess

Figure 5-2 The domain specialist general interaction pattern

5.2.2 Service providers

Once domain ontologies are available in the platform, services can be offered so that their service descriptions are semantically annotated with these domain ontologies. Figure 5-3 depicts an overview of the interactions between Service Providers and the CASP. The Service Provider registers itself in the CASP (sequence 1 in Figure 5-3) as one of the platform's service providers to be authorized to include and maintain its services in the platform. With the received provider's identifier, the Service Provider proceeds to request a list of the available domain ontologies registered in the platform (sequence 2 in Figure 5-3). With the list of the available domain ontologies, the service provider selects the ontology that best describes the domain in which the service is intended to operate (sequence 2.2 in Figure 5-3). The selected domain ontology is then used by the provider to semantically annotate the terms in the service descriptions (sequence 2.3 in Figure 5-3). The service is registered to the platform by submitting its semantically annotated service descriptions to the platform so that the service becomes available to the CASP's service clients (sequence 2.4 in Figure 5-3). The Service Provider receives an unique identifier of the registered service from the platform (sequence 2.5 in Figure 5-3), with which the provider can update or delete the service (sequences 3 and 4 in Figure 5-3, respectively).

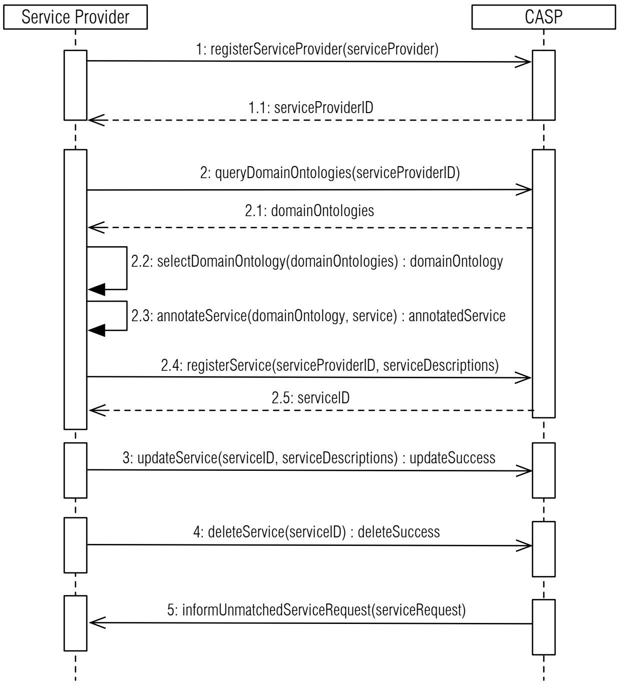

Service Provider
CASP
1: registerServiceProvider(serviceProvider)
1.1: serviceProviderID
2: queryDomainOntologies(serviceProviderID)
2.1: domainOntologies
2.2: selectDomainOntology(domainOntologies) : domainOntology
2.3: annotateService(domainOntology, service) : annotatedService
2.4: registerService(serviceProviderID, serviceDescriptions)
2.5: serviceID
3: updateService(serviceID, serviceDescriptions) : updateSuccess
4: deleteService(serviceID) : deleteSuccess
5: informUnmatchedServiceRequest(serviceRequest)

Figure 5-3
The service provider general interaction pattern

Moreover, in cases where service clients request a service and the platform is unable to discover any service that can fulfill the request, the platform can search for service providers that offer services in the service client's requested domain and inform them of the unfulfilled request (sequence 5 in the Figure 5-3). In this way the platform has a mechanism to inform service providers about service demands, allowing the service providers to offer new services.

The semantic annotation of the service descriptions may be considered as part of the service offering or part of the service design. In case a service has already been designed, one may consider that adding semantic annotations to the service descriptions is part of the activities to offer the service because the annotation is performed to make the service descriptions useful in a semantic-based software platform like the CASP. However, in the case of new services, the semantic annotation can be incorporated as a step in the service design and development process.

5.2.3 Context providers

The Context Provider interacts with the CASP aiming to provide contextual information tho facilitate the platform's service provisioning. As depicted in Figure 5-4, the Context Provider registers itself to the CASP and receives an unique identifier (sequence 1.1 in Figure 5-4). After receiving the available domain ontologies registered in the platform (sequence 2.1 in Figure 5-4), the Context Provider selects a domain ontology that properly describes the domain of interest (sequence 2.2 in Figure 5-4) and annotates the description of the contextual information it intends to register (sequence 2.3 in Figure 5-4). The annotated contextual information description is registered in the CASP (sequence 2.4 in Figure 5-4), which returns an unique identifier for the registered contextual information (sequence 2.5 in Figure 5-4).

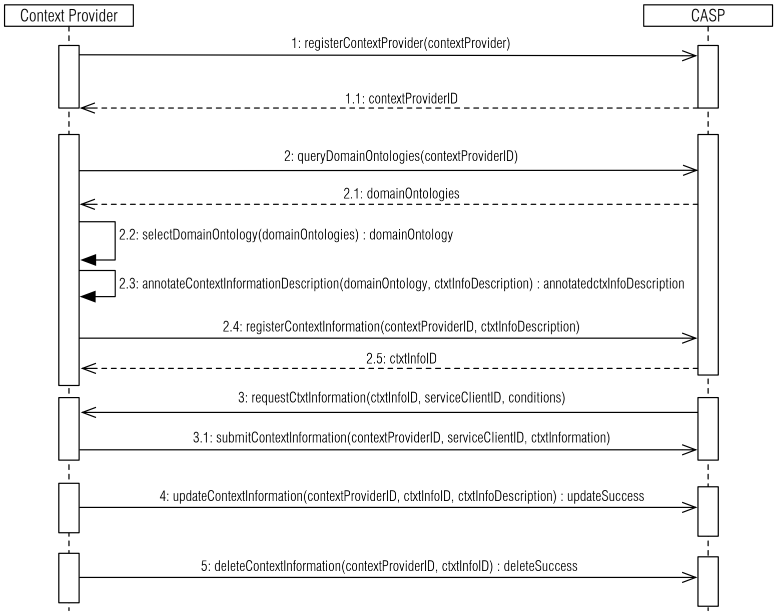

Context Provider
CASP
1: registerContextProvider(contextProvider)
1.1: contextProviderID
2: queryDomainOntologies(contextProviderID)
2.1: domainOntologies
2.2: selectDomainOntology(domainOntologies) : domainOntology
2.3: annotateContextInformationDescription(domainOntology, ctxtInfoDescription) : annotatedctxInfoDescription
2.4: registerContextInformation(contextProviderID, ctxtInfoDescription)
2.5: ctxtInfoID
3: requestCtxtInformation(ctxtInfoID, serviceClientID, conditions)
3.1: submitContextInformation(contextProviderID, serviceClientID, ctxtInformation)
4: updateContextInformation(contextProviderID, ctxtInfoID, ctxtInfoDescription) : updateSuccess
5: deleteContextInformation(contextProviderID, ctxtInfoID) : deleteSuccess

Figure 5-4 The context provider general interaction pattern

When contextual information of service clients is required, the CASP requests the information to the Context Provider (sequence 3 in Figure 5-4). In this context information request, the CASP specifies which information is required, and which service client the context information is related. Contextual information can be provided *impromptu* or through subscription.

The subscribed provisioning of contextual information occurs when the activation and/or triggering conditions of the related service are defined, and contextual information can be used to verify whether these conditions hold. In this case, the CASP subscribes with the context providers to receive updates of the contextual information that is relevant to the activation and/or triggering conditions. When the activation and/or triggering conditions can be implied from the contextual information, the service is activated or triggered. For instance, if a service client contracts a house security service to secure his home when he is traveling on holidays, we can define that the triggering conditions is the detection of an intruder. In this example, the CASP subscribes to the contextual information that informs about an intruder in the house by using motion sensors as context sources. When the sensors detect the presence of someone in the house, the CASP receives this (contextual) information, and triggers the execution of the house security service.

Improptu provisioning occurs when the service for which the contextual information is needed has an immediate execution, i.e., the service execution has been triggered immediately after the service has been discovered and selected. For instance, when a service client requests a taxi service to transport him from his current location to a given hotel, the CASP does not subscribe to any contextual information, but requests the immediate provisioning of relevant contextual information (e.g., his current location and the address of the hotel the client wants to go to). While in the case of context subscription the contextual information is commonly used for assessing whether activation and/or triggering conditions hold, in the case of *impromptu* context provisioning the contextual information is commonly used as criteria in service discovery, selection and negotiation, or used as input information for immediate execution, without any activation or triggering conditions in the future.

The Context Provider can also update the description and delete its provided contextual information (sequences 4 and 5 in Figure 5-4, respectively).

5.2.4 Service clients

With a set of available services registered in the CASP, the platform is ready to support service provisioning to service consumers.

Figure 5-5 depicts an overview of the expected interactions between Service Clients, the CASP and Service Beneficiaries in the scope of service provisioning request. The Service Client requests service provisioning to the platform by informing which goal it wants to be fulfilled through the use of a service. The CASP then tries to gather contextual information from the Service Beneficiary that can facilitate the service provisioning according to this goal. With the available contextual information, a (internal) service request is created. This service request is used to search for (discover) candidate services and, among them, select one for provisioning. After the service execution, the CASP receives the service output and forwards this output to the related Service Beneficiary.

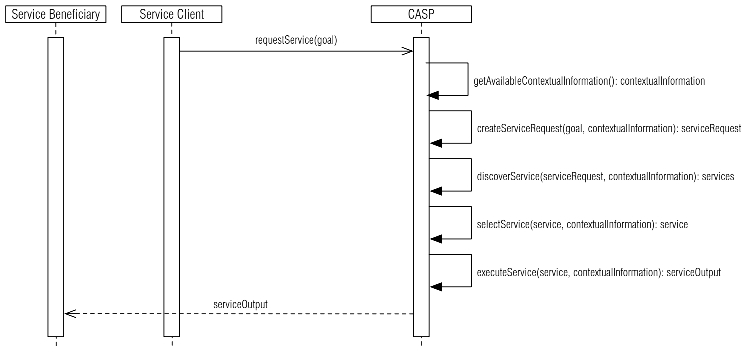

Service Beneficiary
Service Client
CASP
requestService(goal)
getAvailableContextualInformation(): contextualInformation
createServiceRequest(goal, contextualInformation): serviceRequest
discoverService(serviceRequest, contextualInformation): services
selectService(service, contextualInformation): service
executeService(service, contextualInformation): serviceOutput
serviceOutput

Figure 5-5 The service client general interaction pattern

If no service is found that fulfills the informed goal, the CASP informs the service client that no service is available for this request (not depicted in Figure 5-5). Moreover, as depicted in the sequence 5 of Figure 5-3, the platform informs service providers related to the domain of interest of the service client that a service request has been submitted and no service was found.

5.3 Architecture Overview

From the CASP's requirements in Section 5.1 and the interaction patterns between the platform and its users in Section 5.2, we have defined the CASP's architectural design as depicted in Figure 5-6 [Bonino da Silva Santos et al., 2010a]. The architectural components presented in Figure 5-6 aim at covering the requirements of each of the identified users, and have been designed to provide separation of concerns, flexibility and scalability. For this reason the CASP has been designed according to the service paradigm, in which all interactions among its constituent components take place by means of service invocations. Therefore, each component can be replaced or updated without interfering with the others, provided their exposed service interface remains supported.

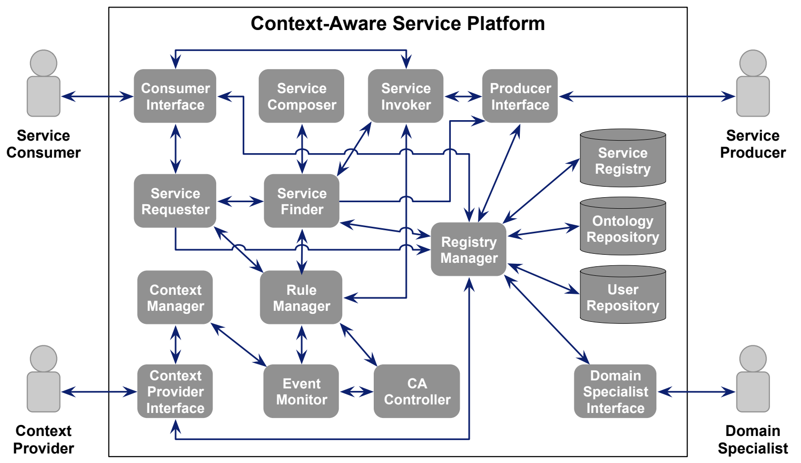


Figure 5-6 The CASP's architectural design

In the deployment of the platform and its constituent components (and services), we have used the OSGi framework [OSG, 2011], more specifically, the Knopflerfish OSGi Service Platform [Kno, 2011]. This infrastructure allows the services to be updated on the fly, and the OSGI bundle-model facilitates deployment by providing some infrastructure features such as service discovery. Since services do not need to be deployed in the same machine as the services they interact with, we can monitor the services that are consuming more computing resources and move them to other machines or use a load-balancing schema to distribute the calls to a service that demands heavy processing load from the machine.

We have separated the platform's components in three main areas, namely the users' interface components, the service provisioning components and the context-aware components, which are discussed in the sequel.

5.4 Users' Interface Components

Figure 5-6 shows that the CASP supports the interactions with its users by providing a set of interface components, namely, Consumer Interface (for Service Clients and Service Beneficiaries), Producer Interface (for Service Providers and Service Executors), Domain Specialist Interface (for Domain Specialists) and Context

Manager (for Context Providers). These interface components provide APIs that allow the user's to interact with the platform. The users interact with the platform by using either GUI applications or these APIs through their client applications.

5.4.1 Domain Specialist's Interface Components

The Domain Specialist Interface component offers interfaces to manage the registration of domain specialists and domain ontologies. Figure 5-7 depicts the Domain Specialist Interface component and its interfaces with their associated components. In Figure 5-7, the Domain Specialist Client Application is represented with a grayed fill to represent to this component is external to the CASP.

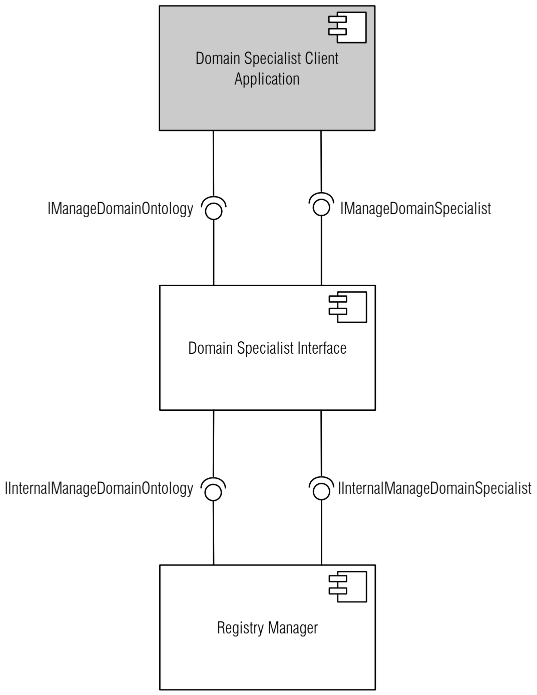
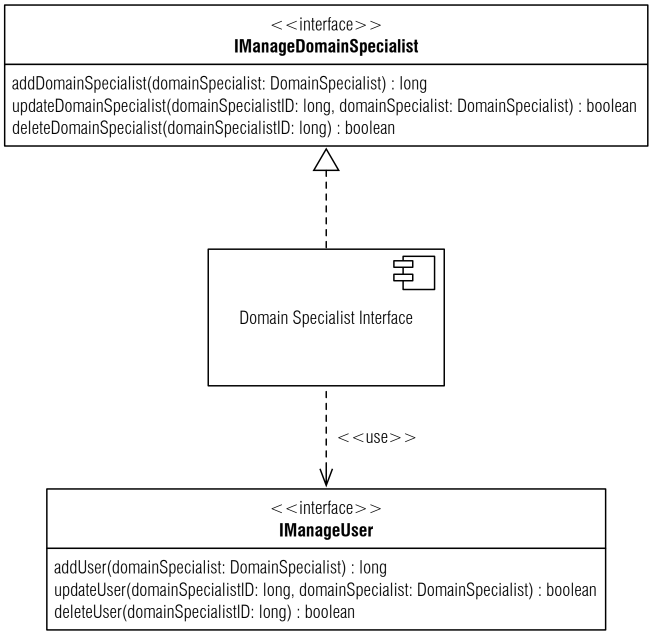

Domain Specialist Client Application
IManageDomainOntology
IManageDomainSpecialist
Domain Specialist Interface
IInternalManageDomainOntology
IInternalManageDomainSpecialist
Registry Manager

Figure 5-7 The interfaces of the Domain Specialist Interface component

The Domain Specialist Interface offers the *IManageDomain-Specialist* interface to receive requests to add, update or delete a Domain Specialist from a Domain Specialist Client Application. Once these requests are received, the Domain Specialist Interface component forwards the requests to the Registry Manager component through the *IManageUser* interface. The Registry manager component is responsible for interacting with the CASP's registries, namely the Service Registry, the Ontology Repository and the User Repository. Since the request received from the Domain Specialist Interface relates to a platform user type, namely the Domain Specialist, the Registry Manager invokes the User Repository to add, update or delete the domain specialist.

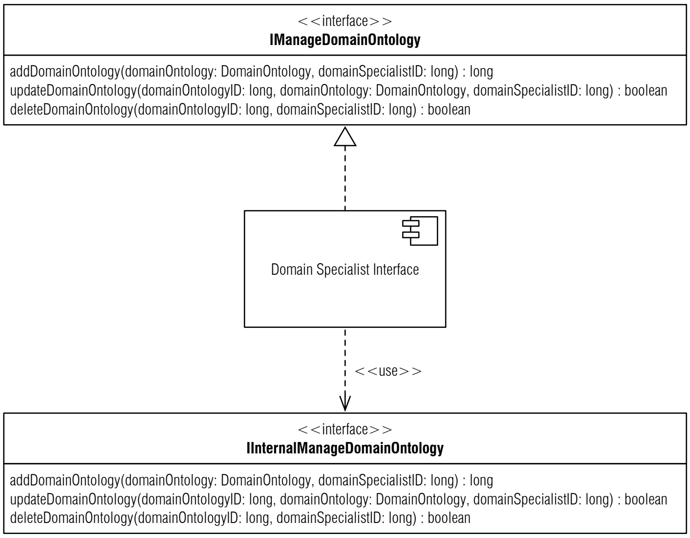
Figure 5-8 depicts the operations defined in the interfaces that are offered (*IManageDomainSpecialist*) and required (*IManageUser*) by the Domain Specialist Interface component to manage domain specialists.



<<interface>>
IManageDomainSpecialist
addDomainSpecialist(domainSpecialist: DomainSpecialist) : long
updateDomainSpecialist(domainSpecialistID: long, domainSpecialist: DomainSpecialist) : boolean
deleteDomainSpecialist(domainSpecialistID: long) : boolean
Domain Specialist Interface
<<use>>
<<interface>>
IManageUser
addUser(domainSpecialist: DomainSpecialist) : long
updateUser(domainSpecialistID: long, domainSpecialist: DomainSpecialist) : boolean
deleteUser(domainSpecialistID: long) : boolean

Figure 5-8 The Domain Specialist Interface component's interfaces to manage domain specialists

Once registered to the CASP, domain specialists can add, update and delete domain ontologies. Similarly to managing domain specialists, the operations to manage domain ontologies are received by the Domain Specialist Interface component and forwarded to the Register Manager component. However, in the case of managing domain ontologies, the Registry Manager interacts with the Ontology Repository to store, retrieve and delete domain ontologies on behalf of the domain specialist. Figure 5-9 depicts the operations of the interfaces that are offered and required by the Domain Specialist Interface component to manage domain ontologies.



<<interface>>
IManageDomainOntology
addDomainOntology(domainOntology: DomainOntology, domainSpecialistID: long) : long
updateDomainOntology(domainOntologyID: long, domainOntology: DomainOntology, domainSpecialistID: long) : boolean
deleteDomainOntology(domainOntologyID: long, domainSpecialistID: long) : boolean
Domain Specialist Interface
<<use>>
<<interface>>
IInternalManageDomainOntology
addDomainOntology(domainOntology: DomainOntology, domainSpecialistID: long) : long
updateDomainOntology(domainOntologyID: long, domainOntology: DomainOntology, domainSpecialistID: long) : boolean
deleteDomainOntology(domainOntologyID: long, domainSpecialistID: long) : boolean

Figure 5-9 The Domain Specialist Interface component's interfaces to manage domain ontologies

GDSL editor

The CASP offers a GDSL editor to help Domain Specialists define domain ontologies [Nijenhuis, 2011]. The GDSL editor aims at supporting domain specialists in defining and maintaining domain ontologies written with the GDSL. Since the GDSL is derived from the specification of the GSO, consistence between the concepts and relations of GSO, and the derived concepts and relations of the GDSL should be kept. However, we assume that the GSO can be subjected to modifications, extensions and updates and these changes should reflect in the GDSL specification.

Although a foundational ontology should be stable and, therefore, less (or never) susceptible to modifications, we assume that some adjustments and/or extensions may be eventually necessary. We consider our foundational ontology still under development, and allow modifications on it. To cope with the possibility that changes in the foundational ontology entail changes in the derived domain specification language, we required facilities for automatic generation of an updated editor for the GDSL. In order to provide the automatic generation of the GDSL editor, we have designed a transformation tool that receives a foundational ontology and generates an editor for the language derived from this foundational ontology. Figure 5-10 presents an overview of the transformation tool.

Since we used EMF/GMF [EMF, GMF] to develop our transformation tool, the generation of the language's editor from a

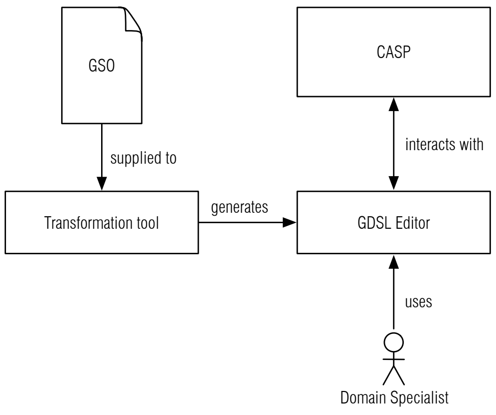
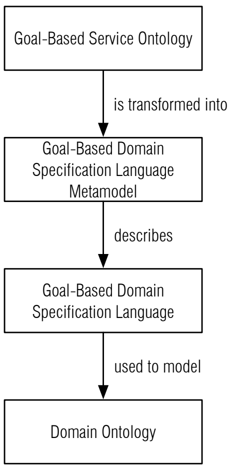


Figure 5-10 Overview of the transformation tool

foundational ontology requires some intermediate steps. As depicted in Figure 5-11, the GDSL is described by its metamodel. Therefore, it is necessary to derive the GDSL's metamodel from the GSO and derive the language from its metamodel. The GDSL can then be used to model domain ontologies.


Goal-Based Service Ontology
is transformed into
Goal-Based Domain Specification Language Metamodel
describes
Goal-Based Domain Specification Language
used to model
Domain Ontology
Figure 5-11 Ontology editor transformations from GSO to domain ontologies

An additional requirement for our transformation tool is the traceability between the constructs through the levels depicted in Figure 5-11. If the GSO is changed and the editor for the derived GDSL is generated again, the constructs specified in already existing domain ontologies (specified with previous versions of the GDSL) might be inconsistent with the newly defined language. Traceability indicates which concepts and properties of the domain ontologies correspond to which concepts and properties of the GSO. With these mappings and the previous GDSL's metamodels, the tool should be able to inform the domain specialists which of their domain ontologies have been affected by changes in the GSO and the effect these changes had in the models. With the information about the effect that the GSO changes caused on the domain ontologies, the language designer should be able to evaluate whether to maintain the changes or to revert the GSO to the previous version.

Figure 5-12 depicts the architectural design of our transformation tool. The input of the transformation tool is a foundational ontology described in some ontology language. This foundational ontology is received by the Translator component, which transforms the ontology into a language metamodel using a metamodel language such as Ecore [EMF] or MOF [MOF]. The foundational ontology-to-metamodel transformation follows transformation rules, which specifies the mappings between the constructs of the language used to describe the foundational ontology and the constructs of the language used to describe the metamodel to be derived from the foundational ontology. For instance, in our prototype the foundational ontology has been described in OWL and the resulting metamodel is to be described in Ecore. Therefore, we required an OWL-to-Ecore transformation rule to guide the transformation.

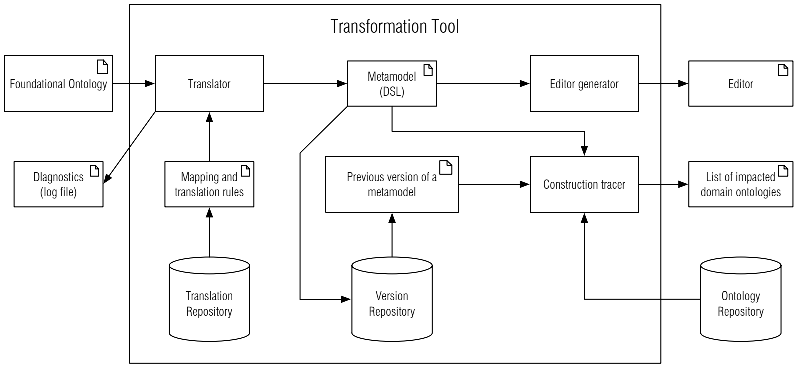


Figure 5-12 The architectural design of the transformation tool

In our Transformation Tool, we allow the generation of metamodels described in different metamodel languages from foundational ontologies written in different ontology description languages. To support this feature, we have designed the Translation Repository component, which stores transformation rules and mappings from and to different ontology and metamodel languages, respectively.

After receiving the foundational ontology and the related transformation rules, the Translator generates a metamodel for the domain specification language. This metamodel is then forwarded to the Editor Generator component, which is responsible for generating an editor for the language specified in the received metamodel. Every generated metamodel is stored in the Version Repository component. Versioning of the generated metamodels is performed when a language designer updates a source foundational ontology. When this situation occurs, the transformation tool retrieves the previous version of the related language's metamodel and sends it together with the newly generated metamodel to the Construction Tracer component. This component requests from the Ontology Repository a list of domain ontologies defined with the domain specification language described by the received metamodels.

The Construction Tracer proceeds to determine the differences between the two versions of the metamodels. The component then analyzes the domain ontologies to identify whether these domain ontologies have been affected by the changes in the specification language. The Construction Tracer generates a list of affected domain ontologies that can be used by domain specialists and language designers. The language designer can assess whether its modifications on the language caused undesired effects, while the domain specialists are warned about the issues and are able to update their affected domain ontologies to comply with the new features of the domain specification language. The internal structure of the Construct Tracer component is depicted in Figure 5-13.

