Angelo Furno

Department of Engineering

University of Sannio, Benevento

September 2014

Dissertation Committee

Prof. Giuseppe Antonio Di Lucca	Università degli Studi del Sannio - Italy
Prof. Elisabetta Di Nitto	Politecnico di Milano - Italy
Prof. Andrian Marcus	University of Texas, Dallas - USA

Dissertation accepted on 18^{th} July 2014

O sol che sani ogni vista turbata,
tu mi contenti sì quando tu solvi,
che, ***non men che saver, dubbiar m'aggrata****.*

O Sun, that healest all distempered vision,
Thou dost content me so, when thou resolvest,
That ***doubting pleases me no less than knowing****.*

Divina Commedia, Inferno: C. XI, vv. 91-93. Dante Alighieri.

Sommario

Questa tesi riporta i risultati di un'intensa attività di ricerca riguardante l'analisi dello stato dell'arte, la definizione di nuove soluzioni e la loro valutazione sperimentale realizzata, grazie allo sviluppo di prototipi, nel contesto della composizione scalabile e context-aware di servizi. La composizione di servizi è un aspetto fondamentale delle applicazioni fondate su architetture orientate ai servizi; essa consente la creazione di nuovi servizi (e quindi nuova conoscenza) attraverso il riuso di quelli esistenti.

Dopo una disamina dello stato dell'arte, nella prima parte di questo lavoro, si descrivono l'approccio ed uno strumento correlato per la composizione automatica ed adattiva di servizi semanticamente descritti. Lo strumento consente la generazione automatica di composizioni eseguibili, analizzando sia descrizioni semantiche di servizi pubblicate in un repository centralizzato sia il problema specifico da risolvere. A differenza di altre soluzioni disponibili, il tool è in grado di costruire automaticamente sia il flusso di controllo che quello dati, sfruttando tecniche di pianificazione derivate dall'intelligenza artificiale ed analizzando semanticamente le dipendenze di ingresso/uscita tra le attività componenti.

Proseguendo lungo questo tema di ricerca, sono state affrontate tematiche relative alla context-awareness al fine di implementare una visione autonomica della composizione di servizi ed una migliore caratterizzazione sia del profilo dell'utente, richiedente il soddisfacimento di obiettivi di business, che dell'ambiente in cui la composizione è realizzata. A questo scopo, è stato definito un linguaggio ontologico per la modellazione dei contesti e tecniche semantiche per realizzare descrizioni context-aware di servizi, in base a regole di adattamento al contesto. Con questo approccio, gli adattamenti della composizione iniziale possono essere attivati in seguito a cambiamenti dell'ambiente, dei profili utente o dei relativi obiettivi di business, molto frequenti nel dominio dei servizi.

Al fine di aumentare la scalabilità del processo di composizione, sono state esplorate le reti e gli algoritmi Peer-to-Peer (P2P). In particolare, sono state sviluppate

una tecnica backward ed una bidirezionale di ricerca per realizzare discovery e composizione di servizi in maniera automatica, completamente distribuita e cooperativa, in reti P2P non strutturate. Questo approccio consente ad ogni peer di agire sia come consumatore di servizi che come fornitore, pubblicando nuove descrizioni di servizi (sia strutturali che semantiche) in un registro locale. Le descrizioni possono essere esposte al Web ed i peer possono partecipare cooperativamente ai processi di ricerca e composizione conseguenti ad una richiesta di servizio sottomessa da un qualsiasi peer della rete. È stata infine proposta e valutata una tecnica efficiente per migliorare le prestazioni del processo di composizione automatica e cooperativa in ambienti P2P non strutturati, basata su gossiping dinamico, che utilizza diverse fonti di conoscenza, come la densità di rete ed il raggruppamento di servizi, per ridurre la quantità di messaggi scambiati fra i diversi peer. La strategia proposta è stata valutata in confronto ad altre tecniche, in diversi scenari e configurazioni, mostrando importanti benefici prestazionali in relazione sia al tempo di esecuzione che al numero di messaggi scambiati nella rete.

Keywords: SOA, Service Composition, Context-awareness, Autonomic Computing, Peer-to-Peer Systems.

Abstract

This thesis reports on the results of an intense research activity concerning the analysis of the state of the art, the definition of new solutions and their experimental evaluation, by means of prototypical developments, in the context of Service Composition. Service composition is a fundamental facet of the applications based on Service Oriented Architectures; it allows for the creation of new services (and therefore new knowledge) by re-using the existing ones.

After discussing related work, in the first part of the thesis, we report on an approach and a related tool for automatic composition of semantically described services. The tool allows for the automatic generation of executable compositions, by analyzing both semantic service descriptions published in a centralized repository and the problem statement to solve. Differently from other available tools, it is able to automatically build both the control and the data flows, by exploiting planning techniques coming from the field of Artificial Intelligence and semantically analyzing input/output dependencies among the composing activities.

Going further along this research subject, we focused our efforts on context-awareness to implement an autonomic vision of service composition and better characterize both the requesting user's profile and the environment where composition takes place. To this purpose, we defined an ontology language for modeling contexts and semantic techniques to allow for context-aware service descriptions, based on context adaptation rules. By this approach, adaptations of the initial service composition may be enacted when the environment or the user profiles/objectives change, which is usual in service domains.

In order to increase the scalability of the composition process, we explored Peer-to-Peer (P2P) networks and algorithms. We have developed backward and bidirectional search techniques for automatic, fully-decentralized and cooperative service composition/discovery in unstructured P2P networks. This approach allows any peer to act both as service consumer and as provider, by publishing new service descriptions

(both structural and semantic) in a local registry. The descriptions can be exposed to the Web and peers may cooperatively participate to discovery and composition processes started for solving service goal queries. We have proposed an efficient technique for improving the performance of automatic and cooperative composition in unstructured P2P networks, based on dynamic gossiping, that uses different sources of knowledge, such as network density and service grouping, to reduce the amount of messages exchanged among the peers. This strategy has been compared to other forwarding techniques, in several scenarios and configurations, showing important performance improvements both in relation to resolution time and message overhead.

Keywords: SOA, Service Composition, Context-awareness, Autonomic Computing, Peer-to-Peer.

List of Publications

Journals

[1] Furno A., and Zimeo, E. (2014, July). Self-Scaling Cooperative Discovery of Service Compositions in Unstructured P2P Networks. In *Journal of Parallel and Distributed Computing (JPDC)*, DOI: 10.1016/j.jpdc.2014.06.006, ISSN: 0743-7315, Elsevier.

[2] Furno, A., and Zimeo, E. (2014, April). Context-aware Composition of Semantic Web Services. In *Springer Mobile Networks and Applications Journal (MONET) - Special Issue on Context-aware Systems and Applications, 2014*, DOI: 10.1007/s11036-014-0494-y, pp. 235-248 - Volume 19, Issue No. 2, ISSN: 1383-469X, Springer US.

[3] Bevilacqua, L., Furno, A., di Carlo, V. S., and Zimeo, E. (2012, October). Automatic Generation of Concrete Compositions in Adaptive Contexts. In *The Mediterranean Journal of Computers and Networks (MEDJCN)* - Volume 8, Issue No. 4, ISSN: 1744-2397.

Conference Proceedings

[1] Furno, A., and Zimeo, E. (2014, February). Gossip Strategies for Service Composition. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pp. 27-35, IEEE.

[2] Zagarese, Q., Furno, A., Canfora, G., and Zimeo, E. (2013, October). Towards Effective Event-Driven SOA in Enterprise Systems. In *Systems, Man and Cybernetics, 2013 International Conference on*, pp. 1419 - 1424, IEEE.

[3] Furno, A., and Zimeo, E. (2013, February). Efficient Cooperative Discovery of Service Compositions in Unstructured P2P Networks. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pp. 58-67, IEEE.

[4] Furno, A., and Zimeo, E. (2013, January). Context-Aware Design of Semantic Web Services to Improve the Precision of Compositions. In *Context-Aware Systems and Applications (pp. 97-107)* - Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering Volume 109, 2013, pp 97-107, Springer Berlin Heidelberg, **Best Paper Award**.

[5] Bevilacqua, L., Furno, A., di Carlo, V. S., and Zimeo, E. (2011, September). A Tool for Automatic Generation of WS-BPEL Compositions from OWL-S described Services. In *Software, Knowledge Information, Industrial Management and Applications (SKIMA), 2011 5th International Conference on*, pp. 1-8, IEEE.

Other

[1] Furno, A., and Zimeo, E. (2014, February). P2P Architectures for Semantic Service Composition. In *Parallel, Distributed and Network-Based Processing (PDP) - Work In Progress Section, 2014 22nd Euromicro International Conference on*, ISBN: 978-3-902457-39-4, Institute for Systems Engineering and Automation Johannes Kepler University Linz.

[2] Furno, A. (2012, September). Enhancing Web Process Self-awareness with Context-aware Service Composition. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Ph. D. Awareness Forum*, September 2012.

Preface

This work is the result of a research path I undertook three years ago at the Department of Engineering of the University of Sannio, under the supervision of Professor Eugenio Zimeo.