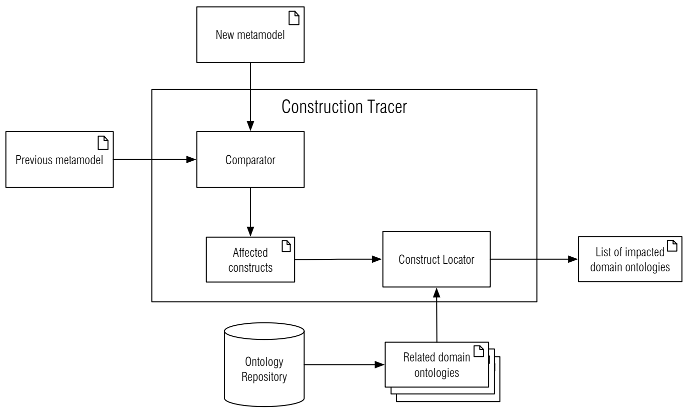
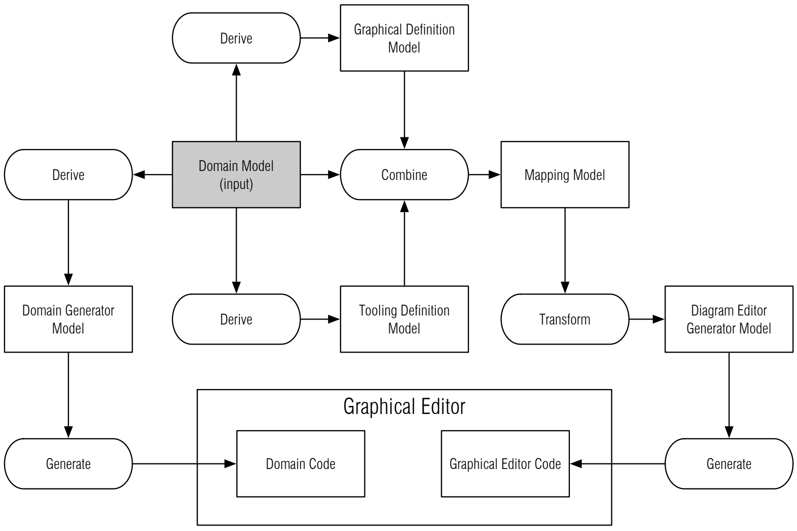


Figure 5-13 Detailed view of the Construction Tracer component

Our prototype implementation of the transformation tool has been developed as an Eclipse plug-in. This plug-in utilizes the Eclipse's EMF [EMF] and GMF [GMF] platforms to generate the GDSL editor. The choice for the Eclipse set of tools has been due to the availability of the tools, documentation and examples as well as for the maturity of the technologies. Since EMF and GMF require the input metamodel to be written in Ecore [EMF], we have used Ecore as the metamodel language in our prototype. Due to availability of tools, documentation and popularity, we have also chosen OWL [OWL, 2009] as the ontology description language. Therefore, we have used an OWL-to-Ecore transformation tool named EMF4SW [Gronback, 2009]. The EMF4SW is available as an Eclipse plug-in, which currently translates from OWL to Ecore and vice-versa, from OWL to UML and vice-versa, and from EMF to RDF and vice-versa.

Figure 5-14 presents a more detailed view of the operation of the Editor Generator. This operation follows the requirements and steps of the EMF/GMF platforms. The metamodel that the Editor Generator receives from the Translator component is considered as a domain model in the EMF/GMF terminology. This domain model is used to generate several other artifacts, such as the Domain Generator Model, the Graphical Definition Model and the Tooling Definition Model. The Domain Generator Model is used to generate the Domain Code, which provides the domain's primitives and a tree-based editor. The Graphical Definition Model defines the graphical elements that will be used as visual modeling elements in the editor. The Tooling Definition Model specifies which tools can be used in the editor.

Figure 5-14
Sequence of interaction for the GDSL editor generation



The Tooling Definition Model, the Graphical Definition Model and the Domain Model (the language's metamodel) are combined to generate a Mapping Model. The Mapping Model specifies the mappings between the elements of the three source models. The Mapping Model is then transformed into a Diagram Editor Generator Model, which is used to generate the graphical editor's code. The Graphical Editor Code and the Domain Code together form the Graphical Editor.

The steps from the Domain Model, the Graphical Definition Model and the Tooling Definition Model to the Mapping Model involve a set of decisions, such as which constructs in the metamodel should become links, and which should become nodes in the resulting graphical editor. An automatic approach can be used to identify these links and nodes based on names or language constructs. This approach is feasible if we consider a small and pre-defined set of source ontology languages. However, since we require the transformation tool to be language-agnostic and to be used for multiple ontology languages and the formalisms behind them, a fully automatic approach could not be used in our prototype. We have adopted a semi-automatic approach where, as much as possible, the decisions are taken automatically, but leaving some specific choices to the language designer. Examples of these choices are: validation of the foundational ontology-to-metamodel translation, definition of the language elements that are represented as nodes or links, and definition of which graphical shapes represent which language elements.

Figure 5-15 depicts the sequence of interactions between the language designer and the transformation tool to generate a graphical editor. The language designer submits to the transformation tool the foundational ontology from which he intends to derive a domain modeling language editor (in the scope of this thesis the foundational ontology is the GSO). The transformation tool forwards the GSO specification to the Translator component, which translates the received ontology specification into a metamodel for the GDSL. The Translator returns the generated metamodel together with a log file containing details about the translation, such as the specific mappings and translations used according to the rules retrieved from the Translation Repository. The translation's log file is forwarded to the language designer for validation.

If the language designer finds invalid translations in the translation's log file, it implies that the transformation rules are not correct. In this case, the translation should be canceled. Since the generated metamodel is not fully compliant with the foundational ontology, models created with the modeling language defined by the generated metamodel do not properly represent the conceptualization described in the foundational ontology, and should not

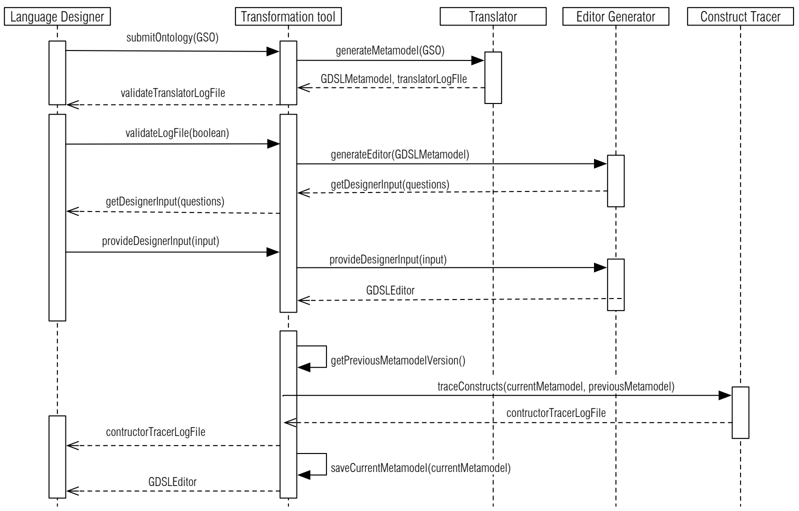

Language Designer
Transformation tool
Translator
Editor Generator
Construct Tracer
submitOntology(GSO)
generateMetamodel(GSO)
GDSLMetamodel, translatorLogFIle
validateTranslatorLogFile
validateLogFile(boolean)
generateEditor(GDSLMetamodel)
getDesignerInput(questions)
getDesignerInput(questions)
provideDesignerInput(input)
provideDesignerInput(input)
GDSLEditor
getPreviousMetamodelVersion()
traceConstructs(currentMetamodel, previousMetamodel)
contructorTracerLogFile
contructorTracerLogFile
saveCurrentMetamodel(currentMetamodel)
GDSLEditor

Figure 5-15 Detailed view of the editor generation

be used. To tackle this issue, the transformation rules should be analyzed, and the incorrect transformation should be fixed.

With the validation of the translation from the foundational ontology to the metamodel, the transformation tool starts the editor generation by invoking the Editor Generator component and supplying the GDSL metamodel. During the editor generation, some decisions should be taken regarding details of the editor, such as which concepts should be mapped to visual elements representing nodes and which concepts should be mapped to elements representing links. After receiving these decisions from the language designer, the Editor Generator can finish the generation of the GDSL editor. The transformation tool then retrieves the previous version of the language's metamodel (if available) and sends it together with the GDSL's current metamodel to the Construct Tracer component.

The Construct Tracer searches for the effects of the changes in the new version of the language on the existing domain ontologies modeled with the earlier version of the language. If any effects are found, the Construct Tracer sends a log file containing details of these effects to the language designer for evaluation. If the language designer decides to maintain the changes in the language, the new version of the GDSL's metamodel is stored in the Version Repository and the GDSL's editor is made available for the domain specialists.

5.4.2 Service Producers' Interface Components

The Producer Interface component offers interfaces to manage the registration of service providers and service executors, to retrieve domain ontologies to be used on semantic annotation of service descriptions, to manage the registration of service descriptions, and to request the execution of services. Figure 5-16 depicts the Producer Interface component and its interfaces with their associated components. In Figure 5-16, the Service Provider Client Application is represented with a grayed fill to represent to this component is external to the CASP.

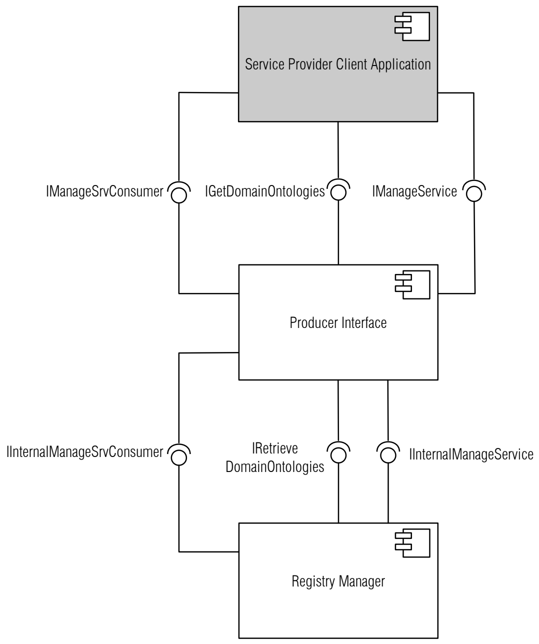
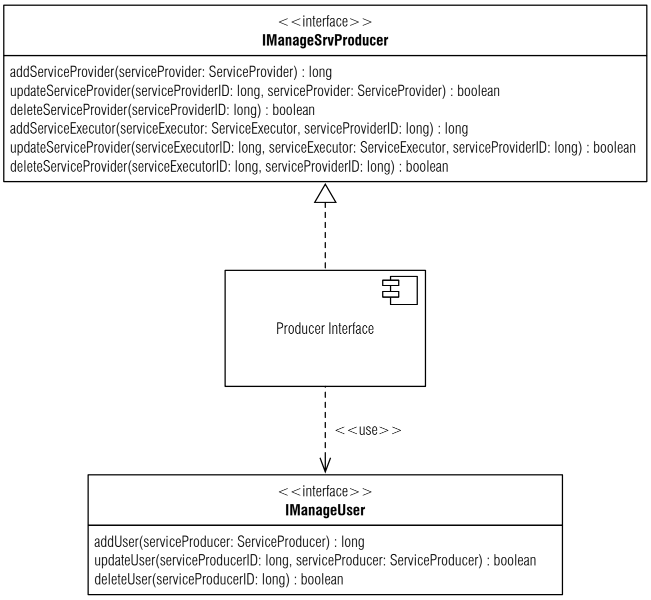


Figure 5-16 The interfaces of the Producer Interface component

The Producer Interface component offers the *IManageSrvProducer* interface for the management of the registration of service providers and service executors by service providers. The service provider interacts with the CASP through a Service Provider Client Application. As depicted in Figure 5-17, this application uses the *addServiceProvider* operation of the *IManageSrvProducer* interface offered by the Producer Interface to request registration as one of the CASP's service providers. The Producer Interface receives the service provider's registration request, and forwards this request to the Registry Manager by calling the *addUser* method defined in its *IManageUser* interface, which stores the provider's information in the User Repository. Similarly, when the Service

Provider needs to update its registration information or remove itself from the CASP, the Producer Interface component is used and forwards the requests accordingly to the Registry Manager. Service executors are added, updated and deleted in the same way by service providers.


<<interface>>
IManageSrvProducer
addServiceProvider(serviceProvider: ServiceProvider) : long
updateServiceProvider(serviceProviderID: long, serviceProvider: ServiceProvider) : boolean
deleteServiceProvider(serviceProviderID: long) : boolean
addServiceExecutor(serviceExecutor: ServiceExecutor, serviceProviderID: long) : long
updateServiceProvider(serviceExecutorID: long, serviceExecutor: ServiceExecutor, serviceProviderID: long) : boolean
deleteServiceProvider(serviceExecutorID: long, serviceProviderID: long) : boolean
Producer Interface
<<use>>
<<interface>>
IManageUser
addUser(serviceProducer: ServiceProducer) : long
updateUser(serviceProducerID: long, serviceProducer: ServiceProducer) : boolean
deleteUser(serviceProducerID: long) : boolean
Figure 5-17 The Producer Interface component's interfaces to manage service producers' registrations

In case the service provider has one or more activated services, the CASP warns the service provider that the contracts related to these services must be terminated before proceeding with the provider's removal. After the services' contracts have been terminated, the services are deactivated and removed, and only then the service provider can be removed from the platform. If a service executor is to be removed, and the user is assigned as the executor of any activated service, the related service provider is contacted to assign another executor to the service before the former executor can be removed.

The Producer Interface component's *IGetDomainOntologies* interface offers methods to retrieve available domain ontologies to be used to semantically annotate the service descriptions. As depicted in Figure 5-18, the *IGetDomainOntologies* interface defines the *getDomainOntologies* operation to retrieve the list of domain ontologies registered to the CASP, and the *getDomainOntology* operation to retrieve a specific domain ontology.

After retrieving the list of available domain ontologies, the service provider selects the one that is appropriate to semanti-

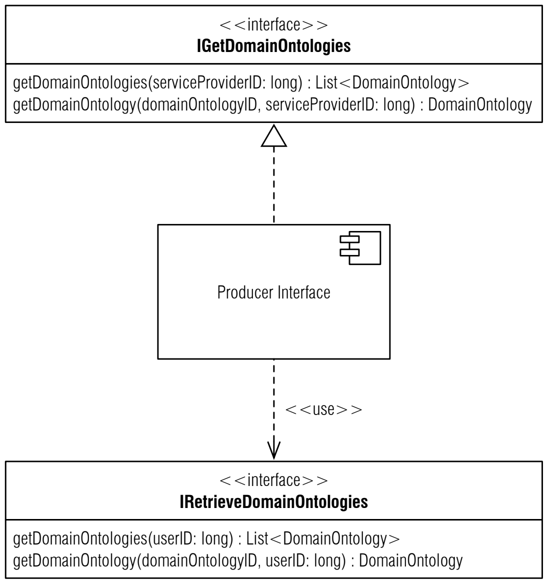
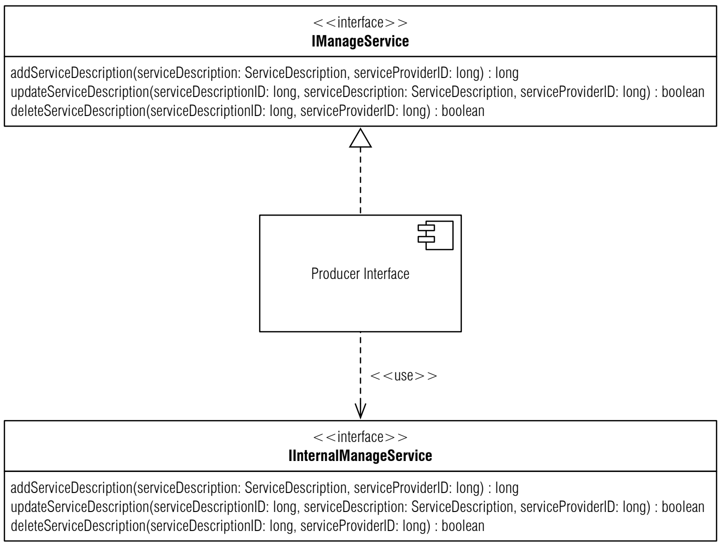

<<interface>>
IGetDomainOntologies
getDomainOntologies(serviceProviderID: long) : List<DomainOntology>
getDomainOntology(domainOntologyID, serviceProviderID: long) : DomainOntology
Producer Interface
<<use>>
<<interface>>
IRetrieveDomainOntologies
getDomainOntologies(userID: long) : List<DomainOntology>
getDomainOntology(domainOntologyID, userID: long) : DomainOntology

Figure 5-18 The Producer Interface component's interface to retrieve domain ontologies

cally annotate the terms in the service's descriptions. As depicted in Figure 5-19, the Service Provider Client Application uses the *addServiceDescription* operation defined in the Producer Interface component's *IManageService* interface to add the semantically-annotated service description as one of the platform's available services. The Producer Interface forwards the service registration request to the appropriate operation defined in the Registry Manager's *IInternalManageService* interface, which stores the service in the Service Registry and returns an unique identifier of the registered service. Similarly, the Service Provider Client Application calls the Producer Interface's *updateServiceDescription* and *deleteServiceDescription* operations to update or delete the service descriptions, respectively.

Figure 5-20 depicts the Producer Interface component's API, exposing interfaces to request service execution and to inform unfulfilled service demands. When the service triggering conditions are met, the Service Invoker component requests the service execution to the Producer Interface component. The Producer Interface then interacts with the proper Service Executor to request the execution of the service by invoking the service. In this event, the Service Executor receives also the information to be used as input parameters for the associated service task. After the service execution, the Service Executor returns to the Producer Interface the generated output information (if available), which is for-



<<interface>>
IManageService
addServiceDescription(serviceDescription: ServiceDescription, serviceProviderID: long) : long
updateServiceDescription(serviceDescriptionID: long, serviceDescription: ServiceDescription, serviceProviderID: long) : boolean
deleteServiceDescription(serviceDescriptionID: long, serviceProviderID: long) : boolean
Producer Interface
<<use>>
<<interface>>
IInternalManageService
addServiceDescription(serviceDescription: ServiceDescription, serviceProviderID: long) : long
updateServiceDescription(serviceDescriptionID: long, serviceDescription: ServiceDescription, serviceProviderID: long) : boolean
deleteServiceDescription(serviceDescriptionID: long, serviceProviderID: long) : boolean

Figure 5-19 The Producer Interface component's interface to manage service descriptions' registrations

warded back to the Service Invoker to be later sent to the service consumer.

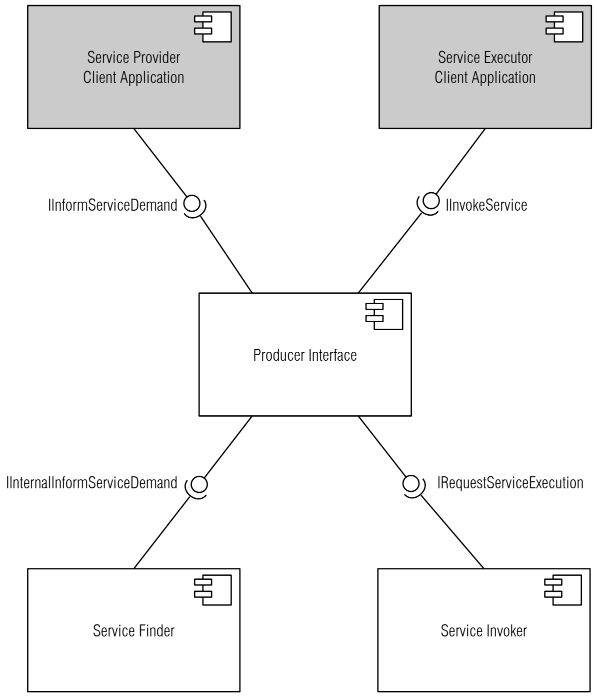


Figure 5-20 The Producer Interface to request service execution and inform of unfulfilled service demands

Moreover, when the CASP receives a service request and the Service Finder component does not find any service to fulfill this request, the Service Finder sends the information about an unfulfilled service demand to the Producer Interface component. The Producer Interface component forwards the information about an unfulfilled service demand to service providers that provide services in the domain in which a service is being requested.

In our prototype implementation, the service descriptions has been written using the W3C's Semantic Annotations for WSDL and XML Schema (SAWSDL) [Farrell and Lausen, 2007]. The SAWSDL defines how to add semantic annotations to service descriptions written in WSDL and allows the existing WSDL documents to be semantic annotated. The semantic annotations refer to concepts in the domain ontologies, and are applied to various parts of the WSDL document, such as input and output message structures, interfaces and operations.

For instance, Listing 5.1 shows a fragment of the service description of a service to set medical appointments. One of the operations defined by this service is named *checkAvailability*, which has one input message named *checkAvailabilityRequest* and one output message name *checkAvailabilityResponse*. The *checkAvailabilityRequest* message is defined as having the complex type *Appointment*. This complex type represents a medical appointment and consists of the physician designated for the appointment, and the date and time of the appointment. In this example, we have defined that the *physician* element of the complex type *Appointment* refers to the concept *Physician* defined in our health ontology.

Listing 5.1 Example of semantic annotations in a SAWSDL document

```
...
<xsd:complexType name="Appointment">
  <xsd:all>
  <xsd:element name="physician" type="xsd:string"
    sawsdl:modelReference="http://www.utwente.nl/ ↩
       ↪ trese/lolavo/healthontology#Physician"/>
    <xsd:element name="date" type="xsd:date"/>
    <xsd:element name="time" type="xsd:time"/>
  </xsd:all>
</xsd:complexType>
...
<wsdl:message name="checkAvailabilityRequest">
  <wsdl:part name="tentativeAppointment" ↩
      ↪ type="tns:Appointment"/>
</wsdl:message>
<wsdl:message name="checkAvailabilityResponse">
  <wsdl:part name="dateTimeAvailability" ↩
      ↪ type="xsd:boolean"></wsdl:part>
</wsdl:message>
```

```
...
<wsdl:operation name="checkAvailability">
  <wsdl:input ↩
     ↪ message="tns:checkAvailabilityRequest"/>
  <wsdl:output ↩
     ↪ message="tns:checkAvailabilityResponse"/>
</wsdl:operation>
...
```

5.4.3 Context Provider's Interface Components

The Context Provider Interface component offers interfaces to manage the registration of the Context Providers, to manage the registration of the contextual information they offer to provide, to receive contextual information from Context Providers, and to retrieve available domain ontologies, which are used to semantically annotate the contextual information descriptions. Figure 5-21 depicts the Context Provider Interface component, and its interfaces with their associated components.

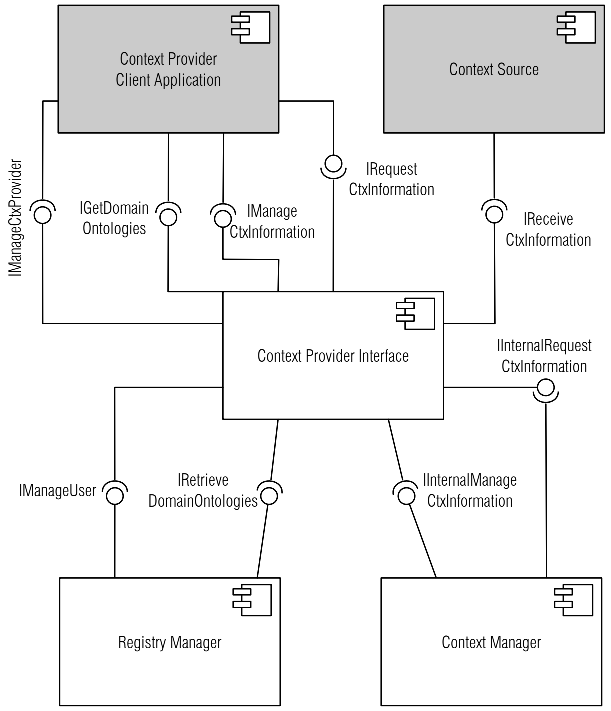
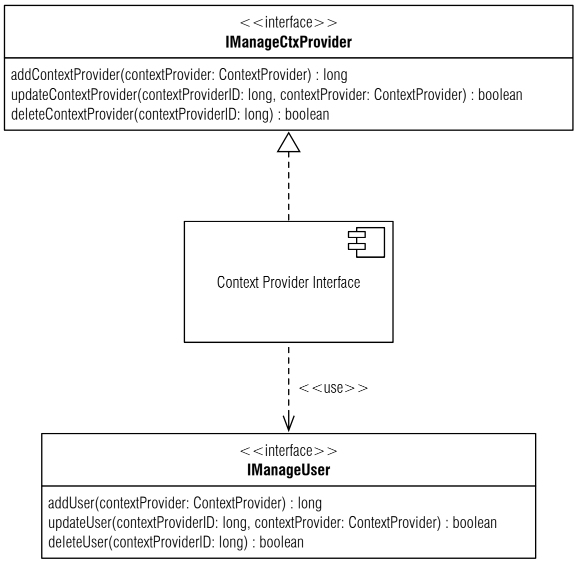


Figure 5-21 The interfaces of the Context Provider Interface component

The Context Provider interacts with the CASP through its Context Provider Client Application, which can request the registration, update and deletion of the Context Provider. These requests are received by the Context Provider Interface component through its *IManageCtxProvider* interface, and forwarded to the Registry Manager through its *IManageUser* interface, which is responsible for performing the requested operations on the User Repository. Figure 5-22 depicts the interfaces offered and used by the Context Provider Interface component to manage the registration of context providers.

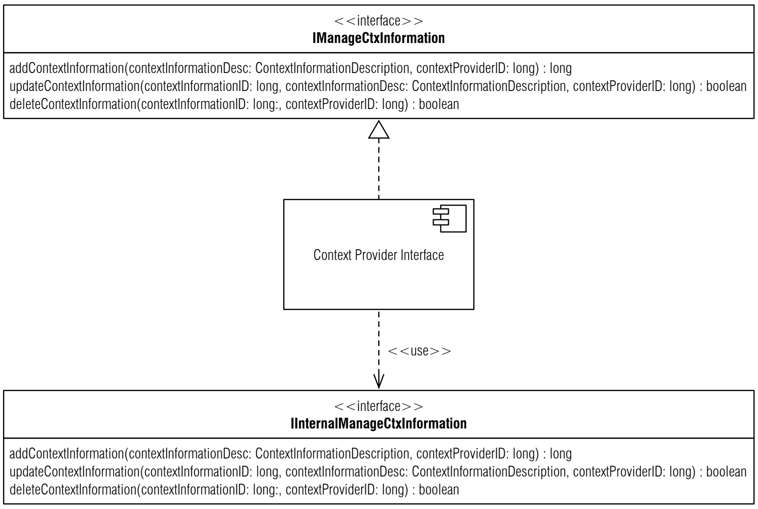


<<interface>>
IManageCtxProvider
addContextProvider(contextProvider: ContextProvider) : long
updateContextProvider(contextProviderID: long, contextProvider: ContextProvider) : boolean
deleteContextProvider(contextProviderID: long) : boolean
Context Provider Interface
<<use>>
<<interface>>
IManageUser
addUser(contextProvider: ContextProvider) : long
updateUser(contextProviderID: long, contextProvider: ContextProvider) : boolean
deleteUser(contextProviderID: long) : boolean

Figure 5-22 The interface of the Context Provider Interface component to manage context provider's registrations

The Context Provider Interface component offers the *IGetDomainOntologies* interface to allow context providers request a list of the available domain ontologies. This interface is similar to the Producer Interface component's interface depicted in Figure 5-18. The registered Context Provider requests to the Context Provider Interface component a list of the available domain ontologies through its Context Provider Client Application. The Context Provider Interface component then forwards the request to the Registry Manager, which retrieves a list of available domain ontologies. The context provider selects an appropriate domain ontology, and semantically annotates the context information description with the concepts from the selected domain ontology.

After the context information description is semantically annotated, the context provider requests the registration of the context information description through its Context Provider Client Application. As depicted in Figure 5-23, this request is performed through the *addContextInformation* operation defined in the *IManageCtxInformation* interface, offered by the Context Provider Interface component, and forwarded to the Context Manager component through the appropriate operation defined in its *IInternalManageCtxInformation* interface. Similarly, requests to update and delete registered context information descriptions are forwarded to the Context Manager component by the Context Provider Interface.



<<interface>>
IManageCtxInformation
addContextInformation(contextInformationDesc: ContextInformationDescription, contextProviderID: long) : long
updateContextInformation(contextInformationID: long, contextInformationDesc: ContextInformationDescription, contextProviderID: long) : boolean
deleteContextInformation(contextInformationID: long:, contextProviderID: long) : boolean
Context Provider Interface
<<use>>
<<interface>>
IInternalManageCtxInformation
addContextInformation(contextInformationDesc: ContextInformationDescription, contextProviderID: long) : long
updateContextInformation(contextInformationID: long, contextInformationDesc: ContextInformationDescription, contextProviderID: long) : boolean
deleteContextInformation(contextInformationID: long:, contextProviderID: long) : boolean

Figure 5-23 The interface of the Context Provider Interface component to manage registrations of contextual information descriptions

Since context information descriptions are only used by the context-aware components and not by any other of the CASP's components, they are stored within the context-aware components and not in a repository accessible by the Register Manager. Although we risk to duplicate storage responsibility, we have made this design choice to allow the substitution of the context-aware components, and consequently, the method they use to store and manage context information, without affecting the remaining of the CASP's components.

Moreover, the deletion of Context Providers or their offered context information does not affect the current service contracts, activation or execution. If one of these service-related activities relied on deleted contextual information, the context-aware components try to search for other providers of that information. In case no other provider for the requested information is found, the provisioning of the information is requested to the Service Beneficiary.

Figure 5-24 depicts the interfaces of the Context Provider Interface component to request and receive contextual information. When one of the CASP's service provisioning components (detailed later in Section 5.5) requires contextual information, the

Context Manager component uses the *IInternalRequestCtxInformation* interface of the Context Provider Interface component to request for a specific type of contextual information. The Context Provider Interface component then forwards the request to the appropriate Service Provider Client Application through the *IRequestCtxInformation* interface, offered by the client applications. The contextual information is returned to the CASP not directly by the Context Provider Client Application, but by the associated Context Source. The Context Source uses the *IReceiveCtxInformation* interface offered by the Context Provider Interface component to submit the requested contextual information.

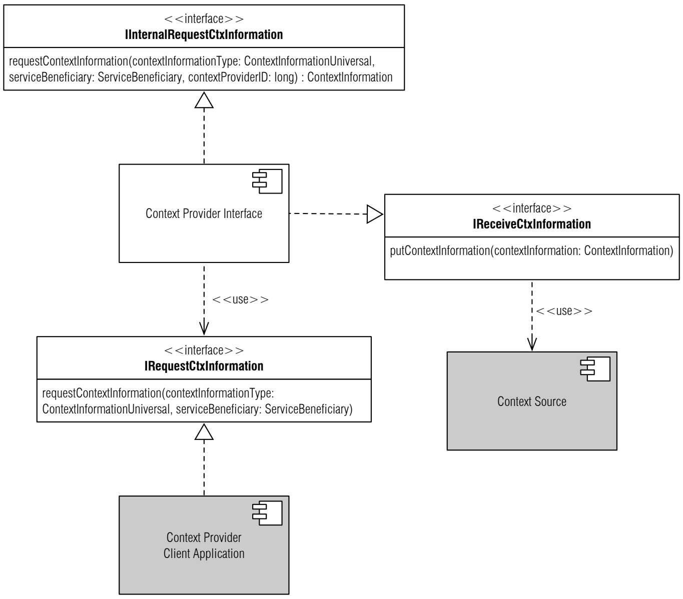

<<interface>>
IInternalRequestCtxInformation
requestContextInformation(contextInformationType: ContextInformationUniversal, serviceBeneficiary: ServiceBeneficiary, contextProviderID: long) : ContextInformation
Context Provider Interface
<<interface>>
IReceiveCtxInformation
putContextInformation(contextInformation: ContextInformation)
<<use>>
<<use>>
<<interface>>
IRequestCtxInformation
requestContextInformation(contextInformationType: ContextInformationUniversal, serviceBeneficiary: ServiceBeneficiary)
Context Source
Context Provider Client Application

Figure 5-24 The interfaces of the Context Provider Interface component to request and receive contextual information

Here, we distinguish the Context Provider, the agent responsible for offering contextual information, from the Context Source, the entity that actually generates the contextual information. Context Sources can be sensors or other information generation facility such as software applications, services, etc., and a Context Provider can offer many types of context information, which are generated by Context Sources. For instance, a meteorological agency (the Context Provider) provides meteorological information for a given location, which can be used as contextual information. This contextual information is generated through a set of temperature, wind, precipitation and atmospheric pressure sensors (the Context Sources).

5.4.4 Service Consumers's Interface Components

The Consumer Interface component's API offers interfaces and operations to manage the registration of service consumers, to allow Service Clients to submit their service requests, to return the results of the service execution to Service Beneficiaries and to request information required by the services that could not have been gathered by the platform's context-aware components. Figure 5-25 depicts the Consumer Interface component and its interfaces with their associated components.

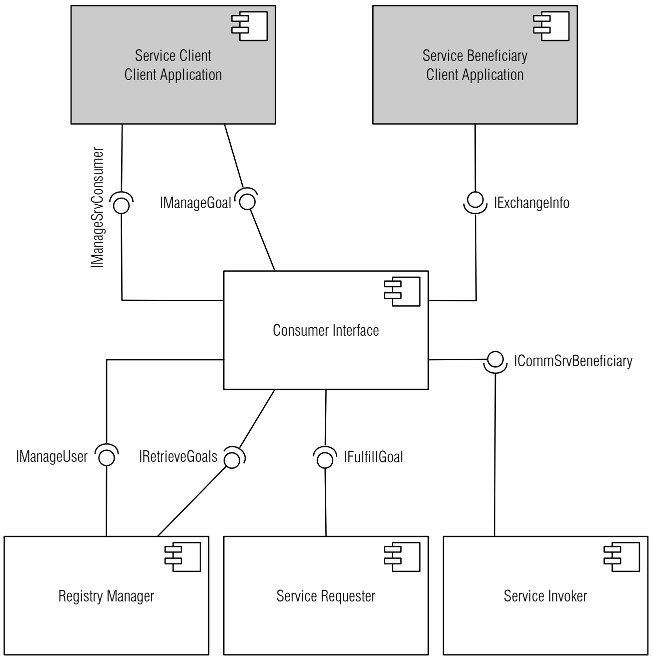
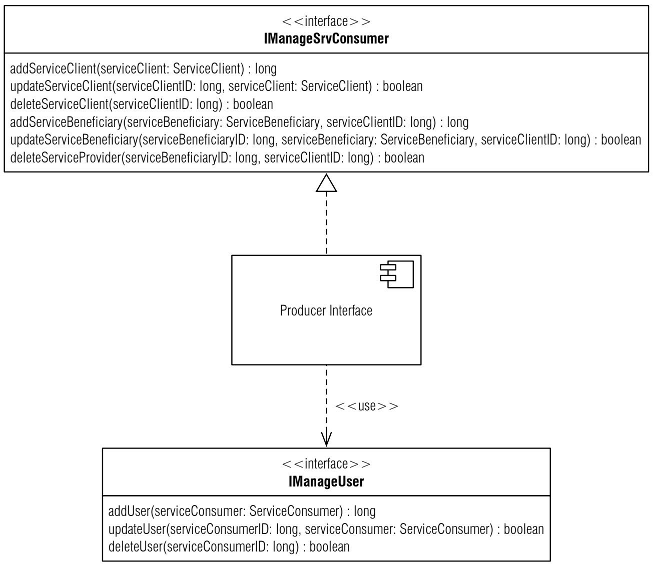


Figure 5-25 The interfaces of the Consumer Interface component

Before being able to request service provisioning support to the CASP, service clients have to be registered to the platform. Service clients interact with the CASP through Service Client Client Applications, which requests the registration (or update) of the service client to the Consumer Interface component. This request is received by the Consumer Interface component through its *IManageSrvConsumer* interface, and is forwarded to the Registry Manager using the *IManageUser* interface. The Registry Manager stores the registered service client in the User Repository. If the Consumer Interface receives a request for service client removal, the CASP checks whether that service client has contracted any service. If this is the case, the service client is requested to terminate the service contracts before its removal. Similar interface has been defined to add, update and delete Service Beneficiaries, as depicted in Figure 5-26.

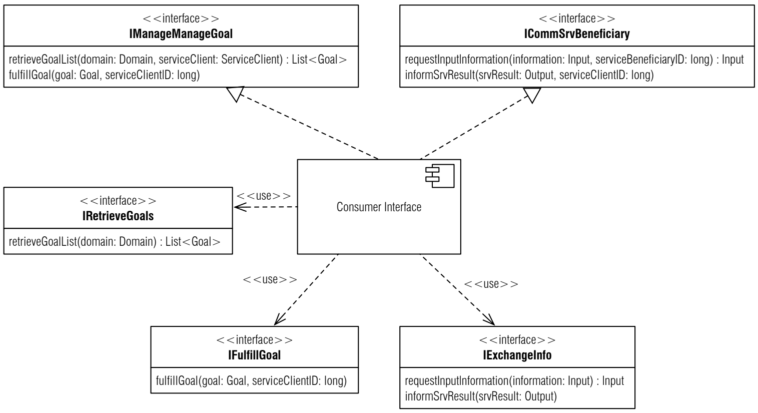
Figure 5-26 The interfaces of the Consumer Interface component to manage the registration of service consumers



<<interface>>
IManageSrvConsumer
addServiceClient(serviceClient: ServiceClient) : long
updateServiceClient(serviceClientID: long, serviceClient: ServiceClient) : boolean
deleteServiceClient(serviceClientID: long) : boolean
addServiceBeneficiary(serviceBeneficiary: ServiceBeneficiary, serviceClientID: long) : long
updateServiceBeneficiary(serviceBeneficiaryID: long, serviceBeneficiary: ServiceBeneficiary, serviceClientID: long) : boolean
deleteServiceProvider(serviceBeneficiaryID: long, serviceClientID: long) : boolean
Producer Interface
<<use>>
<<interface>>
IManageUser
addUser(serviceConsumer: ServiceConsumer) : long
updateUser(serviceConsumerID: long, serviceConsumer: ServiceConsumer) : boolean
deleteUser(serviceConsumerID: long) : boolean

The main utility of the Consumer Interface component is to support the service request. In this way, the service clients submit to the platform the goals they intend to have fulfilled. The Service Client Client Application component requests a service to the Consumer Interface. The GSF uses the concept of Goal as an abstraction for service request. Therefore, as depicted in Figure 5-27, the Service Client Client Application uses the *fulfullGoal* operation defined in the Consumer Interface component's *IManageGoal* interface to request the fulfillment of a goal. However, to constrain the goal options and guide the client's choice, the Service Client Client Application component should present to the Service Client a list of goals admissible for the domain in which the client is operating. To present these options, the Service Client Client Application component requests a list of the available goals to the Consumer Interface component, using the *retrieveGoalList* operation defined in the *IManageGoal* interface. The Consumer Interface component forwards the request to the Registry Manager, using the Registry Manager's *IRetrieveGoals* interface. The Registry Manager then retrieves the related domain ontology from the Ontology Repository, and extracts the admissible goals defined in the ontology.

The goal received from the service client by the Consumer Interface is forwarded to the Service Requester to create the service



<<interface>>
IManageManageGoal
retrieveGoalList(domain: Domain, serviceClient: ServiceClient) : List<Goal>
fulfillGoal(goal: Goal, serviceClientID: long)
<<interface>>
ICommSrvBeneficiary
requestInputInformation(information: Input, serviceBeneficiaryID: long) : Input
informSrvResult(srvResult: Output, serviceClientID: long)
<<interface>>
IRetrieveGoals
retrieveGoalList(domain: Domain) : List<Goal>
<<use>>
Consumer Interface
<<use>>
<<use>>
<<interface>>
IFulfillGoal
fulfillGoal(goal: Goal, serviceClientID: long)
<<interface>>
IExchangeInfo
requestInputInformation(information: Input) : Input
informSrvResult(srvResult: Output)

Figure 5-27 The interfaces of the Consumer Interface component to request services

request (detailed later in Section 5.6.1). For this, the Service Requester component offers the *IFulfillGoal* interface.

When the service is triggered to be executed, and the context-aware components are unable to provide information to be used as inputs for the service execution, the Service Invoker component requests that this information is asked to the Service Beneficiary. The Service Invoker component uses the *requestInputInformation* operation defined in the Consumer Interface component's *ICommSrvBeneficiary* interface. The Consumer Interface component asks the input information to the service beneficiary by using the *requestInputInformation* operation defined in the Service Beneficiary Client Application's *IExchangeInfo* interface. This interface is also used to retrieve to the service beneficiary the results of the service execution.

Figure 5-28 depicts our prototype Service Client Client Application's GUI. In this example, first the Service Client selects the domain of interest. After that, the client application retrieves available goals for that domain and shows to the service client options to define his goal. In our approach we describe a goal indirectly by specifying the parameters for a situation that satisfies the intended goal. In the example in Figure 5-28, the goal is to have the service beneficiary's ambient comfort preferences setup whenever the beneficiary enters his home or office. To characterize a situation that fulfills the goal, the service client specifies the room temperature, light color and intensity, and some auxiliary parameters, such as adjusting these settings whenever the beneficiary enters a room, allowing the settings to be implemented not only in the current but also in remote ambients, and making the beneficiary's favorite media available in these ambients.

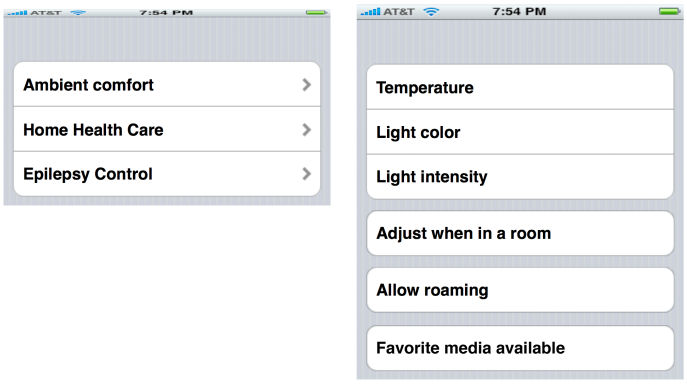


Figure 5-28 Example of a Service Client Client Application GUI

5.5 Context-Aware Components

Figure 5-29 depicts the CASP's context-aware components. These components support the registration of context providers, context sources and contextual information, and the request and subscription of contextual information. Once acquired, the relevant contextual information is forwarded to the CASP's internal component that requested the context information.

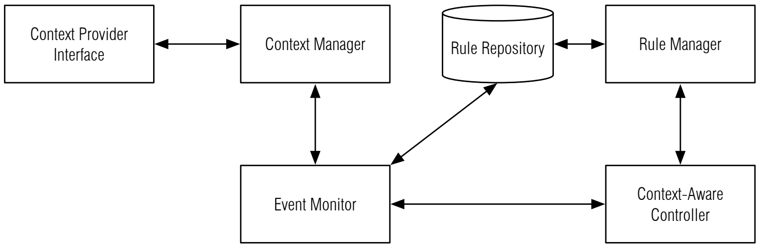


Figure 5-29 The CASP's context-aware components

5.5.1 Context Manager

The Context Manager component is based on the Context Management Service (CMS) [Ramparany et al., 2007] developed in the scope of the Amigo project [Ami]. The Context Manager provides the necessary middleware to manage context sources and their contextual information. Figure 5-30 shows the internal components of the Context Manager as well as the operations that are invoked on and used by these sub-components.

The Context Manager consists of two main element: the Context Broker (CB), and the Context Consumers (CC). The Context Broker keeps track of all the registered context sources and acts as a service directory of context sources, while the Context

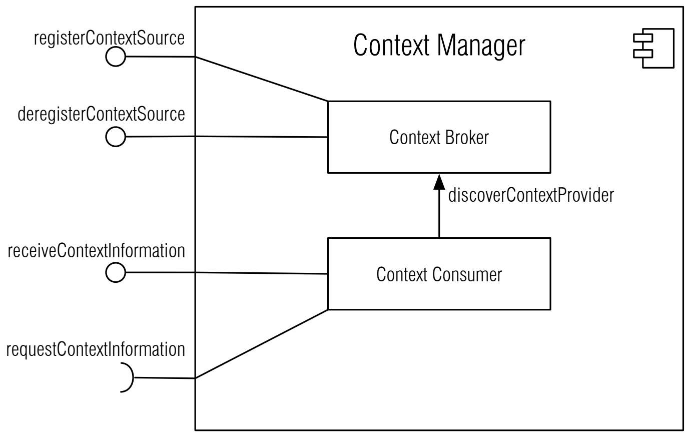


Figure 5-30 Internal structure of the Context Manager

Consumer finds the appropriate context sources by querying the Context Broker and uses the context information provided by the context source.

A context source provides two main modes of operation:

– Synchronous (request/response): the Context Consumer asks for context matching certain specified criteria, and the context source responds to the query by providing the appropriate context information.
– Asynchronous (publish/subscribe): the Context Consumer indicates to the context source that it wants to be informed of the changes in context information that meet certain specified criteria, and the context source informs the Context Consumer of these changes whenever they occur.

The synchronous mode of operation corresponds to the *impromptu* provisioning of context information discussed in Section 5.2.3, while the asynchronous mode corresponds to the subscribed provisioning of context information also discussed in Section 5.2.3.

5.5.2 Context Ontology

The Context Manager uses a context ontology (written in OWL) for context representation. Figure 5-31 shows the RDF representation of the generic *ContextParameter* concept, from which all other types of context information are derived. A *ContextParameter* has one or more object type properties, which refer to a (subclass of an) *Entity*. Each *ContextParameter* has also metadata associated with it, in the form of data or object type properties, which tell something about that particular *ContextParameter*, such as the probability that this context parameter is still valid or its timestamp.

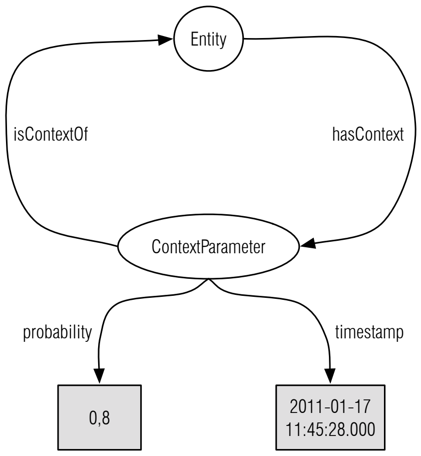


Figure 5-31 The RDF representation of the context ontology's ContextParameter concept

For specific types of context information, subclasses of the *ContextParameter* concept are derived. The *isContextOf* property is also specialized for specific context information. The RDF representation of Figure 5-32 shows an example in which the location of a person is represented as context information. In this example, a subclass of ContextParameter called PersonLocation is represented, which has two subproperties of isContextOf: an isLocationOf object property that refers to the location of the Person, and a hasLocation object property that refers to the Space that corresponds to the location of the person. The timestamp indicates when this context information was determined. This specific example represents that at 11:45 on January 17, 2011, Jerry was in the Kitchen.

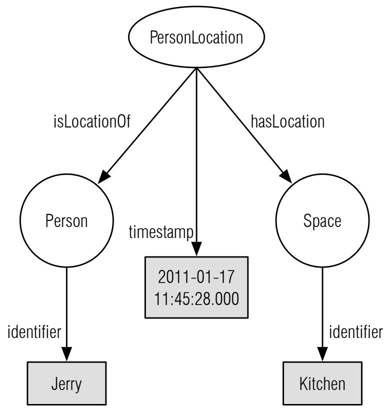


Figure 5-32 Example of a contextual information representation

The Context Consumer component uses SPARQL [Prud'hommeaux and Seaborne, 2008] for querying context sources, allowing a context consumer to ask only for the specific context information of interest. Taking the context from Figure 5-32 as example, Listing 5.2 presents the SPARQL query that asks for all the last known locations of a person.

Listing 5.2 SPARQL query to retrieve known location of a person

```
SELECT ?username ?roomname WHERE {
  ?user rdf:type contextOntology:Person .
  ?user context:identifier ?personId .
  ?userloc rdf:type context:PersonLocation .
  ?userloc context:isLocationOf ?person .
  ?userloc context:isLocatedIn ?room .
  ?room context:identifier ?roomname
}
```

5.5.3 Context monitoring

When one of the CASP's components needs contextual information, it requests this information to the Rule Manager component by defining a *monitoring rule.* This rule specifies the context to be monitored and the notification to be sent once the expected context holds. Once the requesting component has subscribed and started the monitoring rule, the context-aware components start gathering the required contextual information. In case the triggering condition defined in the monitoring rule holds, the context-aware components notifies the requesting component according to the notification message specified in the rule. For example, a rule can specify that a notification should be sent when John arrives home. In this example, the context-aware components monitor the John's location, and notifies the requesting component when he enters his home.

Following the Event-Control-Action pattern described in [Dockhorn Costa et al., 2005], four of the context-aware components, namely the Event Monitor, the Rule Manager, the Context-Aware Controller and the Rule Repository, are responsible for managing monitoring rules and notifying the requesting components when the rules' triggering conditions are met (see Figure 5-29). The Event Monitor component receives context data events from context sources through the Context Manager. The Event Monitor sends these events to the Context-Aware Controller component, which monitors them and evaluates the registered monitoring rules. When the triggering condition of the monitoring rule is evaluated to true, the Rule Manager is called to notify the requesting component. The subscribed monitoring rules are stored in a

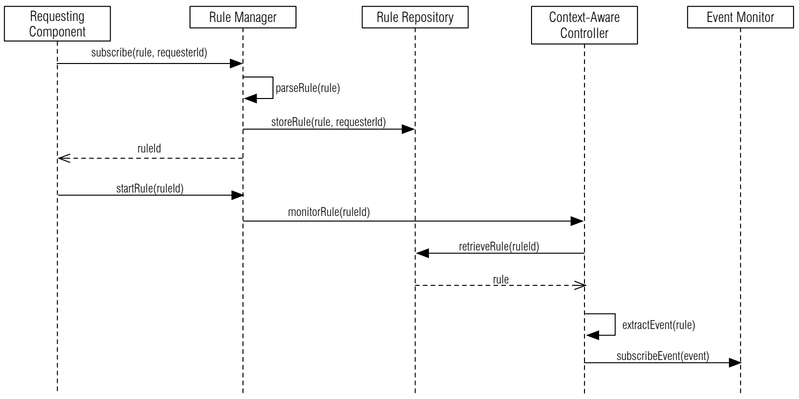
Rule Repository and made available for both the Rule Manager and Event Monitor components.

The RuleManager component provides interfaces for subscribe, unsubscribe, update, activate and deactivate rules. When a requesting component wants to register a rule, it sends the rule to the Rule Manager that is responsible for parsing, validating and storing the incoming rule. In the parsing and validating phases, the Rule Manager translates the given user rules to reaction rules that can be handled by the Context-Aware Controller.

The rules received by the Rule Manager from requesting components are expressed in a domain-specific ECA language named ECA-DL [Dockhorn Costa et al., 2006]. The Rule Manager component transforms this ECA-DL rule into a rule that can be handled by the underlying rule-engine. In our prototype, the context-aware components use the JESS rule engine [JES].

Once a rule is registered, it is available to the context-aware components but still not subject to monitoring, i.e., the rule is only registered in the Rule Repository, but its triggering condition is not activated to be evaluated. A requesting component has to activate the rule to start the evaluation of the rule's triggering condition. When a registered rule is activated, the Rule Manager component sends it to the Context-Aware Controller. The Context-Aware Controller component extracts the context variables (the event part) of the rule and submits the extracted events to the Event Monitor. Figure 5-33 shows a UML Sequence Diagram that depicts the message exchange of the rule subscription.

Figure 5-33 Sequence of interactions for monitoring rule subscription



The main functionality of the Event Monitor component is to facilitate the access to contextual data. The Event Monitor provides to other context-aware components a mechanism for subscribing to or querying for context data. For instance, if the Context-Aware Controller needs to monitor the battery level of a device, the Event Monitor searches for an appropriate context provider (and its related context sources) that could supply this information through the Context Manager component. Once a context provider is found, the Event Monitor subscribes to the request battery level data and informs the Context-Aware Controller of events containing the requested data.

The Event Monitor maintains a subscription to the corresponding context sources for each event that the Context-Aware Controller has requested. A list relating events and context sources should be maintained to avoid redundant subscriptions. To accomplish this, the Event Monitor analyzes the requested subscriptions searching for overlaps in the subscription's requirements. An example of a requirement overlap is when different rules request the same event from a single context source. In this, case the Event Monitor keeps only one subscription to the context source.

After subscribing the events contained in the rule to the Event Monitor, the Context-Aware Controller starts receiving notifications of the occurrence of these events. For each event notification received, the Context-Aware Controller evaluates the new information against the notification rules. When the rule's condition evaluates to true, the Context-Aware Controller informs the Rule Manager, which then notifies the requesting component that provides the requested contextual information.