Its main topic concerns service composition. More specifically, this thesis describes the approaches, solutions and tools aimed at generating concrete, executable and context-aware service compositions from semantically described services and context models. In particular, to allow for scalable service discovery and composition, a P2P approach is proposed to implement a fully distributed service registry. By exploiting a probabilistic gossip-based forwarding strategy, the proposed approach is able to keep low message and resolution time overhead.

The work has been organized into six chapters and can be summarized as follows:

- chapter 1 introduces the problems addressed in this work with respect to service composition and outlines the achieved scientific contributions;
- chapter 2 reports on related work regarding service composition, autonomic computing in service composition and context modeling, and distributed solutions for supporting more scalability in the composition process, by highlighting the relevant limitations of the approaches proposed in the literature;
- chapter 3 describes our proposed approach and tool for generating concrete and executable service compositions from semantically described services;
- chapter 4 introduces our context modeling ontology and the OWL-S extension proposed for describing context-aware services and allowing for context-aware service composition;
- chapter 5 details our solution for automatic and cooperative composition in unstructured P2P networks and an efficient technique proposed for improving its performance;

- chapter 6 analyzes the achieved results and outlines the future research directions.

Acknowledgements

This work is dedicated to my family, my beloved Valeria, my dear colleagues and all of my friends. You made my efforts possible, you make my life meaningful, you are my greatest energy. Special thanks go to my tutor, prof. Eugenio Zimeo, who has been an excellent mentor, both in research and in life, for all these years.

Contents

Chapter 1

Introduction

The adoption of Service Oriented Architectures (SOA) and Web Services (WS) is promoting a novel approach for designing and developing Web applications: they can be created by composing different services, possibly provided by different administration domains.

As the number of Web services deployed in the Internet grows, their discovery becomes a fundamental feature for potential clients and poses important challenging problems to tackle. This is especially true in the cloud context, where the "pay-per-use" economy model asks for efficient and scalable discovery techniques to find the services that better fit user requirements. In particular, discovery by composition increases computational complexity and resource consumption, and, consequently, requires more efficient and scalable techniques to be effective. In addition, services and business processes[1] are sensitive to the environment where they execute. In other words, the execution context significantly impacts both service design and execution, and contains additional and relevant information for understanding the specific service goals to satisfy.

This chapter details the problems and challenges that emerge when dealing with automatic service composition to allow for the execution of the generated service compositions, to adapt services to the surrounding environment, and to preserve efficiency of the composition process, even when working with large or very large-size (Internet scale) service registries.

[1]Even though service compositions can refer to different application domains (e.g. scientific workflows), in this thesis, we mainly refer to them as the implementation of business processes. Hence, the terms *business process*, *workflow* and *service composition* are often used as synonyms in the rest of the thesis.

1.1 Problem Statement

Syzperski [93] defined service composition as the process of constructing a complex service from atomic ones to achieve a specific task. The process of service composition inherently requires the specification of composition requests, a formal specification of static and behavioral properties of the service components, a matchmaking algorithm, and a modeling language expressing the logic of a composite service (see Fig. 1.1). Web service technology is a popular way of developing distributed applications.

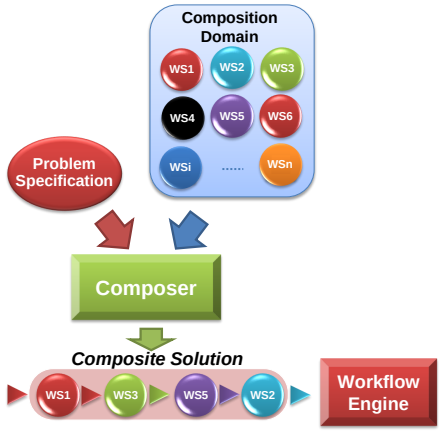


Figure 1.1: Service composition

In spite of the several scientific results already achieved regarding service composition, generating a concrete and runnable service composition from the semantic descriptions of the domain services and the problem to solve is still an open issue. Most of the solutions available in literature (e.g. [55, 72]) can identify a set of services solving the specific requested goals, but such service compositions are abstract and cannot be directly executed on traditional business process engines. Often, a very limited set of control flow primitives is supported (e.g. [109]), service implementations are not bound to the identified process activities and, more often, data flow is not managed. Therefore, to generate an executable business process description starting from user business requirements and service domain requires:

- support for formally describing service domain and business problems;
- an efficient technique for finding single services or combinations of multiple services from the domain, satisfying the specified problem;

- the automatic generation of a formal business process description, possibly in a standard language (e.g. WS-BPEL[2], XPDL[3]), from the abstract plan, containing the definitions of both the process control flow and the data flow.

Another important aspect to consider is that modern service environments present an extremely dynamic nature: services are frequently modified or replaced; they can disappear, and new services with different features may become available. Therefore, service compositions should react to external events and change their structure accordingly, thus reducing human intervention to the minimum.

Service compositions should be able to self-configure themselves through the knowledge coded at design-time or collected at run-time, by changing the overall composition graph to make it runnable within the new conditions. They should be able to change a link between an activity and a concrete service (i.e. *re-binding*); insert, delete or replace an activity; change the endpoints of a transition; substitute an activity with an equivalent sub-process.

From the designer perspective, the descriptions of services and their compositions should be augmented by means of context-awareness. They should be designed by exploiting new models and approaches allowing for slightly change the structure and behavior of the services according to the dynamics needs of the context.

When the number of services to handle grows, traditional centralized service registries (e.g. UDDI) and composition approaches easily become unpractical, with extremely high resolution times. More scalable and efficient architectures than centralized ones become necessary to implement service registries and their discovery capabilities.

The adoption of distributed approaches based on a hierarchical architecture, similar to the one adopted by the Domain Name System (DNS), could represent a solution. In this direction, standards like UDDI v.3.0[4] and ebXML[5] have partially addressed the problems of scalability and fault-tolerance by proposing the use of selective replication to create hierarchical registry federations. However, since services can be linked together to create solutions for complex needs, service discovery becomes a very complex process with respect to DNS resolution: if the desired services are not available in isolation, it is still possible to find solutions by combining multiple services and generate composite services. This way, discovery can exhibit a higher level of recall (i.e. the number of solutions found with respect to the number of solutions actually

[2]http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
[3]http://www.wfmc.org/standards/xpdl
[4]http://uddi.org/pubs/uddi-v3.0.2-20041019.htm
[5]http://www.ebxml.org

present in the network) compared to the well-known discovery of atomic services. Conversely, hierarchical architectures may become inadequate to support large-scale service composition.

Peer-to-Peer (P2P) architectural models, and in particular unstructured P2P overlays, may act as the best candidates to implement fully distributed and scalable service registries. These models, in fact, ensure high *functional* and *non-functional scalability*.

A P2P registry on a large-scale network, such as the Internet, allows for *functional scalability* in the sense that multiple organizations can have their own services (as DNS works for network resources addresses) published in their own local registries and exposed to the Web for discovery and composition. This way, much more services (i.e. more functionalities) than in a traditional centralized approach can be managed and, by exploiting composition, a huge number of value-added services can be offered to consumers.

Non-functional scalability (e.g. lower resolution time, effective bandwidth and local resource usage, etc.) can be obtained by exploiting the increased level of concurrency made possible by a P2P architecture: large repositories can be explored through several efficient discovery processes running in parallel over the network peers.

Moreover, P2P service registries enable a new form of collaboration where the roles of consumers and providers can be used interchangeably to cooperatively create service compositions that satisfy consumers' queries.

In P2P networks, service compositions can be retrieved by exploiting proper search strategies that explore a distributed service registry (i.e. *distributed composition*) to plan the solutions to the requested service goals. In particular, if the nodes that receive a query collaborate to find a solution without any centralized coordination, the planning problem is solved in a cooperative fashion (i.e. *cooperative composition*).

P2P service registries require proper techniques to support *efficient* cooperative service composition.

In particular, DHT-based solutions use hash functions to select the nodes where services are to be published and reduce discovery time to $\mathcal{O}(log(n))$. However, they impose a fixed network structure and only support exact match for discovering services, losing therefore the flexibility of semantic approaches. In addition, DHT network topologies have to be maintained by exploiting proper management protocols, which can produce a significant overhead, especially when peers and services churn is high.

In unstructured P2P networks, no fixed topology is required and semantic matching during discovery can be easily supported. However, message overhead has to be

properly controlled in order to avoid network overloading, which can negatively affect resolution times. Probabilistic gossip protocols can represent a solution to implement effective message broadcast, due to their simplicity, good reliability, and scalability [34, 100]. However, the adoption of gossip strategies for P2P service composition requires to dynamically tune the fanout (i.e. number of forwarded neighbors) of such strategies in a way that it can adapt itself to the characteristics of the underlying network topology and reduce message complexity for cooperative service discovery and composition.