5.5.4 ECA-DL

The ECA-DL has been developed following the event-condition-action (ECA) pattern [Dockhorn Costa et al., 2005]. Rules in ECA-DL consist of an Event part that models an occurrence of interest in the context, a Condition part that specifies a condition that must hold for the execution of the action to be triggered, and an Action part which consists of actions, typically service invocations.

ECA-DL is defined upon two complementary information and behavior foundations. Information foundation refers to the representation of the applications' universe of discourse, i.e., a domain ontology. For example, we should be able to express within ECA-DL rules whether people are in the house or not, whether objects are plugged in or not, whether persons and objects are collocated, among others. The behavior foundation of the ECA-DL language refers to the dynamics of rule execution, i.e., how and when a rule should be executed and what are the elements of the language that should be used to perform a particular piece of reactive behavior.

The context-aware components assume that one is only allowed to use a piece of knowledge in the ECA-DL rule if this has been previously defined in the domain ontology. For example, if the domain ontology does not define the concept of co-location, this concept cannot be referenced in ECA-DL rules [Dockhorn Costa et al., 2005]. A simple, example to illustrate the structure of ECA-DL rule is the following:

```
Upon ↩
   ↪ EnterFalse(isAtHome(ServiceBeneficiary.John))
When isAtHome(Device.Laptop123) and
  isOwnedBy(Device.Laptop, ↩
     ↪ ServiceBeneficiary.John) and
  hasMeeting(ServiceBeneficiary.John)
Do notify(ServiceInvoker,
  ServiceBeneficiary.John.location,
  Device.Laptop123.Location)
Lifetime from ‘‘Monday’’ to ‘‘Friday’’
```

This rule is used to detect if *John* leaves his house without his laptop on a day when he has a meeting. When this condition holds, the rule should notify the location of John and his laptop, so a proper communication method can be applied based on his location to inform him that he should pick up his laptop, at the laptop's location, for the upcoming meeting. In our experiments testing this rule in the scope of the Amigo project [Ami], we have used a RFID tag inside John's wallet to detect whether he is at his home, the GPS capability of his mobile phone to assess his location while outside his home, and the Bluetooth and WiFi signal triangulation to detect the laptop's location.

For the *isAtHome* event part of this example rule, the Event Monitor asks the Context Manager for a context source that keeps track of which service beneficiaries and devices are at home. When the Context Manager returns the reference for a context source, the Event Monitor subscribes to it with a query parameterized for *John* and for *Laptop123*. After the Event Monitor subscribes to the contextual information, the context source informs every time the result of the query changes. This allows the Event Monitor to test if the answer changed from TRUE to FALSE (the *EnterFalse* part of the *Upon* clause). In this example, the transition *EnterFalse* is used to express when *John* is no longer at home. The Event Monitor performs similar subscriptions to other contextual information for the conditions in the *When* clause.

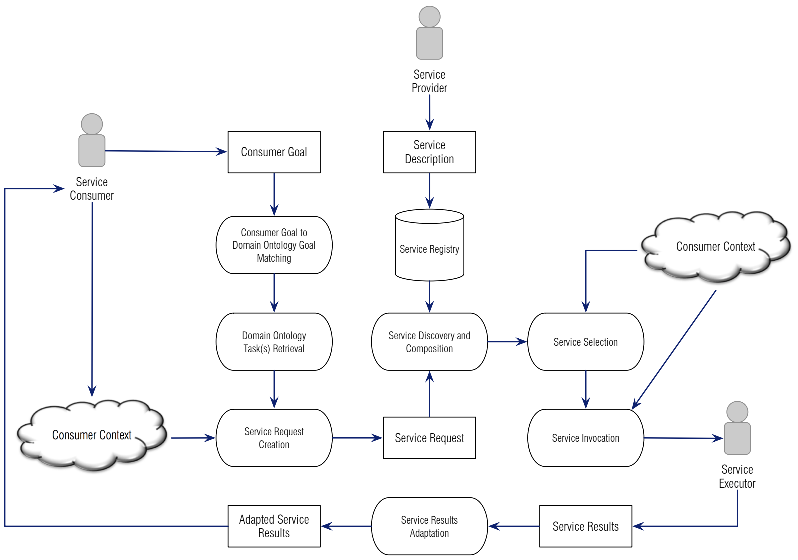
All the information from incoming events is evaluated by the Context-Aware Controller by pushing it to its internal rule engine for evaluation. Since the triggering element is the event, i.e., the element in the *Upon* clause, the conditions in the *When* clause are only evaluated if the event occurs. Once all the events and conditions specified in the ECA-DL rule are met, the Service Invoker component is notified with the location of *John* and of his laptop. In this example, the contextual information is used by the Service Invoker to trigger the execution of a service.

5.6 Service Provisioning Components

5.6.1 Service request creation

As depicted in Figure 5-34, the CASP's service provisioning starts when the CASP's Consumer Interface component receives the goal that the Service Consumer wants to achieve. The Consumer Interface forwards the goal to the Service Requester component, which tries to match this goal with the goals defined in the domain ontology, with which the Consumer Goal has been semantically annotated. For the sake of clarity, we refer to Consumer Goal as the goal which the Service Consumer submits to the platform to be fulfilled, and we refer to Domain Ontology Goal as one of the goals defined by the Domain Specialist for a given domain.

Figure 5-34 The CASP's service provisioning steps



The match between a Consumer Goal and a Domain Ontology Goal is assessed in two distinct ways:

1. When a Consumer Goal is selected directly from the set of

valid goals for that domain, a match does not have to be evaluated since the two goals refer to the same goal instance in the domain ontology. In this situation the platform just locates the referred goal instance in the domain ontology. For instance, suppose Service Consumer *Alfred* is going to a business trip and, instead of defining his own goal, he selects from the Travel domain ontology the goal instance *GetLuxuryHotelRoom* as his goal.

2. When a Consumer Goal has been defined by the Service Consumer, the platform looks for a matching goal by comparing the situations that can satisfy the Consumer Goal with the Situations satisfying each of the goals defined in the domain ontology. In the hotel room example, suppose that instead of selecting an already specified goal in the domain ontology, the Service Consumer describes the goal of having a comfortable accommodation at his trip's destination by defining that his goal is satisfied if he has a hotel suite booked at his destination in a hotel with a minimum of 4 stars. In this example the platform tries to find domain ontology goals that can be satisfied by the situation defined by the Service Consumer and finds the goal *GetLuxuryHotelRoom*, whose satisfying situation matches the situation described by the service consumer.

To find a match between the Consumer Goal and the Domain Ontology Goal, the Service Requester component retrieves the Domain Ontology Goals and their satisfying situations from the Registry Manager component. When a match between a Consumer Goal and a Domain Ontology Goal is found, the Service Requester component proceeds to retrieve from the Registry Manager the tasks defined in the domain ontology that support the goal. Once a task supporting the goal is found, the CASP is able to create a service request specifying the task that the candidate services should perform. Additionally, contextual information from the Service Consumer is queried to the Rule Manager component, and used to refine the service discovery and composition. An overview of the interactions between the Service Requester, the Registry Manager and the Rule Manager components that lead to the creation of the service request is depicted in Figure 5-35.

5.6.2 Service discovery and composition

After the service request has been created, the Service Requester component submits the created service request to the Service Finder component. The Service Finder is responsible for querying the Registry Manager for services complying with the service re-

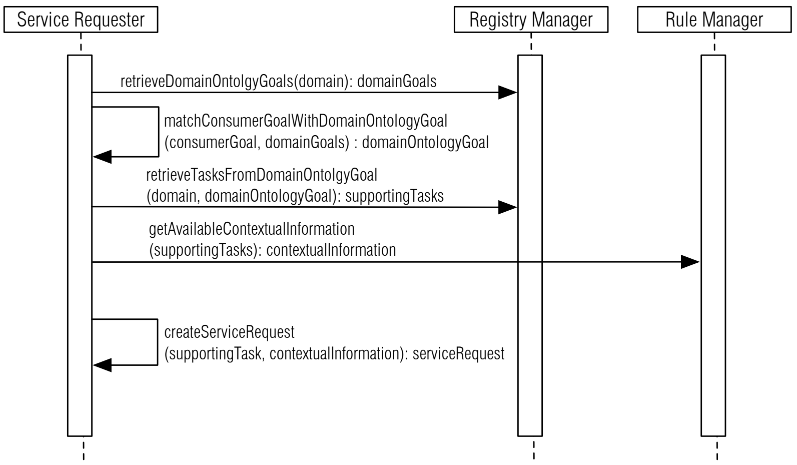

Service Requester
Registry Manager
Rule Manager
retrieveDomainOntolgyGoals(domain): domainGoals
matchConsumerGoalWithDomainOntologyGoal (consumerGoal, domainGoals) : domainOntologyGoal
retrieveTasksFromDomainOntolgyGoal (domain, domainOntologyGoal): supportingTasks
getAvailableContextualInformation (supportingTasks): contextualInformation
createServiceRequest (supportingTask, contextualInformation): serviceRequest

Figure 5-35 Overview of the interactions for service request creation

quest. If no service could be found, the Service Finder invokes the Service Composer component to try to generate a service composition that complies with the service request. We have based the CASP's Service Finder and Service Composer components on the Dynamic Composition of Services [1] (DynamiCoS) [Goncalves da Silva et al., 2011] platform, which is further discussed in Section 5.6.3.

When the CASP finds a service matching the service client's goal, and if the service provider allows negotiation for this service, the platform informs the successful discovery to the service client. In this case, the service client can proceed with service negotiation. Once an agreement on the terms of service provisioning is reached, the CASP searches in the descriptions related to the discovered service, and in the approved agreement for information required for the service activation, triggering and execution. In case service negotiation is not allowed, the CASP uses the adhesion service contract defined in the service description as the service agreement. Moreover, the platform verifies if the information required for the service activation, triggering and execution can be supplied by contextual information. If it is possible to use contextual information, the platform requests the provisioning of this information to the context providers. Otherwise, the information is requested to the service consumer, when needed.

The CASP monitors the requested contextual information and, once the activation condition holds, the discovered service becomes active and waiting for the triggering condition. When the

[1] http://dynamicos.sourceforge.net

triggering condition holds, the CASP requests the service execution to the Service Executor component, sending the input information gathered from the service beneficiary or from the context providers. When the service execution finishes, if any output information has to be delivered to the service client, the CASP processes this information by transforming it to the format expected by the service beneficiary, if necessary.

5.6.3 The DynamiCoS service platform

The CASP components responsible for service discovery and composition are based on the Dynamic Composition of Services (DynamiCoS) [Goncalves da Silva et al., 2011]. DynamiCoS is a platform for service discovery and composition based on semantic service descriptions. The DynamiCoS platform has been designed to be agnostic with respect to any specific service description language. To achieve language-neutrality, DynamiCoS creates mappings between service and service composition representation languages and its internal representations. Internally, DynamiCoS represents service and service compositions as tuples and graphs, respectively. Therefore, to use different languages to represent services and service compositions, it is only necessary to provide the appropriate mappings between these representation technologies and the DynamiCoS internal representations. Figure 5-36 depicts the DynamiCoS platform architecture.

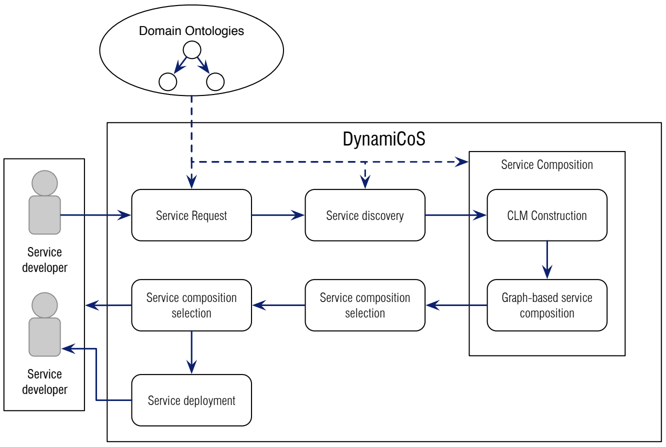


Figure 5-36 The architecture of the DynamiCoS service composition platform

A service in DynamiCoS is internally represented as a seven-tuple $s =< ID, I, O, P, E, G, NF >$, where ID is the service identifier, I is the set of service inputs, O is the set of service outputs,

P is the set of service preconditions, E is the set of service effects, G is the set of goals the service supports, NF is the set of service non-functional properties and constraint values. DynamiCoS assumes that services in a composition are of the type request-response.

A service composition is represented in DynamiCoS as a directed graph $G = (N, E)$. The graph nodes N represent services. Each node $n^i \in N$ represents a discovered service s^i, and a node can have multiple incoming and outgoing edges. Graph edges E represent the coupling between the output/effect of a service and the input/precondition of another service, i.e., $e_{i \to j} = n^i_{O/E} \to n^j_{I/P}$, where $i \neq j$, since DynamiCoS does not allow a service to be coupled with itself.

To perform a service discovery, the DynamiCoS requires that the service request contains a set of tasks the discovered service should perform. The discovery is performed by querying the service registry for all the services that *semantically* match the service request. This semantic match is carried out by comparing the service request's inputs, outputs, preconditions, effects (IOPE) and goals (G) (the root goal and its associated sub-goals) with the service tasks' inputs, outputs, preconditions and effects, and the service's supported goals defined in the registered service descriptions. Since DynamiCoS uses semantic-based service discovery, not only exact matches are retrieved but also partial semantic matches, such as when a concept is semantically subsumed by the service request parameter concept (i.e., $RequestedConcept \sqsupseteq DiscoveredConcept$).

In the DynamiCoS prototype, a service request XML file is parsed and the IOPE and goal parameters are extracted. The service registry is queried using the jUDDI API Inquiry operation [Apache, 2008], and the semantically-related concepts are defined using the OWL-API [Bechhofer et al., 2003] and the Pellet reasoner [Sirin et al., 2007].

The DynamiCos performs service composition by organizing the discovered services in a Causal Link Matrix (CLM) (REF) and storing all possible semantic connections (or causal links) between the discovered services input and output concepts. As defined in Equation 5.1, CLM rows represent the discovered services' input concepts ($DiscovServices_i$). Equation 5.2 shows that CLM columns represent service inputs concepts and the requested service outputs ($ServiceReq_O$).

$$CLM_{rows} = DiscovServices_I \tag{5.1}$$

$$CLM_{col} = DiscovServices_I \cup ServiceReq_O \quad (5.2)$$
$$\backslash(DiscovServices_I \cap ServiceReq_O)$$

When a given service S has an input that is semantically related with the input i in the CLM and an output that is semantically related with the semantic concept in CLM's column j, the service is stored in row i and column j. Moreover, for each service, DynamoCoS stores the semantic similarity for these values in the matrix. Four types of semantic similarity are supported by DynamiCoS:

- **Exact** ($\equiv$) when the output parameter $Output_y$ of service S_y is equivalent to the input parameter $Input_x$ of service S_x. Formally, $\tau \models Output_y \equiv Input_x$.
- **PlugIn** ($\sqsubseteq$) when the concept $Output_y$ is a sub-concept of the concept $Input_x$. Formally, $\tau \models Output_y \sqsubseteq Input_x$.
- **Subsume** ($\sqsupseteq$) when the concept $Output_y$ is a super-concept of the concept $Input_x$. Formally, $\tau \models Output_y \sqsupseteq Input_x$.
- **Disjoint** ($\perp$) when the concept $Output_y$ is semantically incompatible with the concept $Input_x$. Formally, $\tau \models Output_y \sqcap Input_x \sqsubseteq \perp$.

In DynamiCoS, the composition process is simplified since the service composition engine only traverses the CLM searching for services matching a given input/output. Algorithm 5.3, represented in a simplified pseudo code formalism, shows the DynamiCoS service composition algorithm.

Listing 5.3 DynamiCoS' graph composition algorithm

```
Input: CLM, ServiceReq
Result: ValidCompositions

// Variables
$activeG$; // Graph that is active in the algorithm ↩
    ↪ iteration
$activeN$; // Node that is active in the algorithm ↩
    ↪ iteration
$openG$; // Set of open graphs
$validG$; // Set of completed and valid graphs
// Initialization
if $CLM_{rows \cup columns} \supseteq ServiceReq_{I,O}$ then
  // Create new graph
  $activeG \leftarrow createNewGraph()$;
  $createInitialNodes()$;
  $openG \leftarrow activeG$;
else
  // Discovered services cannot fulfill the ↩
      ↪ service request
  $Stop$;
```

```
// Graph construction cycle
while $| openG |> 0$ do
  // Close graph if it matches $ServiceReq_{I,G}$
  if $activeG_{I,G} \supseteq ServiceReq_{I,G}$ then
    $validG \leftarrow activeG$;
    $openG \leftarrow openG \amalg activeG$;
    $activeG \leftarrow openG^{O}$;
    $activeN \leftarrow activeG_{openN^{O}}$;
    break; // Goes to next $openG$
  // Checks CLM for services that match open inputs
  foreach $semCon \in activeN_{I}$ do
    if $CLM_{colu} \supseteq semCon$ then
      $activeN \leftarrow CLMmatchingNode$;
    else
      $openG \leftarrow openG \amalg ActiveG$;
      $activeG \leftarrow openG \dagger O$;
      $activeN \leftarrow activeG_{openN^{O}}$;
      break; // No possible composition, goes to ↩
          ↪ next open graph
  // Check if graph NF props comply with ↩
      ↪ requested NF props
  if $activeG_{NF} \cap ServiceReq_{NF} = 0$ then
    $openG \leftarrow openG \amalg activeG$;
    break; // If Not, composition is not possible;
  // Prepare next cycle
  $openN \leftarrow openN \amalg activeN$;
```

The service composition algorithm starts by analyzing the CLM to check whether it contains the service request parameters (IOPE/ G). If these parameters are present, the algorithm proceeds to search for services that provide the requested outputs. In case services that provide the request outputs are found, the algorithm creates the initial matching nodes and proceeds with a backwards composition strategy towards the requested service inputs.

An *open*, i.e., not yet composed, input of the graph is resolved at each iteration of the algorithm. The algorithm matches the *open* inputs of the services in the graph with the output concepts of services from the CLM matrix, or column concepts. If the algorithm finds multiple services that match a given graph service input, a new composition graph is created, representing an alternative service composition. In each iteration of the algorithm, the aggregated non-functional properties in the composition graph are checked to assess whether these properties match the requested non-functional properties. If a composition graph does not match the requested non-functional properties it is discarded from the set of valid service compositions. When all the requested inputs, preconditions and goals from all the alternative service compositions are resolved, the algorithm successfully finishes.

In the DynamiCoS prototype presented in [da Silva, 2011], the service composition algorithm was implemented in Java, and the ontology handling and reasoning were implemented using the OWL-API [Bechhofer et al., 2003] and Pellet [Sirin et al., 2007], respectively. The prototype used the jGraphT library [jGr, 2011] to print out the resulting service composition graphs, allowing them to be manually verified.

5.6.4 The integration of the CASP and the DynamiCoS

Since both the CASP and the DynamiCoS platforms were designed following similar principles, such as the use of the goal concept and the support for semantic annotations, the integration of the DynamiCoS service discovery and composition components required only minor adjustments. The necessary adjustments for the integration of the DynamiCoS components into the CASP have been the following:

Semantic-based service description language

In the DynamiCoS original prototype, Spatel [Almeida et al., 2006] was used as service description language. Spatel was developed in the scope of the European IST-SPICE project [Cordier et al., 2006], where the development of DynamiCoS was initiated. Spatel supports the semantic annotation of inputs, outputs, preconditions and effects of service operations, the definition of service goals, and the definition of non-functional properties of services. The Spatel semantic annotations refer to concepts defined in four ontologies described in OWL, namely, *Goals.owl*, which describes the goals that the service clients can selected from; *Non-Functional.owl*, which describes a set of non-functional properties to be used in service descriptions; *Core.owl* and *IOTypes.owl*, which can be used to describe services and service requests' IOPE parameters, respectively.

The contents of the four ontologies used in DynamiCoS are defined within a single domain ontology in GSF. Therefore, instead of extracting this information from four different OWL files, in the CASP the DynamiCoS components get the same information from one OWL file containing the whole ontology for a given domain. In this way the GSF fosters integrity among the concepts defined in a domain ontology and their consistency.

Moreover, in the CASP prototype, we have used SA-WSDL as semantic-based service description language. For this reason, changes had to be made in the DynamiCoS components to allow them to interpret service descriptions written in SA-WSDL instead of Spatel.

Service request creation

The DynamiCoS platform discovers and composes services based on inputs, outputs, pre-conditions, effects, non-functional properties and goals. The DynamiCoS service request is defined as a description of a desired service, and the discovery and composition processes try to match the characteristics of this desired service against the service descriptions available in the service registry.

Since the GSF, and consequently, the CASP, targets non-technical service consumers, one of the main assumptions is that these consumers do not need to have the technical expertise to write a service specification. In GSF the service is requested by the service consumer by means of the goal that the consumer wants to be fulfilled. Consequently, before invoking the DynamiCoS' service discovery component, the CASP needs to create the description of the desired service needed by the DynamiCoS component. This description is obtained by discovering the task(s) that can support the service consumer's goal. The DynamiCoS concept of service corresponds to the GSO concept of Service Task. Therefore, when the CASP discovers tasks specified in the domain ontology that can support a given goal, these tasks are used to obtain the definition of the desired service, providing information about the inputs, outputs, pre-conditions and effects that the discovered services should comply with.

Selected service

The DynamiCoS components, after successfully discovering or composing a service, return the selected service to the client or execute the service. In the integration of the DynamiCoS components within the CASP, we considered that the client is the CASP, and after successful service discovery or composition, the CASP takes this selected service and proceeds with the service activation, triggering, invocation and delivery steps. Therefore, in this regard, only small adjustments had to be made to the DynamiCoS components to hand over the selected service back to the CASP for further processing.

Chapter 6

Case Study and Evaluation

This chapter demonstrates the suitability of the Goal-Based Service Framework and its architectural components discussed throughout Chapters 3 to 5, by means of a case study where the framework components are used and tested. The usage scenarios we have considered in this case study are in the Home Health Care domain due to its societal relevance, and its suitability to the objectives of our service provisioning framework.

This chapter is structured following the approach we used to create and execute this case study. Section 6.1 presents and justifies the Health Care usage scenario, Section 6.2 presents the modeling of the scenario's domain using the Goal-Based Domain Specification Language (GDSL), Section 6.3 shows how this usage scenario has been supported by the Context-Aware Service Platform, and finally, Section 6.4 discusses the results of the case study, evaluating the framework's performance.

6.1 Home Health Care Usage Scenarios

Throughout this thesis we have been using examples related to the Home Health Care domain. This domain has been chosen due to the following reasons:

- Global applicability. Scenarios in this domain can be applicable in different parts of the world, specially in countries where the elderly population has reached a significant percentage of the general population. In these countries, there is a pressing need for providing conditions allowing this population to remain in their houses or at elderly-exclusive facilities with the highest possible quality of life.
- Non-technical users. In this domain the end-users are among all segments of society, regardless of their professional background. Therefore, we cannot rely on their technical knowl-

edge to write service request specifications in technical terms, such as WSDL documents.

- Applicability of context-awareness. In this domain, the application of context awareness is particularly useful. For example, in the Netherlands there has been a number of research projects aiming at providing context-aware infrastructure to foster more automatic behavior of the applications deployed in this domain, and to reduce the need for direct user interaction [Ami, A-M, U-C].
- Co-existence of social and computational service. In the Health Care domain in general, and the Home Health Care domain in particular, it is straightforward to identify social and computational services. For instance, services that can can be automated by means of computational devices (computational services), such as appointment scheduling, vital signs' monitoring, etc., coexist with services that pertain exclusively to the social level (social services), such as a medical consultation or a medical diagnosis.

The following scenarios illustrate relevant situations in the health care domain from which we modeled this domain. These situations have been taken from the U-Care [U-C] and A-MUSE project documentation [A-M], and personal experience with the Dutch health system. The Home Health Care scenarios are as follows:

Usage scenario 1 - Assisted living

John Pieters is 78 years old and living alone in a care center. He developed chronic obstructive pulmonary disease (COPD). The treatment of Mr. Pieters' disease focuses on reducing symptoms and avoiding further deterioration of his condition. Some of his medicines work for the symptoms, but physical exercise is the key treatment. The original series of exercise was explained once at the doctor's office. Since then, Mr. Pieters conducts them at his home. During the exercises, he uses a finger clip, which measures the oxygen level in his blood and his heart beat rate. Julie is a software-based personal assistant from whom Mr. Pieters gets feedback on how long he should do each exercise, based on those measurements. Thanks to this feedback, Mr. Pieters is confident to continue the exercise for longer than he would do otherwise. [van 't Klooster et al., 2009]

A COPD nurse uses a two-way video system to check-up on Mr. Pieters monthly. After those check-ups, the doctor may adjust the exercise levels and medicines, based on the acquired measurements *and progress of the disease. The COPD nurse also adds the next check-up into Mr. Pieters' calendar service. Reminders for doing the exercises, taking the medicine and the meetings are sent to Mr. Pieters through Julie, either at home via the wall-mounted screen or (when he is underway) via his mobile phone. This helps him, as his memory is progressively failing.* [van 't Klooster et al., 2009]

Mr. Pieters likes Alice, one of the caregivers that regularly visit him. Alice not only helps with the housekeeping in his apartment, but also checks up on him once in a while via Julie to see how he is doing. Julie suggests activities and new inhabitants of the care center for him to meet. One new inhabitant turns out to be a friendly man, and Alice arranges their conversation through Julie. Afterwards, Mr. Pieters and the new inhabitant meet occasionally for a walk or coffee. [van 't Klooster et al., 2009]

Usage scenario 2 - Medication monitoring

Jan and Linda are 74 and 67 years old respectively. Despite their age and having various medical conditions, they prefer to remain living in their own home. They need to take certain medicines at specific schedule. However, they suffer from Alzheimer's disease and may not remember when to take which medicine and at which dosage. In this situation, a reminder service may help them remember the prescribed time and an electronic medicine dispenser may help them take the correct medicine at the correct dosage. In some situations, the reminder and the dispenser are not enough to guarantee that Jan and Julia take their medicine. For instance, Jan sometimes ignores the reminder and does not take his medicine because he his too disoriented. In such situations, assistance from other people is necessary. Therefore, the couple uses a service that identifies the situations where Jan or Julia did not properly take their medicines and requests for external help. Moreover, Jan has a hearing problem and uses a hearing aid, and Linda cannot see well and because of this she must wear glasses. [Zarifi Eslami et al., 2010]

Usage scenario 3 - Seizure control

Maria has a chronic epileptic condition. However, she wants to carry on with her life as normally as possible. As a daily routine, every morning she runs in the park near her house. It is possible to detect an imminent epileptic seizure based on body signals. Therefore, she receives warnings if her body signals reach a critical *point so she can stop running and try to put herself in a resting position. Concurrently, a relative or friend which is closer to her location is warned of her potentially coming seizure and head on to her whereabouts. The assigned or on-duty caregiver also receives information about her body signals, her location, which relative or friend has been warned and when and how far he/she is from Maria's location. This information is used by the caregiver to decide whether to send an ambulance (depending on the severity of the body signals), or to contact the warned relative or friend to provide further assistance instructions and receive extra situation assessment.* [A-M, Bonino da Silva Santos et al., 2010a]

Usage scenario 4 - Traveler patient

Lucia is a professional whose job requires frequent business trips. In one of these trips, she has a health-related event and needs to consult with a doctor. However, since she is not near her house and, consequently, far away from her general practitioner, she needs to find a nearby doctor that complies with her health insurance's conditions. [Bonino da Silva Santos et al., 2010b]

Since we assume that all the patients mentioned here live in the Netherlands, they are obliged by law to be covered by a health insurance. Several companies offer health insurance policies, and these policies must comply with federal regulations concerning the types of treatments covered by each policy, reimbursement procedures, own risk, dental care, physiotherapy, among others. Everyone is free to choose the health insurance company and the insurance policy of preference, and changes are allowed once a year.

6.2 Domain Modeling

Below we define the domain model used in our case study in terms of stakeholders, goals and tasks, aiming at supporting the scenarios described before.

6.2.1 Domain Stakeholders

From the usage scenario presented in the previous section, we have identified a set of stakeholder types, namely, patient, caregiver, health insurance company, personal assistant, nurse, physician and federal health care regulator agency. Some of these stakeholders have hierarchical relationships with others, such as physician and nurse that can be classified as sub-types of a caregiver. All of these stakeholders have their *intentionality*, and therefore, they are classified as agents in the GSO.

The identified agents can be further classified based on whether these roles can be instantiated by persons, by organizations or by software. For instance, a physician agent role is instantiated by a person while a health care insurance agent role is instantiated by an organization. Figure 6-1 depicts the identified agents classified as Human, Institutional (organizations) or Artificial Agents (e.g. software).

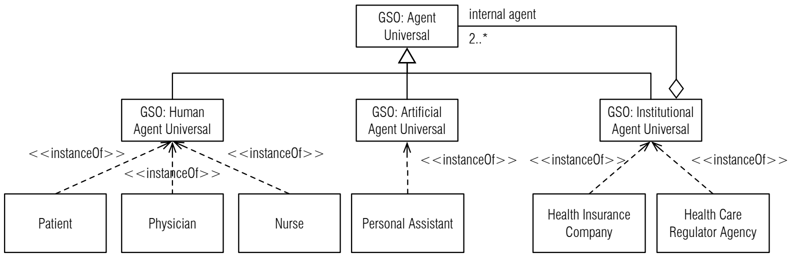


Figure 6-1 The identified agents classified as human, institutional or artificial agents

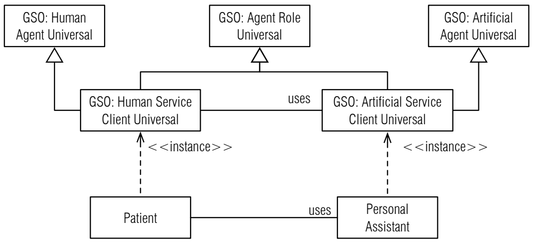
In Figure 6-1 we use the UML stereotype *"instanceOf"* to represent that the *Agent Universal*, the *Human Agent Universal*, the *Artificial Agent Universal* and the *Institutional Agent Universal* concepts are used here as language primitives to classify the identified agents. In other words, the GSO concepts are at the modeling language level (or metamodel level) while the Patient, Caregiver, Physician, Nurse, Personal Assistant, Health Insurance Company and Health Care Regulator Agency concepts are at the model level and are instances of the GSO concepts.

Taking into account the roles these agents play in service provisioning, we classified them into service providers, service clients, service executors and service beneficiaries. We classified the *Patient* agent as a service client because, in this usage scenario, the patient is the main target of the home health care services. Regarding the *Personal Assistant* agent, its classification is as straightforward as the *Patient.* Based on the usage scenario #1, we noticed that the personal assistant provides informational services to the patient (such as reminding the patient that it is time for the medicine) and, as such, can be classified as service provider. However, the personal assistant also acts on behalf of the patient as its surrogate software agent and, as such, can be classified as the patient's artificial service client. In the modeling of our usage scenario we opted for the later classification.

Figure 6-2 depicts the *Patient* agent classified using the *Hu-*

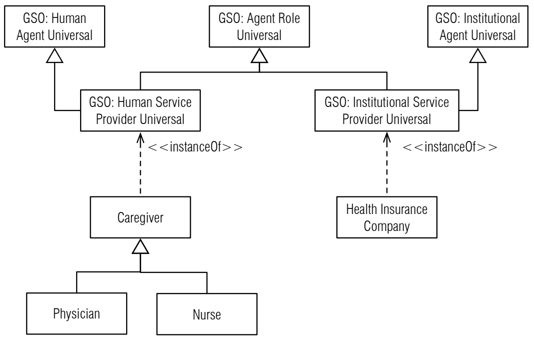
man Service Client Universal concept while the *Personal Assistant* agent is classified using the *Artificial Service Client Universal* concept.

Figure 6-2
The service clients


GSO: Human Agent Universal
GSO: Agent Role Universal
GSO: Artificial Agent Universal
GSO: Human Service Client Universal
uses
GSO: Artificial Service Client Universal
<<instance>>
<<instance>>
Patient
uses
Personal Assistant

In our usage scenarios, the providers of the health services are the caregivers, specialized into physicians and nurses, and the health insurance companies. Figure 6-3 depicts the *Caregiver* agent classified as a GSO *Human Service Provider*, while the *Health Insurance Company* agent is classified as a GSO *Institutional Service Provider.*

Figure 6-3
The service providers


GSO: Human Agent Universal
GSO: Agent Role Universal
GSO: Institutional Agent Universal
GSO: Human Service Provider Universal
GSO: Institutional Service Provider Universal
<<instanceOf>>
<<instanceOf>>
Caregiver
Health Insurance Company
Physician
Nurse

The *Health Care Regulator Agency* agent does not play any service provisioning role in our usage scenarios. This agent is a normative authority, setting policies and guidelines for the functioning of the health care sector. Therefore, caregivers and health insurance companies have to comply with these regulations. The role of the regulator agency is further discussed in the following sections.

The classification of the identified agents into service provisioning roles follows their transient nature, i.e., the classification is valid in the scope of the presented usage scenarios, whereas in other possible scenarios their classes may change. For instance, in a medical trial scenario, instead of being classified as a service client as in our usage scenarios, the patient agent may be classified as the provider (the Service Provider role) of a *supplying medical* *information* service for the trial study.

At the instance level, the classification of individuals as instances of the identified agents, and the consequent assignment of individuals to the roles of service client, service beneficiary, service provider and service executor can be done during domain specification or at runtime. When the assignment is performed during domain specification, it remains fixed until the domain specification is updated, i.e., until the domain specialist modifies the domain ontology. Contrarily, when the assignment is performed at runtime, it can be dynamically changed by the supporting platform.

For instance, Figure 6-4 depicts a model where Mr. Pieters (in usage scenario #1) is classified as an instance of the *Patient* concept and Julie is classified as an instance of the *Personal Assistant* concept. If we consider that this model has been defined during domain specification, it implies that Mr. Pieters is always going to play the role of *Human Service Client* and Julie is always going to play the role of *Artificial Service Client*, as long as the model is not modified. However, if the model (or at least the instance-level part) has been dynamically defined at runtime, it opens the possibility that in a certain circumstance Mr. Pieters plays the role of *Human Service Client* (e.g., when he receives medical care) while in another circumstance he plays the role of a *Human Service Provider* (e.g., when he provides medical-related information to be used by the Health Care Regulator Agency for statistical purposes).

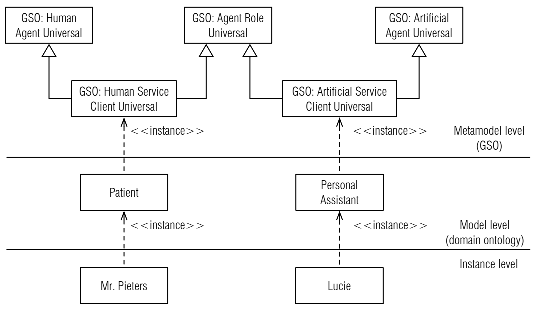

GSO: Human Agent Universal
GSO: Agent Role Universal
GSO: Artificial Agent Universal
GSO: Human Service Client Universal
GSO: Artificial Service Client Universal
<<instance>>
<<instance>>
Metamodel level (GSO)
Patient
Personal Assistant
<<instance>>
<<instance>>
Model level (domain ontology)
Instance level
Mr. Pieters
Lucie

Figure 6-4 The instance-level model with service clients

In this case study we dynamically classify the instance-level individual agents at runtime. This dynamic classification is performed based on context-aware information gathered by the supporting platform and is further discussed in the following sections.

6.2.2 Domain Goals

When producing a domain specification, the domain specialist should identify goals that can be adopted by classes of agents in this domain. For instance, in a football domain, the *Striker* agent may have the goal of *scoring the maximum amount of goals per match*. Conversely, classes of agents can be characterized by the goals they are assigned. For instance, the *Goal Keeper* agent can be characterized as having the goal of *avoiding goals from the opposing team*.

In our approach we are focused on the goals of the service clients. Therefore, in our usage scenario we have identified a set of goals that a client of medical services may have and assigned these goal to the *Patient* agent. These goals are used as *template* goals, which represent typical situations in the domain.

The most basic and general goal that a client of medical services may have is to be and stay healthy. As defined by the World Health Organization (WHO), health is *"a state of physical, mental, and social well-being"* [Grad, 2002]. Therefore, one can be physically healthy while being mentally ill, others can be mentally and socially healthy while physically ill, etc. In this case study we assume that the goal of being healthy entails that the patient should become healthy if he is in an unhealthy situation or stay healthy if he is already in a healthy situation. Moreover, being a continuously pursued goal, it may be the case that a given patient can only reach a certain level of health. For instance, a patient with a chronic disease may only be "as healthy as possible" due to its overall condition.

In our usage scenario #1, we have a COPD patient that aims at improving his health through a medical treatment. This medical treatment consists of medication and physical exercises targeted at enhancing the patient's cardiovascular condition. Based on the needs of this kind of patient, we can infer that for this patient, the goal of being healthy can be specialized into the goal of getting medical treatment. Figure 6-5 depicts our goal model for COPD patients. In this model the *Be Healthy* root goal is defined as an instance of GSO's *Complex Goal* concept. This root goal is decomposed into the other complex goal *Get COPD medical treatment* using the GSO's *AND goal composition* relation (see Section 4.3, Figure 4-19). Using the same relation, the goal of getting medical treatment is further decomposed into the atomic goals *Be medicated* and *Improve cardiovascular condition*. Following the holistic view of the WHO's health definition, we defined as another component of the *Be Healthy* goal the atomic goal of *Maintain social* *activities.* While the previous goals were defined to improve the physical health, this last goal has been defined to improve the mental health of the patient.

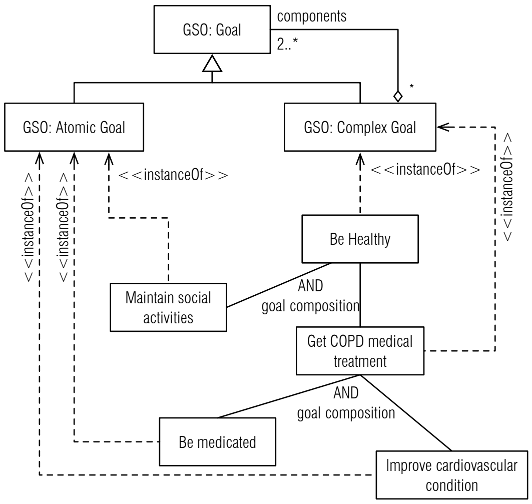


Figure 6-5 Identified goals for COPD patients

In the case of the Alzheimer's patients in our usage scenario #2 , their goals are similar to the COPD patient with the exception that they do not need to improve their cardiovascular condition and, therefore, their medical treatment includes taking medicines. However, these patients' goal of being medicated can be composed of administering the medication themselves or being assisted. Figure 6-6 depicts the goal model for patients needing assisted medication, where the goal *Be Medicated* is defined as a complex goal that could be fulfilled by either the fulfillment of the *Self medication* atomic goal, where the patient administers himself the medication, or the fulfillment of the *Assisted medication* atomic goal, where the patient receives external assistance to get the medication. For simplicity we did not depict in this model the root goal *Be Healthy* and its first decomposition into getting medical treatment and maintaining social activities. However, the goal *Be Medicated* is a specialization of the goal *Get medical treatment.*

Having a chronic condition that otherwise does not impose any physical limitation, the epileptic patient from our usage scenario #3 has only the goal of better handling seizures. As depicted in Figure 6-7, the goal of handling seizure can be either fulfilled by the goal *Prevent seizure* or by the goal *Minimize seizure consequence.*

The traveler patient from our usage scenario #4 has the goal of getting a medical consultation on her current location as depicted

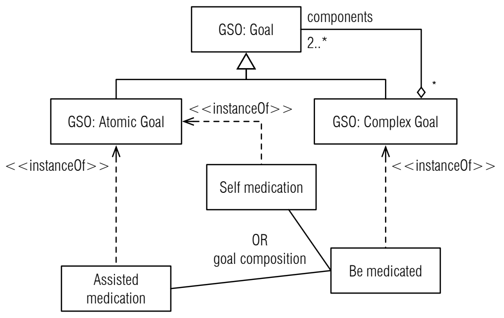


Figure 6-6 Identified goals for patients needing assisted medication

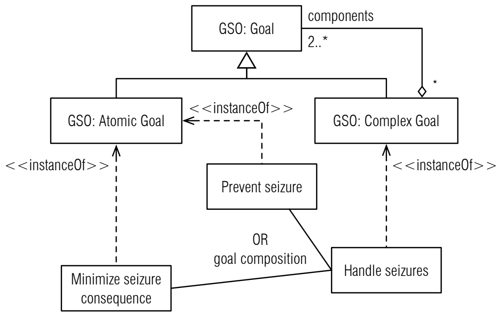
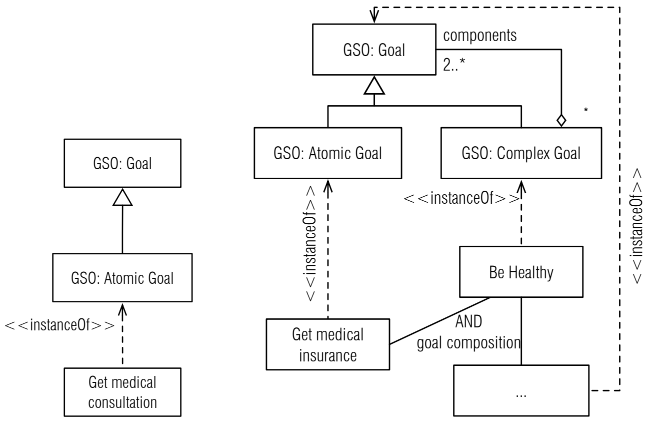


Figure 6-7 Identified goals for epileptic patients

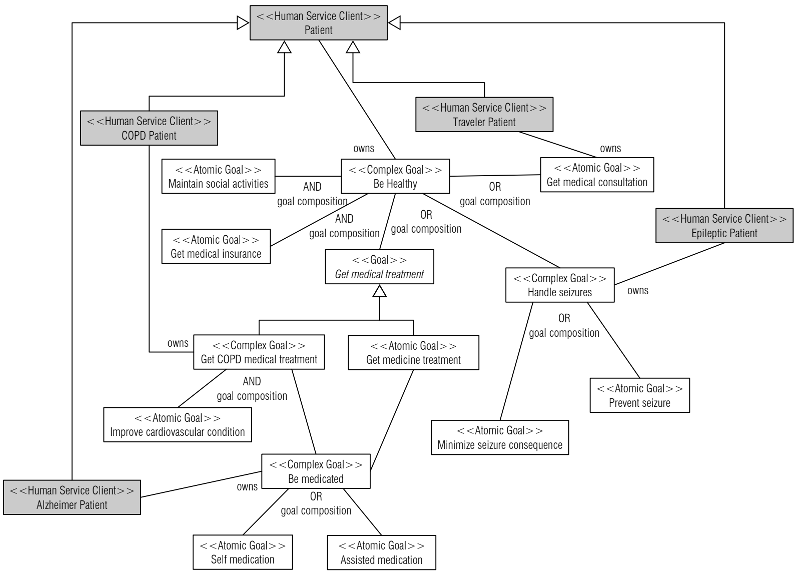
in Figure 6-8a. Moreover, as stated in the usage scenario, all our patients are required to have medical insurance in order to receive any medical care. In our model, we defined this legal requirement as a general goal of having medical insurance. More specifically, we modeled that the *Be Healthy* goal always consists of the goal *Get medical insurance*, as depicted in Figure 6-8b.



(a) Traveler patient goal (b) All patients' goal

Figure 6-8 Identified goal for traveler patients and a general goal for all patients

After the goals have been identified and modeled, the domain specialist can assign different service clients to the goals. In Figure 6-9 we present the set of goals we have identified for our usage scenario and assigned goals to different service clients. We have defined a hierarchy of patients from the root *Patient* being refined into *COPD Patient*, *Alzheimer Patient*, *Traveler Patient* and *Epileptic Patient*. With this hierarchy, the platform can either identify the type of patient based on the intended goal, or the goal to be assigned to the patient based on its specific type. For instance, if the patient has been identified as a traveler, the platform can automatically assign the goal *Get medical consultation.* Inversely, if a given patient informs the platform that it intended the goal *Get COPD medical treatment*, the platform can infer that this patient is a COPD patient.

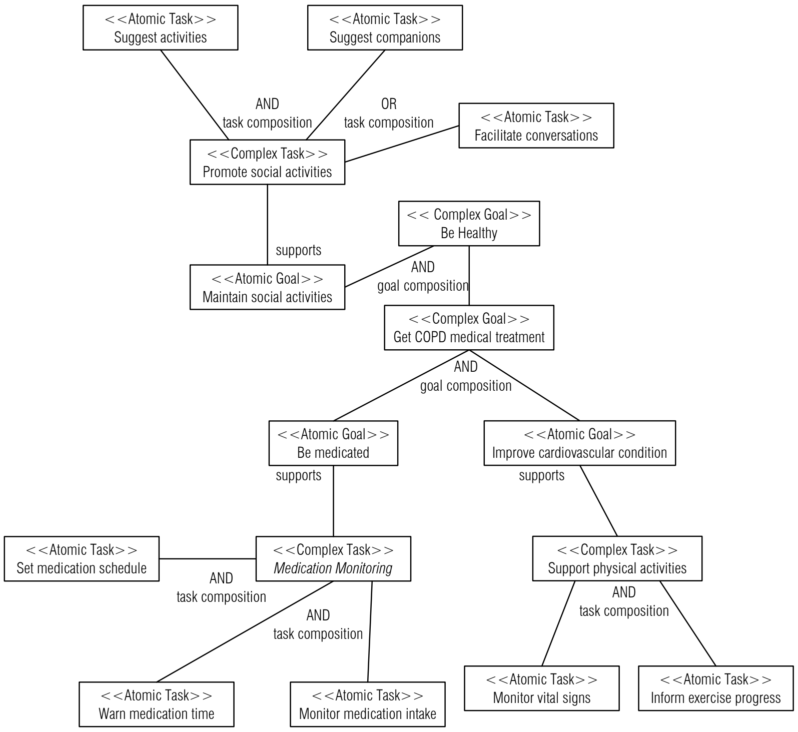

<<Human Service Client>> Patient
<<Human Service Client>> COPD Patient
<<Human Service Client>> Traveler Patient
owns
<<Atomic Goal>> Maintain social activities
AND goal composition
<<Complex Goal>> Be Healthy
OR goal composition
<<Atomic Goal>> Get medical consultation
<<Human Service Client>> Epileptic Patient
AND goal composition
OR goal composition
<<Atomic Goal>> Get medical insurance
<<Goal>> Get medical treatment
<<Complex Goal>> Handle seizures
owns
OR goal composition
<<Complex Goal>> Get COPD medical treatment
<<Atomic Goal>> Get medicine treatment
AND goal composition
<<Atomic Goal>> Prevent seizure
<<Atomic Goal>> Improve cardiovascular condition
<<Atomic Goal>> Minimize seizure consequence
<<Complex Goal>> Be medicated
<<Human Service Client>> Alzheimer Patient
OR goal composition
<<Atomic Goal>> Self medication
<<Atomic Goal>> Assisted medication
Figure 6-9 Identified goals and their owners

6.2.3 Domain Tasks

Once the set of goals has been defined, a domain specialist can proceed with the elaboration of the domain specification by defining a set of tasks whose outcome satisfy these goals. At runtime these tasks are matched with service tasks to find appropriate services to fulfill the goals. However, while the service tasks represent the concrete, implemented tasks, the tasks defined at the domain specification time should represent more abstract and high level tasks, allowing them to be matched with the more detailed and implementation-specific tasks of the services.

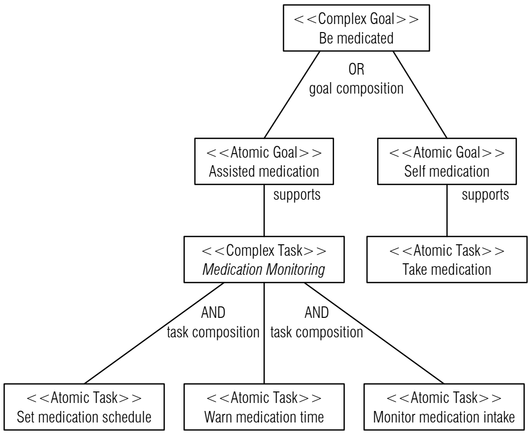
In this case study we adopted the strategy of defining tasks for the leaf goals, i.e., goals that are no further specialized or decomposed. Figure 6-10 depicts the tasks that support the COPD patient's goals. In this model, the goal *Maintain social activities* is supported by the complex task *Promote social activities.* This complex task can be further decomposed using the *AND task composition* relation into the atomic tasks *Suggest activities* and *Suggest companions.* An alternative of decomposition for the complex task is defined using the *OR task composition* relation to the atomic task *Facilitate conversations.*

Figure 6-10 The tasks supporting the COPD patient's goals


<<Atomic Task>> Suggest activities
<<Atomic Task>> Suggest companions
AND task composition
OR task composition
<<Atomic Task>> Facilitate conversations
<<Complex Task>> Promote social activities
<< Complex Goal>> Be Healthy
supports
AND goal composition
<<Atomic Goal>> Maintain social activities
<<Complex Goal>> Get COPD medical treatment
AND goal composition
<<Atomic Goal>> Be medicated
<<Atomic Goal>> Improve cardiovascular condition
supports
supports
<<Atomic Task>> Set medication schedule
<<Complex Task>> Medication Monitoring
<<Complex Task>> Support physical activities
AND task composition
AND task composition
AND task composition
<<Atomic Task>> Warn medication time
<<Atomic Task>> Monitor medication intake
<<Atomic Task>> Monitor vital signs
<<Atomic Task>> Inform exercise progress

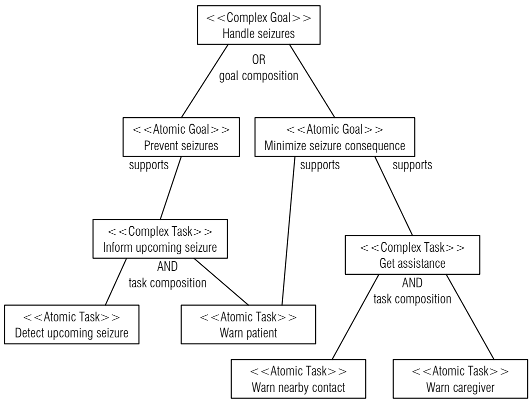
In the case of tasks related to the *Be medicated* goal, the Alzheimer patients goals can also be satisfied by these tasks. However, we have differentiated between assisted and self-medication (see Figure 6-9). As depicted in Figure 6-11, when the patient is able to administer the medication himself, the task to be performed is only the *Take medication* task. If the patient needs assistance to take his medication, we have defined the abstract complex task *Medication Monitoring*, which is decomposed into the tasks *Set medication schedule*, *Monitor medication intake* and *Warn medication time.*

Figure 6-11 The medication-related tasks



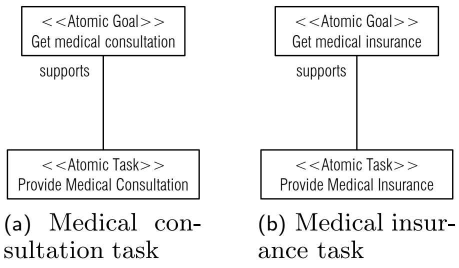
For the epileptic patient's goals, Figure 6-12 depicts the tasks we defined to support these goals. To prevent seizures, we have defined the complex task *Inform upcoming seizure*, which consists of the task *Detect upcoming seizure*, based on the monitoring of the patient's body signals, and the task *Warn patient*, to allow the patient to interrupt any physical activity and put herself in a resting position. If the seizure is unavoidable, we have defined two tasks to support the goal of *Minimize seizure consequence*: task *Warn patient*, to allow the patient to put itself in an appropriate position for the seizure, and task *Get assistance*, which in its structure consists of the task *Warn nearby contact*, to inform a contact person located nearby the patient of the upcoming seizure, and the task *Warn caregiver*.

Figure 6-12 The tasks supporting the epileptic patient's goals



The last two goals are supported by the tasks depicted in Figure 6-13. The goal *Get medical consultation* is supported by the task *Provide Medical Consultation*, while the goal *Get medical in-*

surance is supported by the task *Provide medical insurance*, as depicted in Figure 6-13a and in Figure 6-13b, respectively.



(a) Medical consultation task (b) Medical insurance task

Figure 6-13 The tasks supporting the goal of getting a medical consultation and medical insurance

6.3 Software Platform Support of Service Provisioning

After modeling the Health Care from our usage scenarios using GDSL, we have used the CASP to facilitate service provisioning for the service clients. We have selected the usage scenarios #2 and #4 to experiment with the platform support.