Another important aspect to consider, especially when dealing with unstructured P2P networks, regards the possibility to capture the knowledge about already solved service compositions and exploit it to increase efficiency during cooperative composition. This knowledge, together with information (e.g. network density) about the connectivity low-level overlay, may drive query propagation in a more and more precise and dynamic way. Thus, a service network can change from the initial completely unstructured organization to a semi-structured (in relation to service composition) organization in the steady-state phase.

Being the service network able to structure itself from the knowledge acquired by composing services over different peers, the discovery protocol can be considered as self-scaling since it allows for effectively and efficiently solving queries, even in very large and dynamically growing P2P networks, where a huge number of services may be available in the peers' local registries.

To further improve cooperative P2P service composition, distributed bidirectional strategies can be explored for both reducing the resolution time and proposing functional gaps to service providers or developers.

1.2 Scientific Contributions

This thesis advances the state of the art of automatic service composition by making the following major scientific contributions:

1. an integrated approach and a related tool for generating concrete and runnable compositions from semantically described services;

2. support for adaptivity in the composition process by means of an ontology language and related tools for context-awareness in service design;

3. a P2P cooperative approach for automatic semantic service composition based on backward and bidirectional search strategies;

4. an efficient gossip-based probabilistic technique for improving the performance of automatic and cooperative compositions in unstructured P2P networks.

1.3 Thesis Outline

The remainder of this work is organized as follows.

Chapter 2 discusses state-of-the-art solutions related to automatic generation of executable service composition, use of context-awareness in service design and composition, and efficient search in P2P networks for supporting service discovery and composition in a distributed service registry.

Chapter 3 describes the approach and the related tool proposed to build executable service compositions from semantically described services, based on a centralized architecture. The tool aims at covering the planning and re-planning phases of autonomic workflows.

Chapter 4 proposes a design approach, based on a semantic model for context representation, aimed at enriching the expressiveness of each section of a typical OWL-S semantic service description, by means of context conditions and adaptation rules. By exploiting the proposed approach, atomic or composite services can be better-tuned to the requestor's behaviors and to the particular situations of the surrounding environment.

Chapter 5 presents our approach for automatic and cooperative compositions in unstructured P2P networks and an efficient technique for improving its performance. The technique exploits a probabilistic forwarding algorithm that uses different sources of knowledge, such as network density and service grouping, to reduce the amount of messages exchanged in the network. The evaluation performed in several network configurations, by using a simulator to observe resolution time, recall and message overhead, has shown good performance especially in dense and large-scale service networks.

Chapter 6 concludes the work. It summarizes our solutions, approaches, evaluations and tools, highlights the achieved scientific contributions, discusses the limitations of our results, and illustrates future research directions.

Chapter 2

Related Work

This chapter discusses the state of the art of three related topics addressed in this thesis: *automatic service composition*, *context modeling and context-aware design*, *peer-to-peer search strategies and forwarding techniques*.

The rest of this chapter is organized as follows. Section 2.1 discusses the relevant attempts, documented in literature, to fully automate the process of developing executable service compositions. Section 2.2 presents models, approaches, and solutions related to context-awareness in service-based systems. Finally, in Section 2.3, we focus on related work addressing the problems of performance in distributed service composition, with particular emphasis on scalability.

2.1 Automatic Centralized Service Composition

To generate an executable business process description from a general, and possibly formal, description of user business requirements and service domain is a complex problem, whose solution needs: 1) a support for formally describing service domain and business problems; 2) an efficient technique for finding single services or combinations of multiple services from the domain, i.e. abstract plans, satisfying the specified problem; 3) the automatic generation of formal and executable business process descriptions, possibly in a standard language (e.g. WS-BPEL, XPDL), from the retrieved abstract plans.

In spite of the plethora of efforts devoted to theoretical aspects, up to now only a few proposals have addressed the problem in a comprehensive way and very few tools are able to generate an executable business process description.

In relation to the first problem, the OWL Web Ontology Language[1] allows for representing domain knowledge through a formal and shared XML-based specification of concepts and relationships among them. OWL-S[2] supplies Web service providers with a core set of markup language constructs for describing properties and capabilities of their Web services in unambiguous, computer-understandable form, by referencing concepts and properties from OWL ontologies.

Semantic Annotations for WSDL and XML Schema (SAWSDL)[3] is another W3C recommendation for semantically describing services. It introduces a set of extension attributes to be directly used in WSDL service descriptions to semantically annotate WSDL elements. WSMO-lite[4] is a lightweight set of semantic service descriptions in RDFS for annotating various WSDL elements, using the SAWSDL annotation mechanism.

Regarding the second problem, several approaches, techniques and tools [26, 60, 79] have been proposed in literature to efficiently tackle the automation of the composition process. Many of the proposed approaches exploit AI planning techniques, handling the Web service composition problem as a state-space, constraints satisfaction, situation-calculus or other kind of planning problems. Semantics is considered an important support for the automation of the composition process [68].

The Planning Domain Definition Language (PDDL) [66] is considered the *de-facto* standard for classical planning problems input languages. A PDDL planning problem is described in two sections: domain definition and problem specification. The domain describes the possible actions, in terms of inputs, outputs, preconditions and effects, and predicates. The problem essentially describes initial and final states, by specifying the set of predicates assumed to be true in the initial state and the set of predicates to be satisfied in the goal state. Several planners have been developed based on PDDL as the input language.

SHOP2 [72], is an HTN (Hierarchical Task Networks) planner, which exploits hierarchical relations among tasks for composing Web services. These relations have to be provided in advance to the planner by designers when describing the planning domain. The planning problem is solved by translating its OWL-S description into a SHOP2 description and by converting the SHOP2 generated plan to an OWL-S runnable process. As pointed in [55], SHOP2 is effective when complete and detailed knowledge on at least partially hierarchically structured action execution patterns is

[1] http://www.w3.org/TR/owl-features/
[2] http://www.w3.org/Submission/OWL-S
[3] http://www.w3.org/TR/sawsdl
[4] http://www.w3.org/Submission/WSMO-Lite

available, but, when no concrete set of methods and decomposition rules are available, an HTN planner is not able to find solutions. This problem inherently limits the planning ability of an HTN planner to the availability of decomposition methods designed by human experts.

In [55, 57], Klush et al. propose the OWLS-Xplan planner, which combines graph-based (by using Graphplan [13]) and HTN planning, using OWL-S descriptions (of both domain and problem) as input, translating them into an XML version of the PDDL language, called PDDXML. The output is a sequence of activities described in PDDXML. The approach combines both the advantages of task decomposition, available with HTN planning, and the Graphplan capability of always finding a solution, when present. The authors also propose a re-planning component, able to update plans during execution.

Baresi et al., in [8], describe a framework for the deployment of adaptable Web service compositions. The infrastructure integrates existing heterogeneous repositories and makes them cooperate for service discovery. The deployment platform (i.e. SCENE [21]) supports BPEL-like compositions that can select services dynamically, and also adjust their behavior in response to detected changes and unforeseen events. The infrastructure exploits a publish/subscribe mechanism to share content-related services published by different registries through topic-specific federations. The framework also provides a monitoring-based validation of running compositions (i.e. Dynamo). However, it requires compositions to be (manually) defined in advance since the discovery infrastructure is not able to find them. Moreover, designed compositions do not take into account any semantic context model to support their adaptations to the execution environment.

Another recent composition framework is PORSCE II [43]. Like OWLS-Xplan, the framework input consists of OWL-S service descriptions, which are translated into PDDL. The framework combines a domain-independent planning component (e.g. JPlan, LPG-td) and an ontology concept relevance module for semantic awareness and relaxation during planning. Several plans, with different semantic accuracy levels, can be generated and presented to the user through a graphical component. Moreover, the graphical component can be used to request re-planning by selecting a task and asking the system to find an alternative equal or semantically similar service or composition. Subsume relationships among service pre- and post-conditions are considered to find such alternatives.

Other notable planning solutions for service composition are based on the Golog language. Golog is a logical programming language and has been extended in [67]

to support customized constraints and non-determinism in sequential executions and have been used in order to support service composition, by means of a translation into PDDL. In [76], a process for translating OWL-S descriptions into situation calculus has been proposed, while, in [58], DL reasoning techniques are used together with extended Golog to calculate conditional Web service compositions.

The Haley framework [109, 110] includes a Golog-based planning system for Web service composition. The system uses SAWSDL semantically described services as input, contains a planning Golog-domain generator, and the eDT-Golog planner. Differently from the previous described framework, Haley is able to generate a WS-BPEL description of the plan and execute it on a WS-BPEL engine. However, Haley tackles the service composition problem from the perspective of generating complete business processes from user business requirements, by assuming the presence of concrete services with specified Quality of Service (QoS) parameters. In this sense, scalability is a very important problem and the hierarchical approach, as in SHOP2, is a way to reduce the planning effort, but requires a designer to know how to decompose tasks in sub-tasks. From our perspective, a centralized planning process has to support especially the generation of small business sub-processes, which concretize tasks from an already defined main workflow, while scalable service composition requires the use of more distributed and decentralized solutions, like those proposed in Section 2.3 and in Chapter 5. Haley's authors believe classical planning techniques are not well suited to the Web service domain, because of its inner non-determinism. As detailed in Section 3.1, we argue that non-determinism can be handled through events observation and proper reaction: this way, the autonomic approach can be exploited to fill the gap with the classical planning techniques in Web service composition. Another important difference between Haley and our composition tool is that Haley is not able to generate concurrent sub-processes.