6.3.1 Support for the medication monitoring scenario

The service provisioning support for the service clients from the usage scenario #2 starts when Jan and Linda inform the Context-Aware Service Platform that the goal *Be medicated* has been selected and the goal should be fulfilled. After receiving the information about which goal it should seek to fulfill, CASP uses the Health Care domain ontology to determine the classification of Jan and Linda. As depicted in Figure 6-9, by having the goal *Be Medicated*, the platform infers that Jan and Linda are instances of the *Alzheimer Patient* concept.

CASP then queries the Ontology Repository for the tasks supporting the received goal using the SPARQL query presented in Listing 6.1.

Listing 6.1 SPARQL query to retrieve tasks supporting the BeMedicated goal

```
PREFIX ↩
    ↪ rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ↩
    ↪ owl2xml:<http://www.w3.org/2006/12/owl2-xml#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdf:<http://www.w3.org/1999/02/ ↩
    ↪ 22-rdf-syntax-ns#>
PREFIX gso:<http://www.utwente.nl/ewi/trese/ ↩
    ↪ olavol/gso.owl#>
```

```
PREFIX healthADO:<http://www.utwente.nl/ewi/ ↩
    ↪ trese/olavol/healthDO.owl#>

SELECT ?tasks
WHERE {
  ?tasks gso:supports healthDO:BeMedicated .
}
```

In this SPARQL query, the list of *PREFIX* elements represents the namespace for the query, i.e., the query takes into consideration the concepts defined in the documents referenced in the prefix list. In this list we have included the file *gso.owl*, which represents the domain specification language based on the GSO, and *healthADO.owl*, which represents the domain ontology of the health care domain. The query stores in variable *?tasks* the instance elements that are related to the element *BeMedicated* from the *healthDO* domain ontology, through the relation *supports*, which is defined in the *gso* prefix.

Since this query does not return any task, CASP tries to find whether the *BeMedicated* goal can be decomposed by querying the Ontology Repository for sub-goals that are related to the *BeMedicated* goal through AND or OR composition relations. Listings 6.2 and 6.3 present the SPARQL queries used by CASP to retrieve from the Ontology Repository the goals that have AND or OR composition relations with the *BeMedicated* goal, respectively. Since these queries have the same prefixes as the query presented in Listing 6.1, we have omitted the list of prefixes in these queries.

Listing 6.2 SPARQL query to retrieve AND sub-goals of the BeMedicated goal

```
SELECT ?goals
WHERE {
  ?goals gso:ANDcomposition ↩
      ↪ healthDO:BeMedicated .
}
```

Listing 6.3 SPARQL query to retrieve OR sub-goals of the BeMedicated goal

```
SELECT ?goals
WHERE {
  ?goals gso:ORcomposition healthDO:BeMedicated .
}
```

As depicted in Figure 6-6, the health care domain ontology defines that the *Be Medicated* goal is a complex goal and is decomposed into the *Self medication* and *Assisted medication* sub-goals through an OR composition relation. Therefore, the query presented in Listing 6.2 does not return any result, while the query presented in Listing 6.3 returns the goals *Self medication* and *As-*

sisted medication. For each of these sub-goals, CASP queries the Ontology Repository for tasks supporting these goals. If a complex task is found, CASP queries the Ontology Repository for sub-tasks that have AND or OR task composition relations with the complex task. This process goes recursively until there are no more tasks supporting the goal and no more sub-tasks to be found. The queries for tasks supporting the sub-goals of the *BeMedicated* goal return the tasks depicted in Figure 6-11.

The discovered tasks are used by the Service Requester component to create the service request. This service request is submitted to the Service Finder component to search for candidate service to fulfill the service clients goals.

In our prototype implementation we have defined that service providers should semantically annotate their service descriptions, including information about which goals the services fulfill or which tasks defined in the domain ontology their services perform. For instance, in our Health Care domain ontology, a health insurance company can inform that their services fulfill the *Get Medical Insurance* goal or inform that their services perform the *Provide Medical Insurance* task.

While the services providers register their service descriptions to the software platform, the CASP extends the domain ontology by adding the registered services. Figure 6-14 depicts the services added to the health care domain ontology related to the medication monitoring scenario. In this experiment, we have informed the platform that the registered services perform the tasks related to medication monitoring depicted in Figure 6-11.

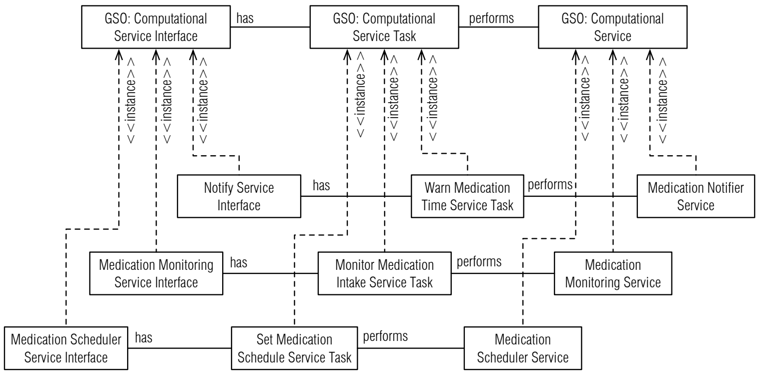
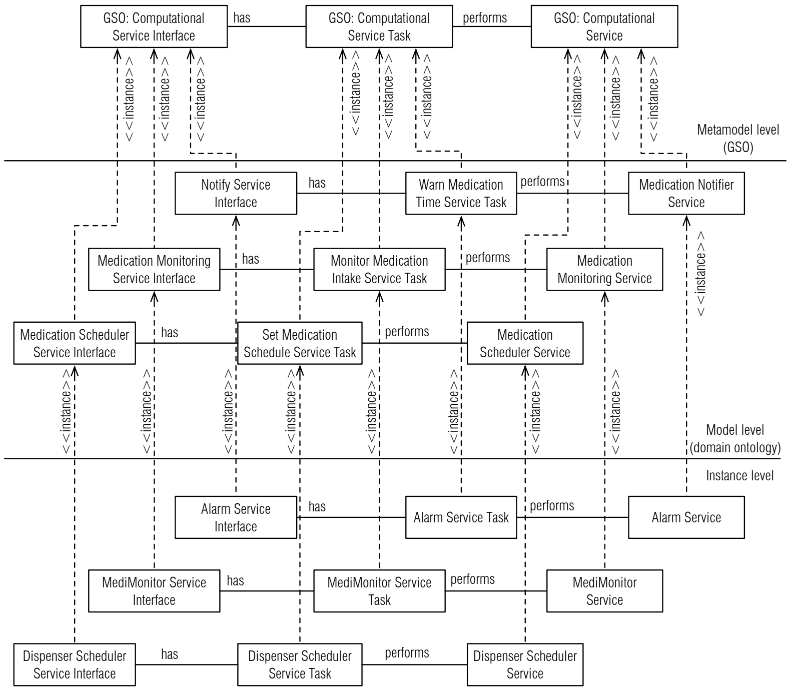

GSO: Computational Service Interface
has
GSO: Computational Service Task
performs
GSO: Computational Service
<<instance>>
Notify Service Interface
has
Warn Medication Time Service Task
performs
Medication Notifier Service
Medication Monitoring Service Interface
has
Monitor Medication Intake Service Task
performs
Medication Monitoring Service
Medication Scheduler Service Interface
has
Set Medication Schedule Service Task
performs
Medication Scheduler Service

Figure 6-14 An excerpt of the domain ontologies depicting the services related to the medication monitoring scenario

In order to test our framework against the usage scenario, we have created a set of services, and registered the semantically annotated service descriptions to the CASP. At runtime, when service providers register their services to the CASP and establish the relation between their offered services and the goals or tasks defined in the domain ontology, the CASP infers the instance level of the domain ontology by relating the individual entities to the concepts at the domain ontology level (i.e., at the model level). The CASP can complete the domain model depicted in Figure 6-14 with the instance-level elements, i.e., the individual services. Figure 6-15 depicts how the CASP extends the model depicted in Figure 6-14 by adding the instance level containing the actual services registered to the platform and their related service providers.

Figure 6-15 The model of services offered to the CASP for the medication monitoring scenario


GSO: Computational Service Interface
has
GSO: Computational Service Task
performs
GSO: Computational Service
<<instance>>
Metamodel level (GSO)
Notify Service Interface
Warn Medication Time Service Task
Medication Notifier Service
Medication Monitoring Service Interface
Monitor Medication Intake Service Task
Medication Monitoring Service
Medication Scheduler Service Interface
Set Medication Schedule Service Task
Medication Scheduler Service
Model level (domain ontology)
Instance level
Alarm Service Interface
Alarm Service Task
Alarm Service
MediMonitor Service Interface
MediMonitor Service Task
MediMonitor Service
Dispenser Scheduler Service Interface
Dispenser Scheduler Service Task
Dispenser Scheduler Service

The CASP's Service Finder component queries the Ontology Repository for services complying with the service request. The service request contains information about the tasks that the candidate services should perform, and the domain ontology has been dynamically extended with the services registered to the platform and the information regarding how these services' tasks relate to the tasks defined in the ontology to satisfy the goals. Therefore, the query returns the service depicted in Figure 6-15.

The service descriptions of the *Dispenser Scheduler Service* and the *MediMonitor Service* require information about which medicine is to be scheduled in the dispenser, and the intake of which medicine the service monitors, respectively. Before asking directly to the service client, CASP queries the Context Manager component for information about Jan and Linda's medication. The electronic medical records system, which stores Jan and Linda's medical information is registered to the CASP as a context provider. Therefore, the Context Manager subscribes to the contextual information of the medicines they should take and their schedules.

Since the discovered services are computational services, the CASP is able to directly invoke their execution. With the contextual information regarding Jan and Linda's medication schedules, the CASP's Service Invoker component invokes the *Dispenser Scheduler Service*, which configures the electronic medicine dispenser to only dispense the specific amount of medicine at a given time. The *Alarm Service* is also invoked to warn Jan and Linda a few minutes before the time for their medicine. Finally, the *MediMonitor Service* is invoked to verify whether they take their medicine correctly and call their caregiver otherwise.

6.3.2 Support for the traveler patient scenario

In our usage scenario #4, the traveler patient informs the Context-Aware Service Platform that the goal *Get medical consultation* has been selected and the goal should be fulfilled.

After receiving the information about which goal it should seek to fulfill, the CASP queries the Ontology Repository for the tasks supporting the received goal using the SPARQL query presented in Listing 6.4.

Listing 6.4 SPARQL query to retrieve tasks

```
PREFIX ↩
    ↪ rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ↩
    ↪ owl2xml:<http://www.w3.org/2006/12/owl2-xml#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdf:<http://www.w3.org/1999/02/ ↩
    ↪ 22-rdf-syntax-ns#>
PREFIX gso:<http://www.utwente.nl/ewi/trese/ ↩
    ↪ olavol/gso.owl#>
PREFIX healthADO:<http://www.utwente.nl/ewi/ ↩
    ↪ trese/olavol/healthDO.owl#>

SELECT ?tasks
WHERE {
  ?tasks gso:supports ↩
      ↪ healthDO:GetMedicalConsultation .
}
```

This SPARQL query stores in variable *?tasks* the instance elements that are related to the element *GetMedicalConsultation* from the *healthDO* domain ontology, through the relation *supports*, which is defined in the *gso* prefix.

In this case study we assume that physicians have registered their services to the CASP informing which tasks their services perform or which goals their services fulfill. While the services providers register their service descriptions to the software platform, the CASP extends the domain ontology by adding the registered services. Figure 6-16 depicts the services added to the health care domain ontologies that are offered by physicians, namely *Medical Consultation* and *Medical Treatment* which can be specialized into *Surgery*.

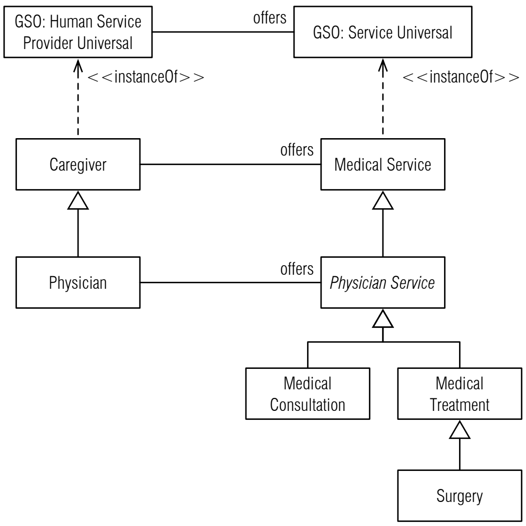


Figure 6-16 An excerpt of the medical services added to the health care domain ontology

Figure 6-17 depicts some of the services and their respective service providers that we have created for this case study. In this figure we have represented the individual physicians and their services as UML object elements to denote that these entities are at the instance level, instantiating the concepts defined in the health care domain ontology. In the objects that represent medical consultations and surgery we have included the information about the location where the service is delivered and the which insurance company policies are supported. However, in the actual services, this information is available to the CASP in the services' descriptions.

After inferring the instance level of the domain ontology by

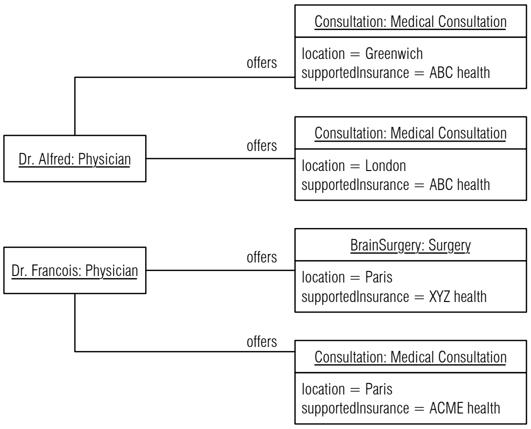

Consultation: Medical Consultation
location = Greenwich
supportedInsurance = ABC health
offers
Dr. Alfred: Physician
offers
Consultation: Medical Consultation
location = London
supportedInsurance = ABC health
Dr. Francois: Physician
offers
BrainSurgery: Surgery
location = Paris
supportedInsurance = XYZ health
offers
Consultation: Medical Consultation
location = Paris
supportedInsurance = ACME health

Figure 6-17 The services offered by physicians registered to the CASP

relating the individual entities to the concepts at the domain ontology level, Figure 6-18 depicts how the CASP extends the model depicted in Figure 6-16 by adding the instance level containing the actual services registered to the platform and their related service providers.

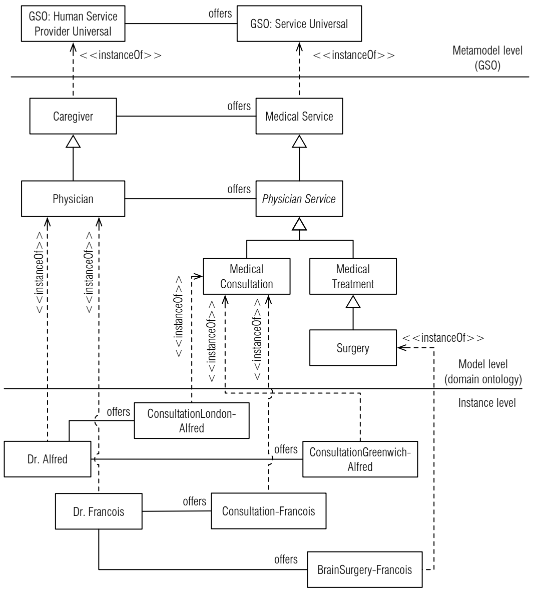


Figure 6-18 The model of services offered by physicians registered to the CASP

As depicted in Figure 6-17, the service descriptions contain information about the location where the service is provided. More specifically, the service grounding of the consultation services specifies that the service requires *Location* and *SupportedInsurance* as input parameters. The CASP uses these parameters as selection criteria when searching for the most suitable service for the service client's goal.

In Figure 6-17 *Dr. Alfred* provides the consultation service in two different locations, namely, London and Greenwich, and accepts the *ABC Health* insurance. When *Lucia*, the traveler patient from our usage scenario, travels to London and requires a medical consultation, the CASP uses her current location as search criteria. Additionally, *Lucia* has a travel insurance, which is related to the *ABC Health* health insurance company. Therefore, the CASP uses the SPARQL query presented in Listing 6.5 to search for consultation services located in London and that accept her travel insurance. The result of this query is the medical consultation service provided by the service provider *Dr. Alfred* in London.

Listing 6.5 SPARQL query to retrieve consultation services in London

```
SELECT ?services
WHERE {
  ?services rdf:type healthDO: ↩
      ↪ MedicalConsultationServiceGrounding .
  ?services gso:hasLocation "London" .
  ?services gso:SupportedInsurance "ABC Health" .
}
```

The nature of the medical consultation service is that there is a person who plays the role of a physician and this person provides the consultation. Therefore, we have defined the medical consultation as an instance of the GSO's *Social Service Universal*. Being a social service, the medical consultation service grounding describes how to activate the service by means of contact information for making an appointment for the consultation. A computational service grounding would, instead, describe some computational interface including URIs, protocols, data types, etc.

In some cases, social services do not have computational counterparts, i.e., computational services that can automate the social services. The medical consultation services from this case study are examples of social services that cannot (yet) be automated by computational services. In these cases, CASP is not able to automate all the service provisioning events, such as service activation, triggering, etc., and should return to the service client the list of candidate services to fulfill the client's goal.

However, before sending the service discovery results to the service client, CASP searches for computational services that can facilitate some of the service provisioning events. While our medical consultation services cannot be automated by a computational service, the medical consultation's activation (i.e., making the appointment for the consultation) can be facilitated by a computational service.

Figure 6-19 depicts an additional computational service registered to the platform named *Medical Consultation Booking*, which is responsible for making appointments for medical consultations. This computational service has a service task named *Medical Consultation Booking Task* that activates the *Medical Consultation* social service.

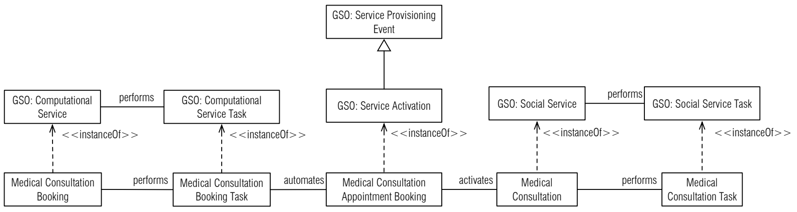

GSO: Service Provisioning Event
GSO: Computational Service
performs
GSO: Computational Service Task
GSO: Service Activation
GSO: Social Service
performs
GSO: Social Service Task
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
Medical Consultation Booking
performs
Medical Consultation Booking Task
automates
Medical Consultation Appointment Booking
activates
Medical Consultation
performs
Medical Consultation Task

Figure 6-19 Computational service support for a social service

After finding a computational service that can automate the booking of a medical consultation, CASP triggers the execution of the *Medical Consultation Booking* and returns to the service client the schedule of the consultation appointment.

6.4 Evaluation of CASP's service provisioning support

In order to evaluate whether our proposed approach is suitable for supporting dynamic service provisioning, we have identified the steps one should take to have a medical consultation service provided for a traveler patient without the support of a software platform. Then, for each of the identified steps, we have verified whether the CASP supports the step, and in case support has been provided, how the CASP supports the step.

Step 1: Identification of the need for a service.

The service client normally identifies that he needs a service when there is something he wants achieved and he is not able to achieve it (only) by himself. In our case study, the traveler patient Lucia is not feeling well, since she is not able to perform a medical evaluation on her condition, she identifies the need for a medical consultation.

This step is supported by the CASP by allowing the service client to inform about a goal she wants fulfilled. In the case study, our prototype implementation of the CASP required direct interaction with the service client to inform her goal.

Step 2: Service discovery.

After identifying the need for a service, the service client should verify whether this service is offered, and by whom. The service client tries to discover the actual services that are available in the market based on some criteria. For instance, flight travelers seeking for cheap flights normally search for the availability of tickets on a given route based on criteria such as ticket price and flight schedule. For these travelers, the airline (the service provider) is less important than finding a good price for the ticket.

In the medication monitoring scenario, CASP supports service discovery by discovering in the domain ontology the tasks that support the Alzheimer's patient *BeMedicated* goal. Since this goal does not have defined any supporting task, CASP searches for sub-goals of the *BeMedicated* goal. After finding the sub-goals, CASP searches for the tasks that support each of these sub-goals.

With the list of tasks supporting the sub-goals of the *BeMedicated* goal, CASP searches for services registered to the platform that perform these tasks.

In the case of our medical consultation service, the traveler patient is first interested in having the service delivered in the same city where she is located. To determine the location where the medical consultation is going to be delivered, a valid assumption is that the service is delivered in the physician's office location.

CASP supports this step by querying the domain ontology on the tasks that support the traveler patient's goal. With the list of the goal-supporting tasks, CASP searches for services that perform these tasks, and that offer this service in London, her current location. In our prototype, the step is carried out automatically, not requiring any interaction with the service client.

Step 3: Service selection.

After discovering a set of candidate services, service clients can classify or filter the services according to certain suitability criteria and, based on this classification, be able to determine the services that better suit their needs.

For our medical consultation service, the traveler patient prefers the service that she can pay using her medical insurance policy, and that can be scheduled for one of her available time slots.

CASP supports this step by filtering the list of available services with the additional criterion of supporting the health insurance policy of the traveler patient (ABC Health). The information about which health insurance policy the traveler patient has, can be gathered by the context-aware components of the CASP.

Step 4: Service activation.

Since the CASP identified that the medical consultation is inherently a social service, i.e., the service cannot be automated by a computational service, the platform searches for services that can facilitate one of the medical consultation's service provisioning events. This search finds the *Medical Consultation Booking*, which facilitates the activation of the medical consultation service. Using the CASP's context-aware components, the time slots of her agenda have been gathered and the CASP invoked the *Medical Consultation Booking* service using the list of candidate physicians and the traveler patient's available time slots as parameters.

Step 4: Service invocation.

In the medication monitoring usage scenario, the discovered services are computational services. Therefore, CASP is able to directly invoke the services. Moreover, in the services descriptions a set of required input information is defined. CASP uses its context-aware components to try to gather this information instead of requesting the information from the service clients. In the case study, CASP uses the gathered contextual information and uses this information as input in the service invocation.

Chapter 7

Conclusion

In this chapter we present the conclusions of the work presented in this thesis, and we identify the topics that we recommend for future work. This chapter is further structured as follows: Section 7.1 presents some general considerations; Section 7.2 elaborates on the most important research contributions of this thesis; and finally, Section 7.3 discusses future work.

7.1 General Considerations

In scenarios where service users have to interact with a potentially large numbers of services through a variety of devices, we have observed that facilities to provide support for these interactions are necessary. Additionally, if we consider non-technical service users, this support should also allow these users to express their service requests in terms that are closer to their common conceptualization than technical terms such as data types and document format. In this thesis we have defined a conceptual framework named Goal-Based Service Framework, which facilitates service provisioning and allows service users to express their service requests in terms of goals.

Software platforms can support service users in discovering, selecting, negotiating and invoking services. To have services provisioned, service users need to interact with the service platform to provide information such as criteria for service discovery, selection and composition, triggering conditions and inputs for service execution. Some of this information relates to contextual information, i.e., information characterizing the service user's context. We have proposed the integration of context-aware components aiming at transparently gathering users' contextual information to be used on service discovery, selection and composition as well as for supplying input information for service executions.

In order to be more effective in supporting the service provisioning activities, the software platform should have the means to understand the service user's requests and the service descriptions. This understanding allows the software platform to reason about the terms included in the user requests and service descriptions, and achieve not only syntactic matches between terms in these artifacts, but also semantic matches. Semantic matching and reasoning can be realized by means of semantic annotations on the terms contained in service requests and service descriptions. These annotations are based on semantic domain specifications, expressed in terms of domain ontologies.

The specification of domain ontologies requires that the domain specialists utilize an ontology representation language that has primitives capable of representing concepts and relations of different domains, i.e., the language should be domain-independent. An ontology representation language should be grounded on a meta-ontology that describes a set of real-world categories that can be used to talk about reality [Masolo et al., 2003, Guizzardi, 2005, 2006]. This meta-ontology (or domain-independent theory of real-world categories) is called a *foundational ontology* [Guizzardi, 2007].

We have observed that, on one hand, we have ontology representation languages based on foundational ontologies such as UFO [Guizzardi, 2005], DOLCE [Gangemi et al., 2002] or GOL [Heller and Herre, 2003, Degen et al., 2001], which define concepts and relations to describe application domains, but lack support for describing service-related concepts directly. On the other hand, we have ontology languages such as the Semantic Markup for Web Services (OWL-S) [Martin et al., 2004], the Web Service Modeling Ontology (WSMO) [de Bruijn et al., 2006], OWL-S, or the Open Group's Service-Oriented Architecture Ontology (SOA Ontology) [Ope, 2010], which define concepts and relations to describe services, but lack support for describing the concepts related to the domain in which the service should operate.

Adding service concepts as first class constructs to ontology representation languages allows the specification of domain ontologies containing service-related concepts and relations. Domain ontologies containing service-related concepts allows software platforms to understand the context in which the described services are inserted, and to reason about the relations between the services, its components and other elements of the domain. Moreover, in increasing complex scenarios, we have identified the need for a clear distinction between social and computational services, and the specification of the relations between services at these two levels.

7.2 Research Contributions

The main contributions of this thesis can be summarized as follows:

- A conceptual framework for semantic service provisioning;
- A foundational ontology containing concepts and relations to allow the specification of application domains including service-related concepts, in which the distinction between social and computational services is acknowledged and supported;
- A software platform to support service provisioning, with context-aware capabilities to allow the use of contextual information to reduce the need for direct interaction with the service users.

These contributions have been developed during the effort to address the research questions we have identified (see Chapter 1). In the sequel we recall the research questions and discuss how they have been answered.

RQ1: What are the best practices in using the concept of goal to capture user requirements?

To answer this research question we have conducted the literature investigation reported in Chapter 2, for the uses and definitions of the goal concept. With this investigation we have gained the necessary understanding to be able to reuse and extend the most suitable uses and definitions for the goal concept, adapting them to the objectives and scope of our work.

RQ2: Which components can be devised to provide facilities for service provisioning in multiple domains?

To answer this research question we have defined a conceptual framework for semantic service provisioning. This framework, discussed in Chapter 3, defines three abstraction layers, namely the software platform support, the domain knowledge and the foundational ontology layers.

The software platform supports services clients and service providers in activities related to service provisioning. The domain knowledge layer represents the information about application domains that is used by the software platform to understand and to perform semantic reasoning in the terms used in service descriptions and requests. The foundational ontology layer represents the conceptualization that underlies the elements defined in the domain knowledge level.

We have developed concrete components to realize the elements in each of these three abstraction layers. The software platform support is realized by the Context-Aware Service Platform, the domain knowledge is realized by the domain ontologies, the domain specification language is realized by Goal-Based Domain Specification Language, and the foundational ontology is realized by the Goal-Based Service Ontology.