Finally, in relation to the third problem, the generation of formal and standard business process representation of the service compositions, several languages have been proposed. Among these, Business Process Modeling Notation (BPMN)[5], XML Process Definition Language (XPDL) and Web Service Business Process Execution Language (WS-BPEL) are the most important and widespread ones. In the following, we focus our attention on WS-BPEL (v. 2.0) which can be considered the de-facto standard for business process description languages in the Web service domain.

In Chapter 3 we propose a composition tool based on:

1. the use of OWL and the OWL-S ontology for the semantic descriptions of do-

[5]http://www.omg.org/spec/BPMN

main services;

2. a classical planning-based approach for creating service compositions, using the PDDL language for domain and problem specification;

3. autonomic workflows to handle non-determinism and enact proper service re-composition as a reaction mechanism;

4. WS-BPEL as the language for describing the resulting service composition.

2.2 Context-awareness in Service Composition

Context-awareness in information systems has been widely acknowledged as an enabling solution for improving how software adapts to dynamic changes in the environment where it executes and how the system is tailored to actual users' needs, preferences and expectations [1, 42, 64, 73, 87, 97]. One of the most referred definitions for the word *context* in computer systems is the one of Dey and Abowd [1], reported below:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

This definition is an extension of [87] and [73] and is still frequently used, allowing context to be either explicitly or implicitly indicated by user.

Context-awareness is also crucial in Web service systems [64]. [97] provides a thorough survey and a comparative analysis about available context-aware Web service systems: reported solutions like [4, 19, 103] provide relevant support in terms of context representation, sensing, storage, distribution, reasoning, security and privacy.

In [104], the authors propose a context modeling approach based on ontologies to dynamically handle various context types and values. Ontologies enhance the meaning of user's context values and allow for automatically retrieving relations among the context values. However, context-aware service composition is not addressed.

The authors of [42] propose meta-models and an aspect-based pattern for context-awareness of services. A *Context* is considered as a set of *Parameters* and *Entities* that can be structured in *SubContexts*. *ContextAwareServices* are modeled as services with a set of associated *ContextViews*, each one containing adaptation rules and

actions, to be applied when the rules are satisfied. Based on this model and the aspect paradigm, context-aware adaptations can be dynamically performed by means of adaptation aspects dynamically woven into the core services during service execution. For example, the same service may provide different responses to the same request message if the context changes (e.g. more/fewer data are returned by a search service depending on the battery level or connection mode of the mobile phone producing the request). Despite some similarities with our model, they focus on AOP instead of considering semantic-based service composition.

The aspect-oriented paradigm is also exploited in [59] to support context-aware semantic service composition. These compositions are performed by weaving context aspects within plain compositions to handle dynamic and hard to predict contexts. Contexts are defined as ontology concepts; context configurations are used to associate service messages to context concepts. Plain composite services are modeled as state automata and transformed into context-aware composite services by means of static weaving. Context aspects are weaved before starting the execution of the main service by inspecting the automaton and introducing in it appropriate context services as new transitions. However, the authors do not focus on automatic composition. Plain compositions already exist and are augmented by adding context services to become context-aware. Goal changes or service unavailability, which have to be considered as part of the context, could require context-aware composition to partially or completely change by taking into account new services, not included in the original plain composition. In such situations, the approach is not effective.

Exploiting context is essential also in pervasive environments [80, 113]. In such environments, most service discovery or composition requests are implicit [80], i.e. driven by state changes.

In [70] the authors propose a framework and a related language, namely EASY-L, based on OWL, to support efficient, semantic, context- and QoS-aware service discovery on top of existing service discovery protocols in pervasive environments. In [113], the authors introduce a design process and an architecture for building context-aware pervasive service compositions, while, in [80], the authors propose an approach for personalized service discovery in pervasive environments, based on the concept of multi-dimensional space (HAC, i.e. *Hyperspace Analogue to Context*). Each space dimension represents one relevant context type and a context is a point in the space. A context-aware service has pre- and post- conditions modeled as regions (context-scopes) in the HAC: service pre-conditions have to be satisfied for the service to be executed; post-conditions determine the new context as result of service execution.

User preferences represent user's desired arrival context (i.e. goals), to be reached by means of a context-aware service execution. The paper is mainly related to service discovery following an event-driven model.

According to [97], in the area of context-awareness, several fundamental research challenges still remain to be addressed. Among these: sharing and integrating a common conceptual representation for context, developing sound techniques for managing distributed context, properly using ontologies for modeling context information. Moreover, we think that context-aware composition is a research field only partially explored. Several problems still need to be addressed to put these approaches in the mainstream. In Chapter 4, we propose our solution to address some of these problems.

2.3 P2P Solutions for Scalable Service Discovery and Composition

Service discovery often relies on the use of centralized registries, discovery engines or brokers. However, some approaches have been proposed to perform discovery more effectively and efficiently than centralized solutions. Such proposals exploit distributed architectures to remove bottlenecks of centralized approaches (e.g. single point of failure, low resolution time, low scalability, etc.), by relying on decentralized structured or unstructured P2P overlay networks. Nevertheless, very few P2P-based approaches have been proposed for both effectively and efficiently supporting semantic service composition.

In the following, we review the literature relatively to P2P networks, with particular emphasis on efficient search in unstructured P2P networks. These techniques usually regard keyword-based file search, which is a simpler problem than semantic service discovery. In addition, at the best of our knowledge, similar techniques have been only marginally addressed to solve the problems of semantic service discovery and composition in P2P networks.

Discovering a service that is not present as an atomic service description in a distributed registry, but as one or more service compositions, asks for several related searches over the network (in order to find all the composing services in the right order). Thus, the problems affecting single-resource search techniques in P2P networks significantly grow when performing service composition and demand for a new domain-specific approach to efficient search in P2P overlays.

2.3.1 Search Techniques

Decentralized P2P overlay networks are distributed systems in nature, without any hierarchical organization or centralized control. They are typically divided in two main classes: *structured* and *unstructured* [62].

Structured P2P overlay network have tightly controlled topologies and content is placed at specific locations to efficiently solve queries. Such overlays often use a Distributed Hash Table (DHT) substrate, where data objects (or values) are placed deterministically at the peers whose identifiers correspond to the data objects' unique keys. Node identifiers are uniform-randomly assigned to the peers from a large space of identifiers. Similarly, unique identifiers, chosen from the same identifier space and called keys, are computed from data objects by means of a hash function. Keys are then mapped by the overlay network protocol to a unique live peer in the overlay network.

Some well-known examples of DHTs are Content Addressable Network (CAN) [81], Chord [92] and Pastry [85]. In general, structured P2P overlay networks support scalable storage and retrieval of {*key, value*} pairs, by means of operations like *put(key, value)* and *get(key)*, respectively. However, they only support exact matching and are strongly affected by peer churn [77]. Since the focus of this thesis is not on structured P2P networks, for the reasons highlighted in the next Subsection 2.3.2, we refer to [62, 111] for more information about them.

An unstructured P2P system is composed of peers joining the network with some loose rules and without any prior knowledge of a specific topology to preserve [62]. The resulting topology may have certain properties, though the placement of objects at the peers is not based on any specific topology-related property [94]. Thus, unstructured overlays provide complete flexibility on where resources can be published by the peers; search methods are not based on hash functions and can naturally support non-exact matches.

Search techniques for unstructured P2P overlays are typically categorized in *blind* and *informed* [88, 94].

Blind search schemes employ flooding or random-based techniques to relay queries to peers in the network. Peers keep no information about the P2P network or the probable locations of objects for routing queries. Flooding is the typical mechanism used to send queries across the overlay with a limited scope (e.g. the Gnutella protocol [39]). It consists in a Breadth First Search (BFS) of the overlay network: when a peer receives a query, it returns the results if present, otherwise it retransmits the query to all of its neighbors, except for the sender, until the *Time-To-Live* (TTL)

associated with the query expires.

Flooding-based techniques are effective for locating highly replicated items and resilient to peers joining and leaving the system, but they are not scalable because of the high number of messages exchanged [62, 83]: low bandwidth nodes easily become bottlenecks when handling a high rate of queries and large systems quickly become overwhelmed by the query-induced load [63]. Several approaches have been proposed to address this problem. These include variants of random BFS [25, 30, 53], which perform random neighbors selection based on local degree information; iterative deepening [105], consisting in successive multiple breadth-first searches with increasing depth reduction; random walks and its variants like random k-walk [25] or probabilistic gossip strategies and a two-level random k-walk[49].

In unstructured and large P2P networks, gossip protocols [28, 36, 41] have been proposed as a solution to implement effective message broadcast, supported by their ease to deploy, high reliability, and scalability [34, 100]. Their probabilistic nature may significantly reduce the amount of exchanged messages, if the application context does not require 100% of reliability. Thus, these techniques have been widely used for information dissemination over the Internet [29, 33] (e.g. multi-player games, video streaming application), or exploited in wireless ad hoc and sensor networks [14, 36, 89].

[16] is an example of using gossip strategies to support adaptation in P2P architectures. The authors propose DEPAS, a decentralized probabilistic algorithm for building auto-scaling service systems over multiple cloud infrastructures, in order to cope with high and highly fluctuating request rates. The algorithm is based on a P2P architecture composed of many autonomic services. Each of them is responsible for processing requests coming from the clients and for deciding to self-destroy or replicate themselves by performing local monitoring activities. The autonomic services are organized into an overlay network in which each of them knows a fixed number of other autonomic services, called neighbors. A gossip-based algorithm is exploited to efficiently maintain the overlay, while a decentralized load balancing technique is used to keep the utilization of each autonomic service close to a given threshold. An autonomic service uses the weighted average load of its neighborhood as an approximation of the average load of the system to decide about removing itself or adding a new autonomic service.

Some well-known gossip techniques are: *Fixed-Fanout Gossip* (i.e. GossipFF) [28, 33, 54], *Probabilistic Edge Gossip* (i.e. GossipPE) [36, 89] and *Probabilistic Broadcast Gossip* (i.e. GossipPB) [14, 41]. They are based on the criterion to limit the outgoing

links to use for information forwarding over well-known random graphs, typically used to model large-scale random topologies (e.g. *Bernoulli* [27], *Random Geometric* [75], and *Scale-Free* [2]). They present different performance figures with reference to the network topologies considered.

In [47], the authors use a generic parameter (*effectual fanout*) for comparing GossipFF, GossipPE and GossipPB strategies. They observe, by means of simulation, the trade-offs among dissemination reliability, message complexity and latency, with various kinds of input over Bernoulli, Random Geometric and Scale-Free topologies.

However, the adoption of gossip strategies for P2P service composition is still in its infancy since very few works address the problem of composition efficiency. In general, blind methods either do not scale well, because of the high message overhead, either exhibit high resolution time since the query routing is not guided towards the target peers during random walks and the same paths can be repeatedly explored.

Therefore, *informed search* algorithms have been introduced to improve search performance by using network information during query routing.

Two traditional examples of informed search are Local Index (LI) and Routing Index (RI) methods. In LI-based algorithms, like [84], each node propagates the index of its available files to nodes within radius r from itself. Each node stores the received indices and processes the query on behalf of all nodes within r-hops distance. If the required resource is not found, the query is flooded to the next layer of neighbor nodes. In the Routing Index (RI) algorithm [22], files are categorized according to their subjects and an index is created based on an estimation of the number of accessible files in each category for each neighbor. When a received query cannot be solved, each node forwards it to a subset of neighbors having access to a larger number of files in the same category of the requested file. This method is not suitable for today's highly dynamic P2P networks. Lightflood search [51] is another mechanism in this category which uses topological information in its forwarding decision making strategies. This algorithm has two stages. The first stage is a standard flooding with a limited number of TTL hops, where a message can spread to a sufficiently large scope with a small number of redundant messages. In the second stage, messages are only propagated along a tree-like overlay that is built in order to cover the whole network, thus significantly reducing the number of redundant messages.

In the more recent work [35], the authors introduce a generalized probabilistic flooding algorithm for resource discovery, which considers resource distribution and heterogeneity of the number of direct outgoing links to other peers to reduce message overhead. Similarly to our proposal, in this method, each peer selects a neighbor

as the query's next hop, based on a function of its degree and a function of its neighbors' degrees. These two functions also depend on the distance from the query's originator to control the amount of generated load. However, this strategy has not been exploited up to now to support service discovery or composition and, differently from our solution, does not consider semantics in resource matching In addition, it does not exploit knowledge about already solved searches to perform new similar searches more efficiently.

Intelligent BFS [53, 106] is an example of informed search with adaptive information. Each node creates a table and stores the number of returned results for each query from its neighbors. The next time it receives a query for the same file, it sends the query to the neighbor with the largest number of returned results. The performance of this method improves over time as nodes gain more information about their environment. However, this algorithm has a poor performance in dynamic networks as it is not designed to adapt itself to departure of nodes and is based on exact match.

Adaptive Probabilistic Search (APS) [98] is another adaptive informed search in which each node keeps a local index for each queried object and for each directly connected node. At the beginning of the search, nodes have no information about the network and, therefore, the probabilities of all neighbors for a requested file are equal. To accelerate APS, the requesting node sends k-random walkers to its neighbors. Receiving nodes forward the query to neighbors with the highest probability. If a query hit occurs, the query returns to the nodes it has traversed and increases their probability value. If a query failed to find an object, it decreases the probability value of nodes on its path back to the requesting node. This method can achieve high success rate over time, however, it has poor performance in dynamic environments.

Ant Colony Optimization (ACO) search has been the base of some search protocols in P2P networks, like in [52, 61]. This mechanism sends the query to a neighbor with higher probability of answering the request. In AntSearch [52], each peer maintains its success rate of previous queries and records a list of pheromone values of its immediate neighbors. At the beginning, each ant walks randomly in the network and leaves a small amount of pheromone on visited nodes. Based on the pheromone values, a query is only flooded to those peers which belong to promising paths for finding results. When a query hit occurs, a query-hit message takes the same path to the requester and updates routing information. This iterative process stops when the desired number of results is returned, or all neighbors have been visited.

A special kind of unstructured P2P overlay network, typically referred as a hybrid P2P, exploits peers with special capabilities, called superpeers, and has been

originally introduced (e.g.[18, 65]) to improve the performance of the first Gnutella specifications. This overlay classifies peers into Superpeer or Ultrapeer (SP) and Ordinary-Peer (OP). SPs have usually superior performance or knowledge than OPs and can provide discovery services for OPs, thus reducing the number of messages generated to find resources. OPs connect to only one SP at a time, for sharing their resources and participating in the network, while SPs form an unstructured P2P overlay network and maintain links to their OPs.

In their recent work [46], Hsiao and Su have proposed an approach based on overlays to improve search results in unstructured P2P networks that exhibit the power-law file sharing pattern. To achieve this goal, a random graph is reconstructed to satisfy three properties: first, peers try to select neighbors that are most similar to themselves. Second, the diameter of the network should be as low as possible in order to propagate messages between peers rapidly. Third, each peer should send the query to one of its neighbors which is most similar to destination than the node itself, hoping that the query is forwarded along an overlay path that gets more similar to destination at each hop. ISI [44] is another mechanism which tries to reorganize network topology by replacing randomly selected neighbors with peers that are more cooperative in the network. In DiffSearch [102], an overlay of ultrapeers is created in a way that each leaf peer has at least one ultrapeer neighbor. A leaf node can promote itself to an ultrapeer when the number of its visited shared files reaches a threshold, based on the idea that peers with high query answering capabilities are given higher probability to be queried. Diff-Index [40] suggests using the online time heterogeneity to enhance search efficiency in unstructured P2P networks. This technique uses the fact that nodes sharing a great amount of files tend to stay in the network for a longer time, and therefore, by querying this small portion of nodes, success rate is increased and search traffic is decreased.

To reduce the number of probed hosts and consequently reduce the overall search load, [20] proposes to replicate data on several hosts. The location and the number of replicas vary in different replication strategies. Thampi et al mention in [94] three main policies: *owner replication*, in which the object is replicated on the requesting node and the number of replicas increases proportional to popularity of the file; *random replication*, in which replicas are distributed randomly; *path replication*, in which the requested file is copied on all of the nodes on the path between the requesting node and source.

2.3.2 Service Discovery and Composition

To improve scalability, dynamicity and robustness of the Service Oriented Architecture paradigm, in recent years, some researchers [77, 82, 86, 112] have exploited DHT-based networks (e.g. Chord [92]) for service discovery. The authors of [77] identify both advantages and disadvantages when using DHTs for semantics-based discovery. DHTs force to place services on hash-suggested peers, reduce the discovery time to $\mathcal{O}(log(n))$ but can retrieve only resources that exactly match the query (semantic matching is not supported). They need a significant overhead for topology management, especially in networks where peers and services appear and disappear frequently (e.g. Chord churn complexity is $\mathcal{O}(log^2(n))$ [92]), and impose a fixed network structure. These limitations make it difficult to satisfy the requirements elicited in Section 1.1.

Consequently, unstructured P2P networks, and in particular Semantic Overlay Networks (SONs), are recently acquiring growing consensus [12, 31, 32, 37, 77, 114] for supporting semantic service discovery, due to their flexibility, fault tolerance and semantic matching capabilities: any peer can publish its services in a local repository, and semantic queries of the desired service may be effectively routed towards the right registry.