> **RQ3:** Which techniques are capable of supporting and enabling domain specifications and semantic annotations?

The support for domain specifications and semantic annotations have been addressed in our work with the definition of our foundational ontology. We have identified the need for integrating service-related concepts with the other domain-specification concepts. Therefore, we have investigated foundational ontologies and service ontologies aiming at finding ways of integrating them. From this investigation we have chosen the Unified Foundational Ontology (UFO) as base foundational ontology, and have extended UFO with service-related concepts.

The definition of the service-related concepts to extend UFO have been conducted after an investigation of the areas of Foundational Ontologies, Semantic Web Services and Service-Oriented Computing. In this investigation of the service-related concepts, we aimed at capturing the nature of these concepts and avoiding commitments with specific technologies or implementation strategies. In other words, we intended to understand what a service is about, what characterizes a service, and which other concepts are related to a service. This investigation resulted in the Goal-Based Service Ontology (GSO). In GSO we have also defined a distinction between social and computational services, and defined relations between these two types of services. The benefits of this distinction has been demonstrated in our case study, presented in Chapter 6.

From GSO we have derived a domain specification language metamodel, which provides the abstract syntax for the Goal-Based Domain Specification Language (GDSL). This language allows domain specialists to define domain specifications in terms of domain ontologies. These domain ontologies are used by service providers and context providers to semantically annotate the terms in their service descriptions and context information descriptions, respectively, and by service clients to express their service requests in terms of goals.

> **RQ4:** How to structure a platform to support dynamic service provisioning using domain specifications and semantic annotations?

To define the architectural design of our software platform, we have identified a set of requirements for the platform. These requirements (discussed in Section 5.1) specialize the more general requirements for the service provisioning framework (discussed in Section 3.4), and focus on how the software platform can automate some of the processes related to service provisioning. We have also defined that the software platform should support not only service clients, but also service providers, context providers, and domain specialists. These other users of the platform should be able to semantically annotate the terms in their service descriptions (service providers) and context information descriptions (context providers), and be able to define and update domain ontologies written using GDSL (domain specialists).

The requirements for each of these platform users influenced the definition of the platform's architectural components. As discussed in Chapter 5, we have defined the architectural design of the Context-Aware Service Platform (CASP), and implemented some of its components in our prototype. The prototype allowed us to demonstrate the suitability of the approach in the case study discussed in Chapter 6.

> **RQ5:** How can we assert whether our proposed framework satisfied the objectives of our research?

The evaluation of the framework was mainly performed during the case study presented in Chapter 6. In this case study we defined a simplified domain ontology for the health care domain using GDSL. Our health care ontology demonstrated that the GDSL concepts allow domain specialists to specify application domains that contain service-related concepts. Moreover, our case study included the medical consultation service, which is an example of a social service. With this service we demonstrated that, although a social service cannot be automated by a computational service, computational services can still facilitate the consumption of the social service, by automating some of the social service's provisioning events. In particular, our case study demonstrated the automation of the medical consultation's service discovery, selection and activation.

The case study also demonstrated the CASP's support to service provisioning. In the case study, our prototype receives the information of the goal the service client wants achieved, e.g., the traveler patient informed that she wants a medical consultation in the city where she is at the moment, and the Alzheimer's patients informed that they want assistance to take their medications. The CASP then uses the health care domain ontology to identify tasks that could support the clients' goals, and searches for services that perform those tasks. Once a set of candidate services has been found, CASP tries to select the most appropriate service(s). In the traveler patient usage scenario, the service selection if performed using as criteria the city where the client is located, the health insurance policy used by the client, and her available time slots for the consultation. The information used to evaluate these criteria can be gathered by the CASP through the available context sources.

In the medication monitoring usage scenario, since the discovered services are computational services, CASP invokes the services on behalf of the service clients and uses contextual information for the services' inputs.

In the traveler patient usage scenario, once the candidate services have been filtered, CASP identifies that the service is a social service, and searches for ways to automate not the service itself, but the service provisioning events related to this service. CASP has been able to find a service to book appointments for medical consultations, and invoked this booking service to automate the activation event of the medical consultation service.

Finally, we have identified the steps that a service client would need to perform to have service provisioned without our framework's support. For each of these identified steps we pointed up how our approach supports the service client. Therefore, we could demonstrate how our approach is suitable to facilitate service provisioning.

7.3 Directions for Future Research

During the research we have conducted in the scope of this thesis, we made choices concerning the scope and focus of this research. We focus on an integrated solution for supporting service provisioning for non-technical service clients, but we certainly did not cover all aspects related to this topic. While we developed the framework and implemented our prototype, we have identified points for improvements, and other topics and aspects that would be complementary to our work. In the sequel we present some of these points, suggesting directions for future research.

Software platform

In our prototype we experimented with the service provisioning support on service discovery, selection, composition and activation. Service triggering has been tested only up to the moment of the identification that the triggering conditions hold, and that the service execution should be triggered. However, we did not invoke the services in our prototype. The implementation of the SAWSDL standard that we used does not support semantic annotations on the binding component of the WSDL. Therefore, strategies to provide semantic annotations to the WSDL's binding component should be investigated, as well as how the CASP should support service execution triggering. In some cases, after the service execution is triggered, adaptation should be performed on the service results to adjust the output of the service to the results the service client is expecting. In a simple example, an online purchasing service returns the price and delivery costs in US dollars, while the service clients expects the amount to the expressed in Euros. In this case, the software platform could perform the adaptation of the results by, for instance, invoking a currency conversion service.

In our CASP prototype, we have implemented some core components of the architecture, such as the Ontology Repository and the context-aware components. However, in the experiment case study, we manually manually integrated the components. Therefore, some work should be performed to automate the integration of the CASP's components.

Modeling editor

In our prototype we have implemented a simple editor for specifying domain ontologies. We have also identified the requirements and have defined the architectural design for the tool support for domain specialists and language designers. However, these tools are not complete. The prototype of the editor generation tool, which takes a foundational ontology, generates a language metamodel from it, and creates a graphical editor for this language, is able to demonstrate the feasibility of the approach, but further work to have an usable tool is still needed to make the tool stable and generally usable.

Moreover, the features to check whether a modification on the foundational ontology creates undesirable effects on the model already created from the previous version of the derived language needs to be implemented, and further work on the mappings between EMF and OWL should be performed. These mappings are used to allow the transformation of the foundational ontology, written in OWL, into EMF-based metamodels.

Foundational ontology

In our work, we defined the GSO as the foundational ontology behind the language to support specialists in specifying domains by means of domain ontologies. GSO has been designed as an extension of the Unified Foundational Ontology and the model created with the language, namely the domain ontologies, are used to semantically annotate service descriptions, contextual information and context source descriptions as well as to support the CASP to search for tasks and services that can fulfill goals. In our prototype implementation described in Chapter 5, and in the case study of our proposed framework discussed in Chapter 6, we have created the domain ontologies by instantiating the concepts and relations defined in GSO and representing the domain ontologies using OWL.

However, during our experiments we observed that the CASP's inference capabilities would be improved if we added some information to the elements of the domain ontologies. For instance, if we define *Traveling abroad* as an instance of the GSO's concept *Situation*, we would be able to represent the situation where a given agent is currently in a trip and is located outside its home country. Therefore, it would be useful to include in this concept (the Traveling abroad situation) the constraints that define the situation, e.g., $TravelAbroad \doteq \exists Agent(x) \wedge travel(x) \wedge currentLocation(x, \neg(homeCountry(x)))$. To accomplish this, the language should include properties to the *Situation* concept, allowing designers to express these constraints.

In our experiments we have defined a health care domain ontology using the domain specification language derived from our foundational ontology. In the experiments we have been able to properly model the domain using the concepts and relations defined in the foundational ontology. However, further investigations in other application domains may elicit possibilities to improve the domain specification language, and lead to requirements to extend the foundational ontology.

Bibliography

A-muse: Architectural modeling for service enabling in freeband. `http://a-muse.freeband.nl/`.

Amigo: Ambient intelligence for the networked home environment. `http://www.hitech-projects.com/euprojects/amigo/`.

Eclipse Modeling Framework Project. `http://www.eclipse.org/modeling/emf/?project=emf`.

Eclipse Graphical Modeling Framework Project. `http://www.eclipse.org/gmf/`.

JESS - the rule engine for the java platform. `http://herzberg.ca.sandia.gov/jess/`.

OMG's MetaObject Facility. `http://www.omg.org/mof/`.

U-care project. `http://ucare.ewi.utwente.nl/`.

Web services description language (wsdl) version 2.0 part 0: Primer. `http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/`.

Reference Model for Service Oriented Architecture 1.0. Oasis standard, OASIS, October 2006. URL `http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf`.

Service-oriented architecture ontology. Draft technical standard 2.0, The Open Group, July 2008. URL `http://www.opengroup.org/projects/soa-ontology/uploads/40/16940/soa-ontology-200-draft.pdf`.

Wsml language reference v. 1.0. Final draft, WSML working group, August 2008. URL `http://www.wsmo.org/TR/d16/d16.1/v1.0/`.

Smart hoMes for All - SM4All. http://www.sm4all-project.eu/, 2008-2011.

Owl 2 web ontology language primer. http://www.w3.org/TR/2009/REC-owl2-primer-20091027/, October 2009.

Service-Oriented Architecture Ontology. Technical standard, The Open Group, October 2010.

Knopflerfish OSGi Service Platform. http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/, February 2011.

OSGi Alliance. http://www.osgi.org/, February 2011.

Semantic Web. http://http://semanticweb.org//, February 2011.

jGraphT. http://jgrapht.sourceforge.net/, 2011.

Vikas Agarwal, Koustuv Dasgupta, Neeran M. Karnik, Arun Kumar, Ashish Kundu, Sumit Mittal, and Biplav Srivastava. A service creation environment based on end to end composition of web services. In Allan Ellis and Tatsuya Hagino, editors, Proceedings of the 14th international conference on World Wide Web (WWW 2005), pages 128–137, New York, NY, USA, 2005. ACM.

R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. Web Service Semantics - WSDL-S, a joint UGA-IBM technical note, version 1.0. Technical report, University of Georgia and IBM, April 2005.

João Paulo A. Almeida, Alberto Baravaglio, M. Belaunde, P. Falcarin, and E. Kovacs. Service creation in the spice service platform. In Proceedings of the 17th Wireless World Research Forum Meeting (WWRF17), Heidelberg, Germany, November 2006.

Annie I. Anton. Goal-based requirements analysis. In Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96), pages 136–, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7252-8. URL http://portal.acm.org/citation.cfm?id=850944.853130.

Annie I. Anton. Goal identification and refinement in the specification of software-based information systems. PhD thesis, Atlanta, GA, USA, 1997. UMI Order No. GAX97-35409.

Apache. Apache jUDDI. `http://ws.apache.org/juddi`, 2008.

Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, and Kishore Channabasavaiah. Design an SOA solution using a reference architecture, March 2007. URL `http://www-128.ibm.com/developerworks/library/ar-archtemp/index.html`.

Ali Arsanjani, Nikhil Kumar, Chris Harding, Mats Gejnevall, Tony Carrato, Heather Kreger, and Jorge Diaz. Soa reference architecture. Technical report, The Open Group, http://www.theopengroup.org/projects/soa-ref-arch/, September 2009.

Carliss Y. Baldwin and Kim B. Clark. Design Rules: The Power of Modularity, volume 1. MIT Press, Cambridge, MA, USA, March 2000. URL `http://www.amazon.com/Design-Rules-Vol-Power-Modularity/dp/0262024667`.

Alistair Barros, Marlon Dumas, and Peter Bruza. The move to web service ecosystems. BPTrends, November 2005.

Sean Bechhofer, Raphael Volz, and Phillip Lord. Cooking the semantic web with the owl api. In International Semantic Web Conference, pages 659–675. Springer, 2003.

Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. Scientific American, May 2001.

L. O. Bonino da Silva Santos, L. Ferreira Pires, and M. J. van Sinderen. Service provisioning support for non-technical service clients. In Proceedings of the 7th International Conference on Information Technology, Las Vegas, USA, pages 672–677, Los Alamitos, April 2010a. IEEE Computer Society Press.

L. O. Bonino da Silva Santos, V. S. Sorathia, L. Ferreira Pires, and M. J. van Sinderen. An approach to dynamic provisioning of social and computational services. In Proceedings of the 6th IEEE Congress on Services, Miami, FL, USA, pages 24–31, Los Alamitos, July 2010b. IEEE Computer Society Press.

Luiz Olavo Bonino da Silva Santos, Luís Ferreira Pires, and Marten J. van Sinderen. Architectural models for client interaction on service-oriented platforms. In Marten van Sinderen, editor, 1st International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2007), pages 19–27. INSTIC, July 2007.

Luiz Olavo Bonino da Silva Santos, Giancarlo Guizzardi, Renata Silva Souza Guizzardi, Eduardo Gonçalves da Silva, Luís Ferreira Pires, and Marten J. van Sinderen. Gso: Designing a well-founded service ontology to support dynamic service discovery and composition. In 2nd International Workshop on Dynamic and Declarative Business Process (DDBP 2009), September 2009.

David Booth, Hugo Haas, Francis G. McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. `http://www.w3.org/TR/ws-arch/`, February 2004.

M. E. Bratman. Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, MA, 1987.

P. Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems, 8(3):203–236, 2004. URL `http://citeseer.ist.psu.edu/bresciani02tropos.html`.

R. J. A. Buhr and R. S. Casselman. Use case maps for object-oriented systems. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-456542-8.

Steve Burbeck. The tao of e-business services: The evolution of web applications into service-oriented components with web services. Online document, October 2000.

Christoph Bussler, Emilia Cimpian, Dieter Fensel, Juan Miguel Gomez, Armin Haller, Thomas Haselwanter, Michael Kerrigan, Adrian Mocan, Matthew Moran, Eyal Oren, Brahmananda Sapkota, Ioan Toma, Jana Viskova, Tomas Vitvar, Maciej Zaremba, and Michal Zaremba. Web Service Execution Environment (WSMX). W3C Member Submission, June 2005. URL `http://www.w3.org/Submission/WSML/`.

Cristiano Castelfranchi and Rino Falcone. Towards a theory of delegation for agent-based systems. Robotics and Autonomous Systems - Special issue on Multi-Agent Rationality, 24(3–4): 141–157, 1998.

Tiziana Catarci, Febo Cincotti, Massimiliano Leoni, Massimo Mecella, and Giuseppe Santucci. Smart homes for all: Collaborating services in a for-all architecture for domotics. In Elisa

Bertino, James B. D. Joshi, Ozgur Akan, Paolo Bellavista, Jiannong Cao, Falko Dressler, Domenico Ferrari, Mario Gerla, Hisashi Kobayashi, Sergio Palazzo, Sartaj Sahni, Xuemin (Sherman) Shen, Mircea Stan, Jia Xiaohua, Albert Zomaya, and Geoffrey Coulson, editors, Collaborative Computing: Networking, Applications and Worksharing, volume 10 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 56–69. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-03354-4. URL `http://dx.doi.org/10.1007/978-3-642-03354-4_6`.

B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What are Ontologies, and why do we need them? IEEE Intelligent Systems, 14:20–26, 1999. ISSN 1094-7167. doi: http://doi.ieeecomputersociety.org/10.1109/5254.747902.

Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. Non-Functional Requirements in Software Engineering, volume 5 of The Kluwer International Series in Software Engineering. Springer, 1st edition, October 1999. ISBN 0792386663.

Emilia Cimpian, Adrian Mocan, and Michael Stollberg. Mediation enabled semantic web services usage. In Riichiro Mizoguchi, Zhongzhi Shi, and Fausto Giunchiglia, editors, 1st Asian Semantic Web Conference (ASWC2006), volume 4185 of Lecture Notes in Computer Science, pages 459–473, Beijing, China, September 2006. Springer.

Emilia Cimpian, Harald Meyer, Dumitru Roman, Adina Sirbu, Nathalie Steinmetz, Steffen Staab, and Ioan Toma. Ontologies and matchmaking. In Dominik Kuropka, Peter Tröger, Steffen Staab, and Mathias Weske, editors, Semantic Service Provisioning, chapter 3, pages 19–54. Springer Berlin Heidelberg, Berlin, 2008. ISBN 978-3-54078-616-0. doi: http://dx.doi.org/10.1007/978-3-540-78617-7_3.

Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. Artificial Intelligence, 42:213–261, March 1990. ISSN 0004-3702. doi: 10.1016/0004-3702(90)90055-5. URL `http://portal.acm.org/citation.cfm?id=77754.77757`.

Robert M. Colomb. Ontology and the Semantic Web, volume 156 of Frontiers in Artificial Intelligence and Applications. IOS Press, 2007.

Roberto Confalonieri, John Domingue, and Enrico Motta. Mediation of semantic web services in IRS-III. In Proceedings of the

Workshop on Mediation in Semantic Web Services in conjunction with the 3rd International Conference on Service Oriented Computing, 2005.

Christophe Cordier, François Carrez, Herma Van Kranenburg, Antonietta Spedalieri, and Jean-Pierre Le Rouzic. Addressing the challenges of beyond 3g service delivery: the SPICE service platform. In International Workshop on Applications and Services in Wireless Networks, Berlin, Germany, May 2006.

Eduardo Manoel Gonçalves da Silva. User-centric service composition – towards personalised service composition and delivery. PhD thesis, University of Twente, May 2011.

Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. Science of Computer Programming, 20(1-2):3–50, April 1993. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/0167-6423(93)90021-G. URL `http://dx.doi.org/10.1016/0167-6423(93)90021-G`.

Jos De Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kifer, Birgitta König-Ries, Jacek Kopecky, Rubén Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dumitru Roman, James Scicluna, and Michael Stollberg. Web service modeling ontology (WSMO). W3C Member Submission, 2005. URL `http://www.w3.org/Submission/WSMO-primer/`.

Jos de Bruijn, Dieter Fensel, Uwe Keller, Michael Kifer, Holger Lausen, Reto Krummenacher, Axel Polleres, and Livia Predoiu. Web service modeling language (WSML). W3C Member Submission, 2005. URL `http://www.w3.org/Submission/WSML/`.

Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Michael Kifer, Birgitta König-Ries, Jacek Kopecky, Rubén Lara, Eyal Oren, Axel Polleres, James Scicluna, and Michael Stollberg. Web Service Modeling Ontology (WSMO). WSMO Final Draft, October 2006. URL `http://www.wsmo.org/TR/d2/v1.3/`.

Keith Decker, Mike Williamson, and Katia Sycara. Matchmaking and brokering. In 2nd International Conference in Multi-Agent Systems (ICMAS'96), pages 432–443, December 1996.

Wolfgang Degen, Barbara Heller, Heinrich Herre, and Barry Smith. GOL: toward an axiomatized upper-level ontology. In

Nicola Guarino, Barry Smith, and Christopher Welty, editors, Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS 2001), pages 34–46, New York, NY, USA, 2001. ACM. doi: http://doi.acm.org/10.1145/505168.50517.

Tom DeMarco. Structured Analysis and System Specification. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1979. ISBN 0138543801. URL `http://portal.acm.org/citation.cfm?id=1102012`.

Patricia Dockhorn Costa, Luis Ferreira Pires, and Marten J. van Sinderen. Architectural patterns for context-aware services platforms. In S. Kouadri Mostefaoui and Z. Maamar, editors, Second International Workshop on Ubiquitous Computing, pages 3–18, Portugal, April 2005. INSTICC Press. URL `http://doc.utwente.nl/63460/`.

Patricia Dockhorn Costa, Luis Ferreira Pires, Marten J. van Sinderen, and Tom H.F. Broens. Controlling services in a mobile context-aware infrastructure. In Klaus David and Sandra Haseloff, editors, Proceedings of the Second Workshop on Context Awareness for Proactive Systems, CAPS 2006, pages 153–166, Kassel, Germany, June 2006. Kassel University Press. URL `http://doc.utwente.nl/65633/`.

John Domingue, Dumitru Roman, and Michael Stollberg. Web Service Modeling Ontology (WSMO): an ontology for Semantic Web Services. In W3C Workshop on Frameworks for Semantics in Web Services, pages 776–784. W3C, June 2005. URL `http://www.w3.org/2005/04/FSWS/Submissions/1/wsmo_position_paper.html`.

Wilco Engelsman, Dick Quartel, Henk Jonkers, and Marten van Sinderen. Extending enterprise architecture modelling with business goals and requirements. Enterprise Information Systems, 5:9–36, February 2011. ISSN 1751-7575. doi: http://dx.doi.org/10.1080/17517575.2010.491871. URL `http://dx.doi.org/10.1080/17517575.2010.491871`.

Thomas Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

Christian Facciorusso, Simon Field, Rainer Hauser, Yigal Hoffner, Robert Humbel, René Pawlitzek, Walid Rjaibi, and Christine

Siminitz. A web services matchmaking engine for web services. In Kurt Bauknecht, A. Min Tjoa, and Gerald Quirchmayr, editors, Proceedings of the 4th International Conference on E-Commerce and Web Technologies (EC-Web 2003), volume 2738 of Lecture Notes in Computer Science, pages 37–49. Springer, September 2003.

Ricardo de Almeida Falbo, Giancarlo Guizzardi, and Katia Cristina Duarte. An ontological approach to domain engineering. In Proceedings of the 14th international conference on Software engineering and knowledge engineering (SEKE '02), pages 351–358, New York, NY, USA, 2002. ACM. ISBN 1-58113-556-4. doi: http://doi.acm.org/10.1145/568760.568822.

Joel Farrell and Holger Lausen. Semantic annotations for wsdl and xml schema, August 2007. URL `http://www.w3.org/2002/ws/sawsdl/`.

Dieter Fensel and Christoph Bussler. The web service modeling framework WSMF. Electronic Commerce Research and Applications, 1(2):113–137, Feb 2002. ISSN 15674223. doi: 10.1016/S1567-4223(02)00015-7. URL `http://dx.doi.org/10.1016/S1567-4223(02)00015-7`.

Roberta Ferrario and Nicola Guarino. Towards an ontological foundations for services science. In Dieter Fensel and Paolo Traverso, editors, Proceedings of Future Internet Symposium 2008. Springer Verlag, 2008.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2(3-4):189–208, 1971. doi: 10.1016/0004-3702(71)90010-5. URL `http://dx.doi.org/10.1016/0004-3702(71)90010-5`.

Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94), pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Jörg P. Müller, Michael Wooldridge, and Nicholas R. Jennings, editors, Intelligent Agents III Agent Theories, Architectures, and Languages, volume 1193 of Lecture Notes in Computer Science, chapter 2, pages 21–35–35. Springer Berlin / Heidelberg, Berlin/Heidelberg, 1997. ISBN 3-540-62507-0. doi: 10.1007/BFb0013570. URL `http://dx.doi.org/10.1007/BFb0013570`.

Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with DOLCE. In Asunción Gómez-Pérez and V. Richard Benjamins, editors, Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW '02), volume 2473, pages 166–181, London, UK, 2002. Springer-Verlag. ISBN 3-540-44268-5.

Aldo Gangemi, Peter Mika, and Daniel Oberle. An ontology of services and service descriptions. Technical report, Laboratory for Applied Ontology (ISTC-CNR), November 2003.

Malik Ghallab, Craig K. Isi, Scott Penberthy, David E. Smith, Ying Sun, and Daniel Weld. PDDL - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control, 1998.

Eduardo M. Goncalves da Silva, L. Ferreira Pires, and M. J. van Sinderen. Towards runtime discovery, selection and composition of semantic services. Computer communications, 34(2):159–168, February 2011. ISSN 0140-3664. doi: http://dx.doi.org/10.1016/j.comcom.2010.04.003.

Frank P. Grad. The preamble of the constitution of the world health organization. Bulletin of the World Health Organization, 80(12):981–4, 2002.

R. C. Gronback. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional, 2009. ISBN 0321534077.

Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In Proceedings of the 12th international conference on World Wide Web (WWW '03), pages 48–57, New York, NY, USA, 2003. ACM. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775160. URL `http://doi.acm.org/10.1145/775152.775160`.

Thomas R. Gruber. The role of common ontology in achieving sharable, reusable knowledge bases. In James F. Allen, Richard

Fikes, and Erik Sandewall, editors, Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91), pages 601–602, Cambridge, MA, USA, April 1991. Morgan Kaufmann Publishers.

Thomas R. Gruber. A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2):199–220, 1993. ISSN 1042-8143. doi: http://dx.doi.org/10.1006/knac.1993.1008.

Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies, 43(5-6):907–928, 1995. ISSN 1071-5819. doi: http://dx.doi.org/10.1006/ijhc.1995.1081.

Nicola Guarino. Formal ontology and information systems. In Nicola Guarino, editor, Proceedings of the 1st International Conference on Formal Ontologies in Information Systems (FOIS'98), pages 3–15, Trento, Italy, June 1998. IOS Press.

Christoph Guger, Shahab Daban, Eric Sellers, Clemens Holzner, Gunther Krausz, Roberta Carabalona, Furio Gramatica, and Guenter Edlinger. How many people are able to control a P300-based brain-computer interface (BCI)? Neuroscience Letters, 462(1):94 – 98, 2009. ISSN 0304-3940. doi: DOI:10.1016/j.neulet.2009.06.045. URL `http://www.sciencedirect.com/science/article/pii/S0304394009008192`.

Giancarlo Guizzardi. Ontological Foundations for Structural Conceptual Models. PhD thesis, University of Twente, 2005.

Giancarlo Guizzardi. The role of foundational ontologies for conceptual modeling and domain ontology representation. In Olegas Vasilecas, Johann Eder, and Albertas Caplinskas, editors, Proceedings of the 7th International Baltic Conference on Databases and Information Systems, pages 17 –25, 0-0 2006. doi: 10.1109/DBIS.2006.1678468.

Giancarlo Guizzardi. On ontology, ontologies, conceptualizations, modeling languages, and (meta)models. Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV, 2007.

Giancarlo Guizzardi and Gerd Wagner. Some applications of a unified foundational ontology in business modeling. In Peter

Green and Michael Rosemann, editors, Business Systems Analysis with Ontologies, chapter 13, pages 345–367. IGI Global, 2005.

Giancarlo Guizzardi, Gerd Wagner, Nicola Guarino, and Marten van Sinderen. An ontologically well-founded profile for uml conceptual models. In Anne Persson and Janis Stirna, editors, Proceedings of the 16th International Conference on Advanced Information Systems Engineering, CAiSE 2004, volume 3084 of Lecture Notes in Computer Science, pages 112–126. Springer, June 2004.

Giancarlo Guizzardi, Ricardo Falbo, and Renata Silva Souza Guizzardi. Grounding software domain ontologies in the unified foundational ontology (ufo): The case of the ode software process ontology. In 1th Iberoamerican Workshop on Requirements Engineering and Software Environments (IDEAS'2008), Recife, Brazil, 2008.