Originally proposed in [23], SON is a paradigm for organizing peers and enhancing content-based search. A SON is a virtual network designed to connect peers that are semantically related, i.e. their respective content are semantically similar. A recent paper [77] proposes a hybrid solution that mixes DHTs with SONs for performing service discovery. The role of DHT is limited by the hash function used to map services to peers: it works only for exact matching and selects the peers where services are published. Consequently, the authors have to rely on SONs to outcome these limitations. In addition, they do not consider the service composition problem.

The authors of [12] present an approach based on semantic P2P overlay networks with superpeers. However, their JXTA-based discovery framework performs only simple service discovery, with no possibility to retrieve composite solutions. In addition, query propagation relies on simple flooding inside the groups managed by superpeers.

The CHOReOS[6] [6, 101] project is aimed at leveraging model-based methods and SOA standards for making choreography development a systematic process for the reuse and the assembling of services discovered within the Internet. The CHOReOS core consists of a model-based development process and associated synthesis methods, and a middleware for choreographing services. As an initial basis for their work,

[6] http://www.choreos.eu/

the CHOReOS team has proposed in [5] an approach for mining service abstractions, i.e. semantically compatible services, out of sets of available service descriptions. Experimental results have shown a high percentage of useful abstractions that can be extracted from the services available on the Web and the importance of taking into account, as addressed by our approach, developer/designer preferences and/or business requirements during the mining process. Starting from these results, the CHOReOS team has focused on offering a development process and a supporting Integrated Development and Runtime Environment (IDRE) to provide a comprehensive platform for the design, the deployment, the execution and the evolution of Ultra Large-Scale (ULS) service choreographies. The CHOReOS IDRE provides model-driven techniques and tools that allow domain experts to specify requirements and to model choreographies. Then, the choreography is automatically synthesized into distributed coordinating logic. A Choreography Synthesis Processor allows for deriving the Coordination Delegate artefacts from the BPMN specification of the choreography, by exploiting model transformation and service discovery. The derived coordination delegates can be deployed on the network nodes and allow for enacting the choreography by realizing the distributed coordination logic among the discovered services. It is important to point out that, differently from our approach, the CHOReOS solution assumes choreographies to be directly specified by domain experts and does not provide them with tools for automatically composing services together in order to satisfy complex user requirements. Therefore, the authors' focus is on how developing coordination logic for enacting choreographies. Even though in the following we assume to work mainly with orchestration logic, our composition tool and P2P approach could complement the CHOReOS solutions, by providing an additional support for the initial choreography specification. In addition, in the P2P approach, our solution could be extended to directly create the coordination logic for choreography during the exploration of the P2P network for service composition.

Very few works address the problem of efficient service composition in unstructured P2P networks, since most proposed solutions for both service discovery and composition are based on flooding, which generates an important message overhead, causing high routing costs and low scalability [83], as highlighted in Subsection 2.3.1.

In [114] and [31], our research group have firstly proposed semantic composition overlay networks to improve the efficiency of P2P discovery of service compositions. We foster the creation of groups of peers, hosting the services previously used to create compositions. The groups are managed by superpeers, which are in charge of addressing the service requests sent to the groups that host compositions, thus increasing the

probability of finding the desired services in a shorter time. As Chapter 3 will detail, service composition is performed by exploiting traditional approaches, typically applied to centralized registries [9]: we adopt AI planning techniques, and semantics to improve automation capabilities.

In [37], the authors propose to use a caching mechanism to increase efficiency in solving service queries through service composition. The approach has some similarities with our solution, but differently, when no entry is present in the composition cache, the resolution strategy still relies on flooding, generating high message overhead. In addition, no strategy for propagating service requests towards the most promising directions (semantic routing) is provided to lower resolution time, as allowed in our approach by exploiting multiple semantic overlays managed by super-peers.

2.3.3 Bidirectional Service Composition

Very few works address the problems of automatic bidirectional service composition and, at the best of our knowledge, no work proposes to use it in a fully distributed and P2P approach, neither exploits bidirectional search to suggest existing gaps to service providers.

In [107, 108], the author introduces the concept of composition distance between Web services and proposes a heuristic bidirectional state-space search algorithm based on it. In general, a Web service (i.e. S_j) is *composable* with another one (i.e. S_i) if S_j's inputs can be derived from S_i's outputs (*composability relationship*). By analyzing the composability relationship over the set of available services, the author proposes to build a relationship graph. The composition distance between S_i and S_j is defined as the shortest path distance between the corresponding vertices over the relationship graph. By knowing the composition distance matrix, a bidirectional search can be performed to explore the service space both from the goal to the initial state and in the opposite search direction in order to find a solution. This approach, however, cannot be realistically applied to a large and dynamic P2P network, since it is supervised. Moreover, it only takes into account the input/output interfaces of the service, without considering any semantic information about pre-conditions or effects.

The authors of [99] define a domain independent bidirectional heuristic, based on the notion of dependency graph to model Web services. As in our approach, the authors rely on the OWL-S ontology for Web service specification (using the *Service Profile* and the *Service Model* sections) and Web services are represented as transitions from an input state to an output state. However, differently from our approach, service

pre- and post-conditions are not taken into account. The information contained in the OWL-S descriptions is used to build the dependency graph. Their strategy tries to adapt a well-known heuristic bidirectional search algorithm [90] to the Web service domain. The original algorithm provides two search fronts, one going from the initial state to the final state and another one going from the final state to the initial state, aimed at finding a shortest path, if any. The heuristic function for a node N (i.e. $h(N)$) belonging to one of the two fronts depends on the sum of two costs: (i) the estimate of the minimum distance between nodes N and Y (a node in the other front), and (ii) the cost of going from the goal node to Y. The idea is to select a node Y from the set of nodes in the other front such that $h(N)$ is minimized. Since a shortest path may not be always meaningful in the service composition context, the authors propose a simpler heuristic function that does not use estimates of the minimum distance. The reported results show the advantages of their heuristic over some well-known search algorithms. However, as in [107] and [108], the author considers only a local repository containing the Web service specifications required for constructing the dependency graph.

In [37], the authors propose two algorithms to perform P2P composition: a backward-chaining algorithm (from goal outputs to goal inputs) and a forward-chaining algorithm (from goal inputs to goal outputs). Besides the problems related to the use of a simple flooding solution for query forwarding, previously discussed, the authors do not propose any combination of the two strategies that could be considered as a bidirectional approach.

Chapter 3

Executable Service Compositions

Web services and Service Oriented Architectures allow for developing Web applications by combining distributed services hosted by servers in different administration domains. We refer to this new kind of large-scale, distributed application as "multi-organization Web application".

With the term large-scale, we intend the involvement of a large numbers of services available throughout the Internet. Services can be modified or replaced; they can disappear, and new services with different features may become available.

Thus, multi-organization Web applications require a new level of exception handling to address the variability of the execution context.

In order to illustrate the idea of service composition, we introduce an example scenario related to emergency handling in natural disaster management. The considered example, which will be further detailed in Section 3.5, is about population alert by using multiple communication channels like mobile networks, SMS, MMS, automatic calls, TV, etc.

A possible service composition for alerting people, living in the specific geographic area where the disaster occurred, could consist in the following subprocesses (each constituted by one or more services), to be executed concurrently:

1. *Landline Alert*: getting the list of home telephones in the area using a White Pages service; then executing an automatic call center service to call the retrieved phone numbers while concurrently producing a list of citizens without

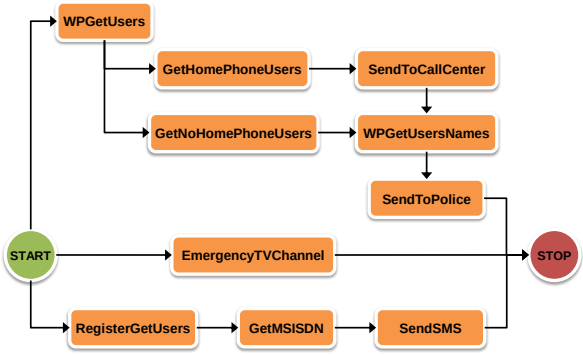


Figure 3.1: An example service composition for people alert in natural disaster management

home phone; notifying the local police station with the list of citizens that were not warned (as they do not have home phone or because they did not answered the call) in order to physically alert them at their habitations.

2. *TV Broadcast*: executing a service to transmit a broadcast alert message on TV channels.

3. *Mobile Alert*: executing a service to retrieve the personal details of all the people living in the area; then getting their mobile numbers (MSISDN) from telecom companies, by means of one or more concurrent services, and finally sending SMS (another service).

Fig. 3.1 graphically represents the service composition.

In the following, we mainly focus on automatic service composition in the context of autonomic workflow by presenting a tool able to generate concrete and runnable compositions starting from a repository of service descriptions, a domain ontology and constraints. In particular, the domain is expressed as a set of WDSL service descriptions annotated with OWL-S, whereas the target runnable compositions can be generated either in WS-BPEL or in XPDL.