Thomas Haselwanter, Paavo Kotinurmi, Matthew Moran, Tomas Vitvar, and Maciej Zaremba. Wsmx: A semantic service oriented middleware for b2b integration. In Asit Dan and Winfried Lamersdorf, editors, Proceedings of the 4th International Conference on Service Oriented Computing, volume 4294, pages 477–483, Chicago, USA, December 2006. Springer-Verlag.

B. Heller and H. Herre. Formal ontology and principles of GOL. Onto-med report nr. 1/2003, Forschungsgruppe Ontologies in Medicine, Universität Leipzig., 2003.

Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. Management Information Systems Quarterly, 28(1):75–106, 2004.

Peter Hill. Tangibles, intangibles and services: a new taxonomy for the classification of output. Canadian Journal of Economics, 32(2):426–446, April 1999. URL `http://ideas.repec.org/a/cje/issued/v32y1999i2p426-446.html`.

Yigal Hoffner, Christian Facciorusso, Simon Field, and Andreas Schade. Distribution issues in the design and implementation of a virtual market place. Computer Networks: The International Journal of Computer and Telecommunications Networking, 32 (6):717–730, May 2000. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/S1389-1286(00)00028-1.

Pavel Hruby. Ontology-based domain-driven design. In Jorn Bettin, Ghica van Emde Boas, Jean Bézivin, Markus Völter, and William Cook, editors, OOPSLA Workshop on Best Practices for Model-Driven Software Development, San Diego, CA, USA, October 2005.

Michael N. Huhns and Munindar P. Singh. Service-oriented computing: Key concepts and principles. IEEE Internet Computing, 9(1):75–81, January 2005. ISSN 1089-7801. doi: 10.1109/MIC.2005.21. URL `http://dx.doi.org/10.1109/MIC.2005.21`.

ISO/IEC-ITU/T. Information technology – open distributed processing – trading function: Specification, 1998. ISO/IEC 13235-1, UIT-T X.950.

ITU-T. ITU-T Rec. Z.151 – Formal description techniques (FDT) – Specification and Description Language (SDL) – User requirements notation (URN) â“ Language definition, November 2008.

Rim Samia Kaabi, Carine Souveyet, and Colette Rolland. Eliciting service composition in a goal driven manner. In Marco Aiello, Mikio Aoyama, Francisco Curbera, and Mike P. Papazoglou, editors, Proceedings of the 2nd international conference on Service oriented computing (ICSOC 04), pages 308–315. ACM, 2004. URL `http://dblp.uni-trier.de/db/conf/icsoc/icsoc2004.html#KaabiSR04`.

Eirini Kaldeli. Using CSP for adaptable web service composition. Technical report, University of Groningen, 2009. URL `http://www.cs.rug.nl/~eirini/tech_rep_09-7-01.pdf`.

Eirini Kaldeli, Alexander Lazovik, and Marco Aiello. Extended goals for composing services. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009). AAAI, September 19-23 2009. ISBN 978-1-57735-406-2.

Eirini Kaldeli, Ehsan Warriach, Jaap Bresser, Alexander Lazovik, and Marco Aiello. Interoperation, composition and simulation of services at home. In Paul Maglio, Mathias Weske, Jian Yang, and Marcelo Fantinato, editors, Service-Oriented Computing, volume 6470 of Lecture Notes in Computer Science, pages 167–181. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-17357-8. URL `http://dx.doi.org/10.1007/978-3-642-17358-5_12`.

Atanas Kiryakov, Borislav Popov, Ivan Terziev, Dimitar Manov, and Damyan Ognyanoff. Semantic annotation, indexing, and retrieval. Web Semantics: Science, Services and Agents on the World Wide Web, 2(1):49–79, December 2004.

Maksym Korotkiy. From Ontology-enabled Services to Service-enabled Ontologies: Making Ontologies Work in e-Science with Onto-SOA. PhD thesis, Vrije Universiteit Amsterdam, June 2009.

Yannis Labrou, Tim Finin, and Yun Peng. Agent communication languages: The current landscape. IEEE Intelligent Systems, 14:45–52, March 1999. ISSN 1541-1672. doi: http://dx.doi.org/10.1109/5254.757631. URL `http://dx.doi.org/10.1109/5254.757631`.

Marc Lankhorst. Enterprise Architecture at Work: Modelling, Communication and Analysis. Springer-Verlag, Berlin, 2. edition, 2009. ISBN 978-3-642-01309-6. doi: http://dx.doi.org/10.1007/978-3-642-01310-2.

Alexei Lapouchnian. Goal-oriented requirements engineering: An overview of the current research. Technical report, University of Toronto, June 2005.

Ken Laskey, Jeff A. Estefan, Francis G. McCabe, and Danny Thornton. Reference architecture foundation for service oriented architecture version 1.0. Committee Draft 02, Oasis, http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf, October 2009.

Hui Lei, Daby M. Sow, John S. Davis, II, Guruduth Banavar, and Maria R. Ebling. The design and applications of a context service. ACM SIGMOBILE Mobile Computing and Communications Review, 6(4):45–55, October 2002. ISSN 1559-1662. doi: http://doi.acm.org/10.1145/643550.643554. URL `http://doi.acm.org/10.1145/643550.643554`.

Emmanuel Letier and Axel van Lamsweerde. Deriving operational software specifications from system goals. In Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering, pages 119–128, 2002a.

Emmanuel Letier and Axel van Lamsweerde. Deriving operational software specifications from system goals. ACM SIGSOFT Software Engineering Notes, 27(6):119–128, November 2002b. ISSN

0163-5948. doi: http://doi.acm.org/10.1145/605466.605485. URL `http://doi.acm.org/10.1145/605466.605485`.

Lin Liu and Eric Yu. Designing information systems in social context: a goal and scenario modelling approach. Journal of Information Systems, 29(2):187–203, April 2004. ISSN 0306-4379. doi: 10.1016/S0306-4379(03)00052-8. URL `http://dl.acm.org/citation.cfm?id=982308.982314`.

David Martin, Mark Burstein, Jerry Hobbs Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. Owl-s: Semantic markup for web services, November 2004. URL `http://www.w3.org/Submission/OWL-S/`.

Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. WonderWeb deliverable D18 ontology library (final). Technical report, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web, December 2003.

Deborah L. McGuinness and Alexander Borgida. Explaining subsumption in description logics. In IJCAI (1), pages 816–821, 1995.

E. Mena, V. Kashyap, A. Illarramendi, and A. Sheth. Domain Specific Ontologies for Semantic Information Brokering on the Global Information Infrastructure, pages 269–283. IOS Press, June 1998. ISBN 0922-6389.

John-Jules Ch. Meyer. Intelligent agents: Issues and logics. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, Logics for Emerging Applications of Databases, pages 131–165. Springer, 2003.

Simon K Milton and Ed. Kazmierczak. An ontology of data modelling languages: a study using a common-sense realistic ontology. Journal of Database Management, 15(2):9–38, 2004.

M.N. Moghadasi, A.T. Haghighat, and S.S. Ghidary. Evaluating markov decision process as a model for decision making under uncertainty environment. In 2007 International Conference on Machine Learning and Cybernetics, volume 5, pages 2446 –2450, aug. 2007. doi: 10.1109/ICMLC.2007.4370557.

Dana Nau, Malik Ghallab, and Paolo Traverso. Automated Planning: Theory & Practice. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558608567.

Christiaan Frank Nijenhuis. Automatic generation of graphical domain ontology editors. Master's thesis, University of Twente, March 2011.

Barry Norton, Mick Kerrigan, Adrian Mocan, Alessio Carenini, Emilia Cimpian, Marc Haines, James Scicluna, and Michal Zaremba. Reference ontology for semantic service oriented architectures. Public review draft 01, OASIS Semantic Execution Environment TC, November 2008.

Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In Proceedings of the Conference on The Future of Software Engineering, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi: 10.1145/336512.336523. URL http://dx.doi.org/10.1145/336512.336523.

Lin Padgham and Wei Liu. Internet collaboration and service composition as a loose form of teamwork. Journal of Network and Computer Applications, 30(3):1116 – 1135, 2007. ISSN 1084-8045. doi: DOI:10.1016/j.jnca.2006.04.006. URL http://www.sciencedirect.com/science/article/pii/S1084804506000361.

Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing research roadmap. Technical report, European Union Information Society Technologies (IST), Directorate D, 2006. URL http://infolab.uvt.nl/pub/papazogloump-2006-96.pdf.

Mike P. Papazoglou and Dimitris Georgakopoulus. Service-oriented computing. Communications of ACM, 46(10):25–28, October 2003.

Chris Preist. A conceptual architecture for semantic web services. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, Proceedings of the International Semantic Web Conference (ISWC 2004), volume 3298 of Lecture Notes in Computer Science, pages 395–409. Springer, 2004.

Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008.

Dick A. C. Quartel, Wilco Engelsman, Henk Jonkers, and Marten van Sinderen. A goal-oriented requirements modelling language for enterprise architecture. In Proceedings of the 13th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2009, pages 3–13, Auckland, New Zealand, September 2009. IEEE Computer Society.

Dick A. C. Quartel, Wilco Engelsman, and Henk Jonkers. Archimate extension for modeling and managing motivation, principles and requirements in togaf. Technical report, The Open Group, BiZZdesign, October 2010.

Sudha Ram and Jinsoo Park. Semantic Conflict Resolution Ontology (SCROL): An ontology for detecting and resolving data and schema-level semantic conflicts. IEEE Transactions on Knowledge and Data Engineering, 16(2):189–202, February 2004. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2004.1269597.

Fano Ramparany, Remco Poortinga, M. Stikic, J. Schmalenströer, and T. Prante. An open context information management infrastructure – the IST-Amigo project. In Proceedings of the 3rd IET International Conference on Intelligent Environments 2007 – IE07, Ulm, Germany, September 2007.

Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a bdi-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91), pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.5675`.

Anand S. Rao and Michael P. Georgeff. Decision procedures for bdi logics. Journal of Logic and Computation, 8(3):293–343, 1998. doi: 10.1093/logcom/8.3.293. URL `http://logcom.oxfordjournals.org/content/8/3/293.abstract`.

Colette Rolland, Rim Samia Kaabi, and Naoufel Kraïem. On isoa: Intentional services oriented architecture. In John Krogstie, Andreas L. Opdahl, and Guttorm Sindre, editors, Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007), volume 4495 of Lecture Notes in Computer Science, pages 158–172. Springer Verlag, 2007.

Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. Applied Ontology, 1(1):77–106, 2005. ISSN 1570-5838. URL `http://www.metapress.com/content/dccb867p347xxebx`.

Jeffrey S. Rosenschein and Gilad Zlotkin. Rules of Encounter: Designing Conventions for Automated Negotiation among Computers. Artificial Intelligence series. MIT Press, July 1994.

S. Russel and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, 2nd edition edition, 2002.

Stuart J. Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, NJ, USA, 3rd edition, December 2009.

John R. Searle. Mind, Language, and Society : Philosophy in the Real World. Basic Books, 1998. ISBN 0465045219.

Amit P Sheth, Karthik Gomadam, and Ajith Ranabahu. Semantics enhanced Services: METEOR-S, SAWSDL and SA-REST. IEEE Bulletin of the Technical Committee on Data Engineering, 31(3):8–12, September 2008.

Yoav Shoham. Agent-oriented programming. Artificial Intelligence, 60(1):51–92, March 1993. ISSN 0004-3702. doi: 10.1016/0004-3702(93)90034-9. URL `http://portal.acm.org/citation.cfm?id=152185.152188`.

Renata Silva Souza Guizzardi. Agent-oriented Constructivist Knowledge Management. PhD thesis, University of Twente, February 2006.

E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. Journal of Web Semantics, 5(2):51–53, June 2007. ISSN 1570-8268. URL `http://apps.isiknowledge.com.proxy.library.ucsb.edu:2048/full_record.do?product=WOS&search_mode=GeneralSearch&qid=11&SID=4ClAPHkFckJgGHMNI5N&page=1&doc=2`.

Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Code: A development environment for owl-s web services. Technical Report CMU-RI-TR-05-48, Robotics Institute, Pittsburgh, PA, October 2005.

Angelo Susi, Anna Perini, John Mylopoulos, and Paolo Giorgini. The tropos metamodel and its use. Informatica, 29:401–408, 2005.

Katia Sycara, Massimo Paolucci, Julien Soudry, and Naveen Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. IEEE Internet Computing, 8(3):66–73, 2004. ISSN 1089-7801. doi: http://dx.doi.org/10.1109/MIC.2004.1297276.

The Open Group. ArchiMate 1.0 Specification. Van Haren Series. Centraal Boekhuis, 2009a. ISBN 9789087535025. URL `http://www.opengroup.org/archimate/doc/ts_archimate/`.

The Open Group. TOGAF 9 - The Open Group Architecture Framework Version 9, 2009b. URL `http://www.opengroup.org/togaf`.

Stephen Ullmann. Semantics : an introduction to the science of meaning. Basil Blackwell, Oxford, revised edition, 1972.

Wiebe van der Hoek and Michael Wooldridge. Towards a logic of rational agency. Logic Journal of the IGLP, 11(2):133–157, 2003.

A. Van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. In Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE95), pages 194–203, Washington, DC, USA, 1995. IEEE Computer Society.

Axel van Lamsweerde and Laurent Willemet. Inferring declarative requirements specifications from operational scenarios. IEEE Transactions on Software Engineering, 24(12):1089–1114, December 1998. ISSN 0098-5589. doi: 10.1109/32.738341.

Jan-Willem van ’t Klooster, Bert-Jan van Beijnum, Pravin Pawar, Klass Sikkel, Lucas Meertens, and Hermie Hermens. What do elderly desire? a case for virtual communities. In Proceedings of the International Workshop on Web Intelligence and Virtual Enterprises (WIVE09), Thessaloniki, Greece, October 2009.

Venu Vasudevan. Augmenting OMG traders to handle service composition. `http://www.objs.com/survey/compositional-trader.html`, September 1998.

K. Verma and A. Sheth. Semantically annotating a web service. Internet Computing, IEEE, 11(2):83 –85, march-april 2007. ISSN 1089-7801. doi: 10.1109/MIC.2007.48.

M. Vukovic and P. Robinson. Goalmorph: Partial goal satisfaction for flexible service composition. In Proc. International Conference on Next Generation Web Services Practices NWeSP 2005, page 6pp., 22–26 Aug. 2005a. doi: 10.1109/NWESP.2005.44.

Maja Vukovic and Peter Robinson. Goalmorph: Partial goal satisfaction for flexible service composition. In Proceedings of the International Conference on Next Generation Web Services Practices (NWESP '05), pages 149–154, Washington, DC, USA, 2005b. IEEE Computer Society. ISBN 0-7695-2452-4. doi: http://dx.doi.org/10.1109/NWESP.2005.44.

Hans Weigand, Willem-Jan van den Heuvel, and Marcel Hiel. Business policy compliance in service-oriented systems. Information Systems, 36(4):791 – 807, 2011. ISSN 0306-4379. doi: 10.1016/j.is.2010.12.005. URL `http://www.sciencedirect.com/science/article/pii/S0306437910001377`.

Mark Weiser. The computer for the 21st century. Scientific American, 265(3):66–75, September 1991.

Daniel S. Weld. An introduction to least commitment planning. AI Magazine, 15:27–61, 1994.

Christopher Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. Data & Knowledge Engineering, 39(1):51–74, October 2001. ISSN 0169-023X. doi: http://dx.doi.org/10.1016/S0169-023X(01)00030-1.

Terry Winograd and Fernando Flores. Understanding computers and cognition - a new foundation for design. Addison-Wesley, 1987. ISBN 978-0-201-11297-9.

Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener. Designing object-oriented software. Prentice Hall, 11 edition, 1990. URL `http://books.google.com/books?id=KpJQAAAAMAAJ`.

Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. Knowledge Engineering Review, 10 (2):115–152, 1995. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.2702`.

Michael Wooldridge, Nicholas R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems, 3(3):285–312, 2000. URL `http://eprints.ecs.soton.ac.uk/3748/`.

E.S.K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In Proceedings of the Third IEEE International Symposium on Requirements Engineering, 1997, pages 226 –235, January 1997. doi: 10.1109/ISRE.1997.566873.

M. Zarifi Eslami, A. Zarghami, B. Sapkota, and M. J. van Sinderen. Service tailoring: Towards personalized homecare systems. In Proceedings of the 4th International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2010), Athens, Greece, pages 109–121, Athenes, Greece, July 2010. SciTePress.

Pamela Zave. Classification of research efforts in requirements engineering. ACM Computing Surveys, 29:315–321, December 1997. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/267580.267581. URL `http://doi.acm.org/10.1145/267580.267581`.

Kangkang Zhang, Qingzhong Li, and Qi Sui. A goal-driven approach of service composition for pervasive computing. In Proceedings of the 1st International Symposium on Pervasive Computing and Applications, pages 593–598, 3–5 Aug. 2006. doi: 10.1109/SPCA.2006.297491.

Publications by the Author

During the development of this thesis, the author has published various parts of this work in the following papers (listed in reverse chronological order):

Bonino da Silva Santos, L.O., Sorathia, V., Ferreira Pires, L., van Sinderen, M. J., *Towards a Conceptual Framework to Support Dynamic Service Provisioning for Non-Technical Service Clients*, Journal of Software, Academy Publisher, 6 (4), pp. 564-573, April 2011.

Bonino da Silva Santos, L.O., Sorathia, V., Ferreira Pires, L., van Sinderen, M. J., *An Approach to Dynamic Provisioning of Social and Computational Services*, the IEEE 6th World Congress on Services (SERVICES 2010), Miami, Florida, USA, July 5-10, 2010.

Bonino da Silva Santos, L.O., Ferreira Pires, L., van Sinderen, M. J., *Service Provisioning Support for Non-Technical Service Clients*, the 7th International Conference on Information Technology: New Generations (ITNG 2010), Las Vegas, Nevada, USA, April 12-14, 2010.

Bonino da Silva Santos, L.O., Guizzardi, G., Ferreira Pires, L., van Sinderen, M. J., *From User Goals to Service Discovery and Composition*, the 3rd International Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGIM'09) with the 28th International Conference on Conceptual Modeling (ER 2009), Gramado, Brazil, November 9-12, 2009.

Bonino da Silva Santos, L.O., Guizzardi, G., Silva Souza Guizzardi, R., Gonçalves da Silva, E., Ferreira Pires, L., van Sinderen, M. J., *GSO: Designing a Well-Founded Service Ontology to Support Dynamic Service Discovery and Composition*, the 2nd International Workshop on Dynamic and Declarative Business Process (DDBP 2009) with the 13th IEEE International EDOC Conference (EDOC 2009), Auckland, New Zealand, September 1st, 2009.

Bonino da Silva Santos, L.O., Gonçalves da Silva, E., Ferreira Pires, L., van Sinderen, M. J., *Towards a Goal-Based Service*

Framework for Dynamic Service Discovery and Composition, the 6th International Conference on Information Technology: New Generations (ITNG 2009), Las Vegas, Nevada, USA, April 27-29, 2009.

Bonino da Silva Santos, L.O., Ferreira Pires, L., van Sinderen, M. J., *A Trust-Enabling Support for Goal-Based Services*, the 2008 IEEE International Symposium on Trusted Computing (TrustCom 2008), Zhang Jia Jie, Hunan, China, November 18-20, 2008.

Bonino da Silva Santos, L.O., Ferreira Pires, L., van Sinderen, M. J., *A Goal-Based Framework for Dynamic Service Discovery and Composition*, 2nd International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2008) in conjunction with the 3rd International Conference on Software and Data Technologies (ICSOFT 2008), Porto, Portugal, July 5, 2007.

Bonino da Silva Santos, L.O., Poortinga-van Wijnen, R., Vink, P., *A Service-Oriented Middleware for Context-Aware Applications*, 5th International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2007) in conjunction with the ACM/IFIP/USENIX 8th International Middleware Conference (Middleware 2007), Newport Beach, CA, USA, November 26-30 2007.

Bonino da Silva Santos, L.O., Vink, P., Poortinga-van Wijnen, R., *Demo: A Service-Oriented Middleware for Providing Context Awareness and Notification*, ACM/IFIP/USENIX 8th International Middleware Conference (Middleware 2007), Demo Track, Newport Beach, CA, USA, November 26-30 2007.

Bonino da Silva Santos, L.O., van Sinderen, M., Ferreira Pires, L., *Architectural Models for Client Interaction on Service-Oriented Platforms*, in Proceedings of the 1st International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2007) in conjunction with the 2nd International Conference on Software and Data Technologies (ICSOFT 2007), Barcelona, Spain, July 22 2007.

Bonino da Silva Santos, L.O., Ramparany, F., Dockhorn Costa, P., Vink, P., Etter, R., Broens, T., *A Service Architecture for Context Awareness and Reaction Provisioning*, in Proceedings of the 2007 IEEE Congress on Services (Services 2007), ISBN 978-0-7695-2926-4, p. 25-32, 2nd Modeling, Design, and Analysis for Service-Oriented Architecture Workshop (MDA4SOA 2007), Salt Lake City, USA, July 13th 2007.

Bonino da Silva Santos, L.O., van Sinderen, M., Ferreira Pires, L., *Dynamic Service Discovery and Composition for Ubiquitous Networks Applications*, Second Conference on Future Networking

Technologies (CoNEXT 2006), Poster Track, Lisbon, Portugal, 4-7 December 2006.

Bonino da Silva Santos, L.O., Guizzardi, R.S.S., van Sinderen, M., *Agent-Oriented Context-Aware Platforms Supporting Communities of Practice in Health Care*, In proceedings of the 4 th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2005), vol. 3, pp. 1287-1288, Utrecht, the Netherlands, July 25-29 2005.

Bonino da Silva Santos, L.O., Guizzardi, R.S.S., van Sinderen, M., *Agent-Oriented Approach to Develop Context-Aware Applications: A Case Study on Communities of Practice*, CTIT Report TR-CTIT-05-19, ISSN 1381-3625, May 2005, 17 pp.

Bonino da Silva Santos, L.O., Guizzardi, R.S.S., van Sinderen, M., Pires, L.F., Wagner, G., Pereira Filho, J.G., Konstantas, D., *Context-Aware Support for Communities of Practice*, In the 3rd International Semantic Web Conference (ISWC2004), Poster Track , Hiroshima, Japan, November, 2004.

Resumo

Computação Orientada a Serviços (SOC, na sigla em inglês) é um paradigma para projeto, uso e gerência de aplicações de sistemas distribuídos na forma de serviços. Na visão da SOC, serviços representam pedaços de funcionalidades distribuídos que podem ser combinados (ou compostos, na terminologia da SOC) para gerar novas funcionalidades com mais valor agregado. Em um cenário ideal baseado nessa visão, um cliente de serviço expressa seus requisitos a uma infraestrutura de software, e esse software descobre, seleciona e invoca os serviços sem a necessidade de interações adicionais com o cliente humano. Requisitos não-funcionais como custo, confiança e privacidade, entre outros, também devem ser expressos pelo cliente de serviço e tratados automaticamente pela infraestrutura de software.

A visão da SOC também se sobrepõe a algumas das características da Computação Pervasiva. No seu artigo seminal sobre Computação Pervasiva (também conhecida como Computação Ubíqua), Weiser previu que dispositivos computacionais, de sensoriamento e de comunicação seriam embutidos de forma transparente no ambiente ao nosso redor. Esse ambientes enriquecidos com dispositivos computacionais dariam acesso permanente e em qualquer lugar a informações e serviços. Informação disponível prontamente pode contribuir para a concretização da visão da SOC, especialmente ao permitir que infraestruturas de software capturem informações relacionadas à execução de serviços sem que seja necessária uma interação direta do usuário.

Apesar da completa automação do provisionamento de serviços ser o objetivo final da SOC, muito trabalho ainda precisa ser realizado para que esse objetivo seja alcançado. Além disso, uma ampla adoção da Computação Orientada a Serviços e da Computação Pervasiva requereria que essas tecnologias se tornassem mais atraentes aos usuários não-técnicos em sua vida cotidiana. Em cenários onde um número significativo de serviços, provedores de serviços e clientes de serviços estejam disponíveis podem surgir questões como: *(i)* como expressar os requisitos de serviços de uma forma intuitiva (adequada a clientes não-técnicos); *(ii)* como tratar problemas de interoperabilidade semântica em requisitos de serviços, descrições de serviços e na interpretação interna dos termos usados em operações de serviços, que usem diferentes modelos conceituais; *(iii)* como suportar a descoberta, seleção e invocação de serviços que satisfaçam os objetivos de seus clientes de forma menos invasiva e disruptiva; e *(iv)* como combinar serviços executados por humanos com serviços executados por sistemas computacionais.

Em nosso trabalho, nós consideramos cenários onde usuários não-técnicos estão cercados por dispositivos e sensores inteligentes e capazes de se comunicar, e onde um grande número de serviços está disponível. Nesses cenários, um suporte adicional deve ser oferecido aos usuários finais para auxiliá-los diante da possibilidade de uma quantidade (possivelmente) não gerenciável de decisões e interações no que concerne às etapas de provisionamento de serviços como especificação dos requisitos de serviços, descoberta, seleção, acordo, composição e invocação de serviços.

Nessa tese, nós apresentamos um framework conceitual para suportar o provisionamento dinâmico de serviços a usuários não-técnicos. As principais contribuições do nosso framework são: *(i)* permitir que clientes de serviços expressem seus requisitos de serviços utilizando o conceito de objetivo, que é mais próximo ao seu entendimento intuitivo que artefatos técnicos como, por exemplo, um documento escrito utilizando a linguagem WSDL; *(ii)* reduzir a necessidade de interações diretas entre o usuário e os serviços através do uso de informações obtidas automaticamente do ambiente; *(iii)* fornecer uma linguagem para a especificação de domínios que suporte especialistas de domínio na modelagem de conceitos de domínio e serviços. Adicionalmente, essa linguagem de especificação de domínios oferece primitivas de modelagem que permitem a distinção entre serviços computationais e sociais, que são prestados por sistemas computacionais e por humanos, respectivamente.

Os resultados concretos dessa tese são: *(i)* a descrição e o projeto de um framework baseado em objetivos para o provisionamento dinâmico de serviços; *(ii)* o projeto e a implementação de um protótipo da plataforma de software descrita pelo framework, que suporta o provisionamento dinâmico de serviços; *(iii)* a definição de uma ontologia fundamental que oferece a base ontológica para a *(iv)* linguagem de especificação de domínios.

O framework proposto nessa tese foi avaliado a partir de es-

tudos de caso representativos que cobriram o uso do framework no que concerne à modelagem de domínios de aplicação e à operação da plataforma de software para suportar o provisionamento dinâmico de serviços nesses domínios.