The rest of the chapter is organized as follows. In Section 3.1, we briefly report on the main concepts related to the autonomic computing vision, useful to deal with the high dynamicity of service environments. Section 3.2 discusses our general approach for automatic generation of service compositions. In Section 3.3, the architecture of the proposed tool for service composition is described. A detailed description of the WS-BPEL serializer is provided in Section 3.4. Section 3.5 reports on the use of the

proposed tool for the example of automatic service composition previously described, together with a performance analysis for the considered composition problem.

3.1 Autonomic Workflows

Autonomic computing is about the idea of systems that can automatically and autonomously manage themselves according to high level objectives defined by system administrators [45]. The central element in the implementation of autonomic computing is the autonomic manager, also called autonomic controller. Autonomic managers work in a closed loop with the systems or components they control: they sense the environment and the main variables of the controlled system, and then apply control actions to modify it. In between sensing and acting, controllers implement activities that logically form a loop, usually called the MAPE-K loop [48].

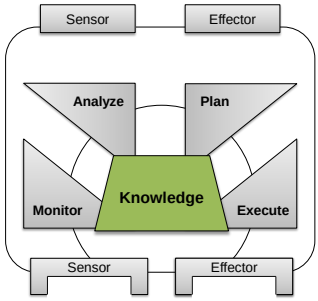


Figure 3.2: Autonomic manager

In the MAPE-K loop of Fig. 3.2, controllers monitor data (M), analyze (A) them to plan (P) a suitable control strategy, and execute (E) the strategy by implementing the control actions on the system. This thesis focuses mainly on the two central activities of the loop, i.e. analysis and planning, and on how dynamic knowledge, i.e. the context, may affect such activities.

The high level objectives that controllers try to achieve may concern several aspects, called self-* properties, that include: optimization, configuration, fault tolerance and security. Self-optimizing systems aim to maximize some utility function of the system; self-configuring systems adapt the software architecture to dynamic environments; self-healing systems recover from failures; self-protecting systems deal with security attacks. Simple self-* systems usually deal with only one property, while more advanced solutions deal with many properties at the same time. In this thesis, we narrow our scope to implementing self-configuring abilities for service compositions.

In order to apply the autonomic vision to service compositions, we have adopted the concept of *Autonomic Workflow* (AW), previously defined in [78, 95] and supported by our Semantic Autonomic Workflow Engine (i.e. SAWE [78]).

An AW denotes a composition of automatic or manual services that is able to proceed towards the goal even if external events significantly change the execution context. To survive the changes, a service composition needs to be modified, taking into account the new environment. The autonomic workflow life-cycle is reported in Fig. 3.3.

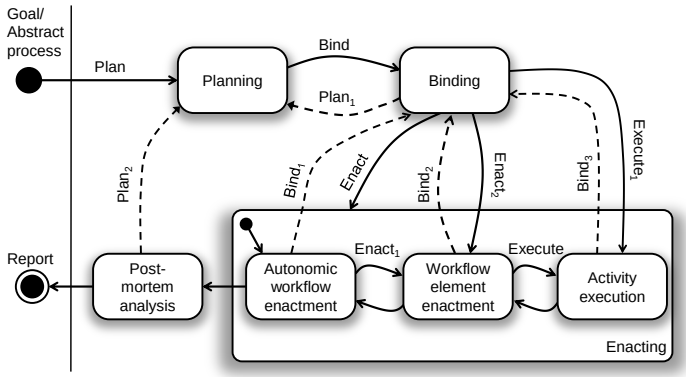


Figure 3.3: Autonomic workflow, life-cycle [78]

An AW is supervised by an autonomic system that should be able to self-manage workflow execution (see Fig. 3.4). Even if several aspects of a workflow could be changed to react to an anomaly occurring at run-time, in this thesis we focus our attention on the planning and re-planning phases (*plan/plan1/plan2* in Fig. 3.3).

To this purposes, an important role is performed by the *configurator* (see Fig. 3.5), a component of an autonomic composition engine that is in charge of implementing self-configuration of service compositions through the knowledge coded at design-time or collected at run-time.

The configurator acts on every aspect of a concrete service composition by changing the overall composition graph to make it runnable within the new conditions. To this end, it can: change a link between an activity and a concrete service (i.e. *re-binding*); insert, delete or replace an activity; change the endpoints of a transition; substitute an activity with a sub-process that is able to perform the same actions and to produce the same effects on the external world. The configurator exploits some internal components to generate or change compositions.

An important component of the configurator is the *Composer*. It can be used

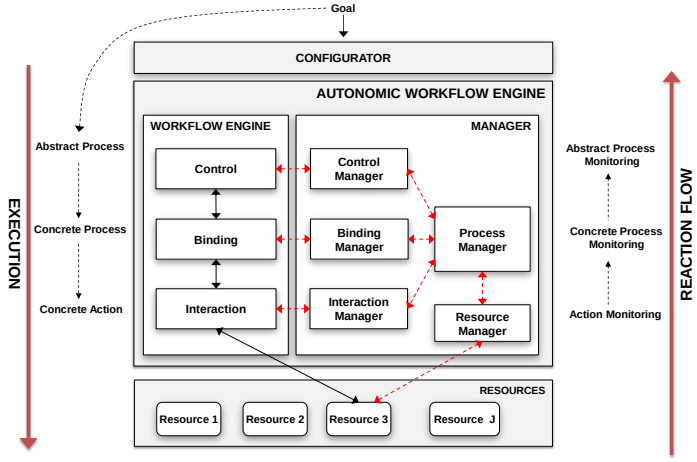


Figure 3.4: Autonomic workflow, main architecture (SAWE engine)

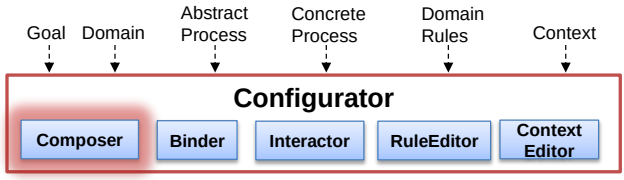


Figure 3.5: Configurator component of the SAWE engine

either for the initial definition (*plan*) of a service composition or to *re-plan* an already defined composition, which could need to be changed completely or in part to react to external events.

The composer exploits *planners* to transform the descriptions of *goals* and *service domains* into *abstract processes*.

Abstract processes can be further concretized through the *Binder*, a component in charge of linking an abstract activity with a concrete service, thus generating *concrete compositions*.

As Fig. 3.5 shows, the approach needs several information to support its autonomy during execution. In particular, the *Domain* refers to formalized knowledge related to the specific application domain where a service composition takes place. It is an ensemble of: 1) concepts and relationships (ontology) that enrich service descriptions and 2) causal constraints (e.g. pre- and post- conditions) that cause services to be correctly ordered in a service composition.

Domain rules, instead, represent higher-level knowledge that is useful to express

some constraints that invalidate or validate the generated plans, so reducing the space of admissible solutions.

Finally, *context rules* are used to observe the external world during planning or execution in order to generate new knowledge that potentially can enrich the service domain, user goals and domain rules (see Chapter 4).

3.2 Composition Process

Fig. 3.6 shows a graphical representation of the notion of Web service composition in workflow design. In the top of the figure, workflow tasks represent complex activities, which can be implemented as service compositions. Automatic support for generating executable service compositions is the focus of this chapter.

Starting from a task description (the problem, e.g. task 4 in the picture), a set of candidate Web services (the service domain) is inspected in order to find a chain of services (a plan), which is consistent with the task description. Consistency means that, starting from a provided description of the initial state (i.e. the set of predicates which are true before the task beginning), the chain is able to reach the goal state (i.e. the predicates in the goal state have to be true at the end of the service chain).

The initial state includes a description of all the input data and the pre-conditions known at the beginning of the execution, while the goal state specifies the desired outputs and the post-conditions to be obtained by the task execution. The problem task may correspond to a single task in an already defined business process or to the whole business process. The first case can be related to a re-planning process, where

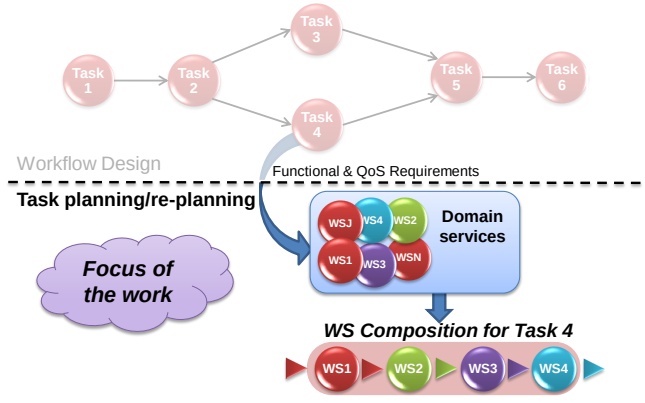


Figure 3.6: Web service Composition Process

an activity of the process is replaced by a service composition, while the second case relates to the planning process, which is the definition of a new complete business process, satisfying some user's specific business needs.

We define (see also [11, 17, 114]) *service composition* as an AI classic planning problem with a deterministic transitional model, represented as the 5-tuple: $\{S, s_0, s_f, A, \Gamma\}$, where S is a finite set of states, A a finite set of available actions, s_0 the initial state and s_f the desired final state. A state can be either the set of necessary conditions for the execution of a service operation (pre-conditions) or the set of the effects produced by the execution (post-conditions). Problem inputs and desired outputs can be expressed as predicates in the initial state s_0 (describing available knowledge) and predicates in the s_f final state (describing additional knowledge produced by service execution), respectively [55].

Two states are connected by a transition if an operation exists that can be performed to transit from the first state into the second one. An action, generated by a service operation, triggers a transition from its pre-conditions to its post-conditions. $\Gamma \subseteq S \times A \times S$ represents this transition relationship. We call *service goal* G (or simply goal) the pair (s_0, s_f), denoted as $s_0 \rightarrow s_f$ in the following, representing the desired state transition from the initial state s_0 to the final state s_f. We call *queries* the user requests for any service (atomic or composite) able to satisfy the goal transition carried by the query itself.

We adopt the *Closed World Assumption* (CWA) to deal with the semantic description of the state transitions performed by domain services and of the desired service represented by the goal. By using these assumptions, we do not need to execute services to know their actual effects.

In addition, we assume context changes are reasonably slower than the time required for a typical composition problem to complete (*offline planning* assumption) [91]. Hence, the composition process may be completed with respect to the states at the time the composition is started, being the final composition still meaningful.

We assume both the service domain and the problem to solve are described by using the OWL-S ontology, according to the IOPE semantics. The set of semantically described services (the domain) has to be known before starting the composition process and provided as an input to the system together with the semantic description of the problem. The service domain can be retrieved manually or in a (semi-) automatic way, by relying on a service registry and exploiting the semantic description of the goal to reach during a matchmaking process.

Any service operation is associated to an OWL-S file, which includes the definition of an OWL-S atomic process (<*process:AtomicProcess*>), specifying the inputs, the outputs, the preconditions and the effects (IOPE) of the specific operation performed by a service. Inputs and outputs may refer to concepts imported from OWL ontologies or XML-Schema data types. Preconditions and effects are specified within the OWL-S process description in the <*process:hasPrecondition*> and the <*process:hasResult*> sections respectively. We use the Semantic Web Rule Language (SWRL)[1] to define these conditions.

The problem is considered as a desired operation and is specified as an OWL-S process with inputs, outputs, preconditions and effects, like a service operation. Inputs and preconditions make up the initial state, which is data known to be available and predicates known to be true when the related task starts, while outputs and effects make up the goal state, which includes desired data outcomes and true predicates at the end of task execution.

By following the same approach proposed in [55], input/output parameters contained in an OWL-S profile are converted into additional service pre-conditions and service effects, respectively, during the discovery phase. To this purpose, as will be detailed in Section 3.3, for each input parameter we consider a specific predicate (*hasKnowledge*), defined in our planning ontology, having the input parameter (concept) of the OWL-S profile as a predicate variable. For example, if a service has an input tied to the "EMailAddress" concept in the OWL-S profile, we will consider the predicate *hasKnowledge(EMailAddress)* during the discovery phase as an additional pre-condition. Similarly, each output parameter of the OWL-S profile is treated as a service effect by means of the same *hasKnowledge* predicate, having the output parameter (concept) of the OWL-S profile as a predicate variable.

Fig. 3.7 describes the main logic flow associated to our composition tool.

The OWL-S service domain and problem descriptions represent the main inputs for the composer. Moreover, it is possible to specify simple business rules that have to be validated on the generated plans. Provided inputs are then processed and transformed into proper internal data structures.

The processed information (service domain and rules) is captured into internal incremental knowledge bases, used to quickly solve future requests related to the same domain or business rules. The internal input data are then supplied to the planner component to find some solution plan, or *abstract plan*, (set of activities) from the domain, satisfying the specified problem and business rules (through the plan valida-

[1] http://www.w3.org/Submission/SWRL

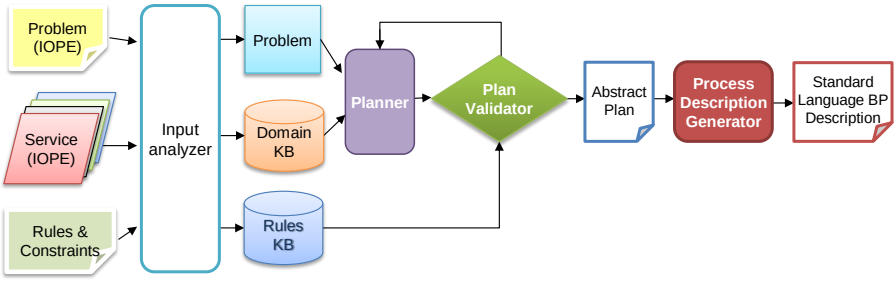


Figure 3.7: Planner-based composition process

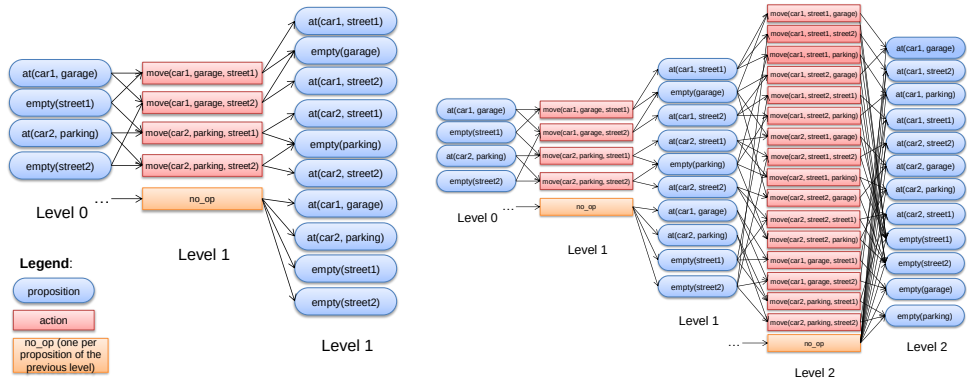
tor). If the validation fails, a new plan may be found. The generated abstract plan is finally bound to concrete Web services and transformed into a standard business process representation (e.g. BPEL, XPDL), or *concrete plan*. The produced description can be executed on our SAWE Engine or on standard execution engines, like JBoss RiftSaw[2], Apache ODE[3].

In our research, we focused on software and data engineering aspects by leveraging on existing results from the AI field. In particular, to generate abstract plans we selected the Graphplan algorithm, briefly described in the following.

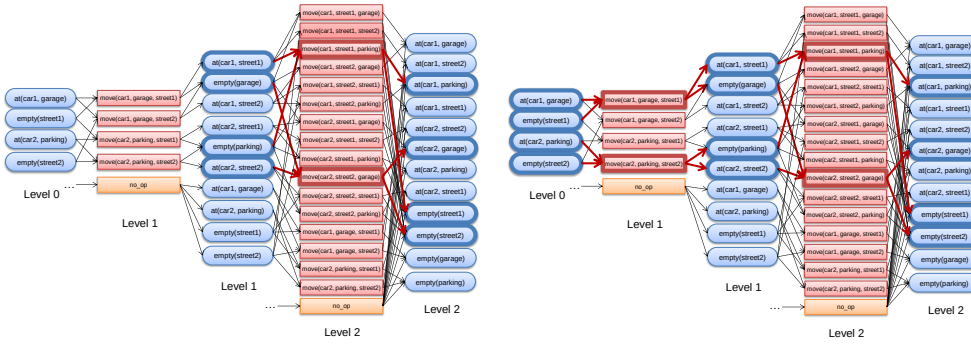
The Graphplan algorithm [13] was proposed by Blum and Furst to efficiently solve planning problems in STRIPS-like domains (made of objects, operators and propositions). It consists of two interleaved phases: a forward phase, where a data structure called *planning-graph* is incrementally extended, and a backward phase where the planning-graph is searched to extract a valid plan. The planning graph can be created in a polynomial time, with respect to the size of the problem domain, while the search phase has an exponential complexity in the worst-case.

The planning graph structure is organized in multiple levels, each containing proposition or action nodes, connected by three kinds of edges: pre-condition, add and delete edges. The 0-level is a propositional list, containing one node for each proposition in the initial state of the problem. Each other level contains both a proposition and an action node list. The 1-level action list will contain all the actions which can be executed given the 0-level propositions, while the 1-level proposition list will include all the effects of the 1-level actions. In general, given a k-level planning graph, the extension of the structure to level $k + 1$ involves introducing all actions (*no-operations* included), whose preconditions are present in the k-th level proposi-

[2]http://riftsaw.jboss.org
[3]http://ode.apache.org



(a) Planning-graph forward expansion



(b) Backward search on the planning graph

Figure 3.8: Graphplan example

tion list. The $k+1$ propositional list includes all the propositions added or deleted as effect of the actions belonging to the $k+1$ level. Since no-operations (a.k.a. *persist operations*) are included in the k-th action list, all the proposition from level k will also be contained in level $k+1$ proposition list. The planning graph construction takes into account mutual exclusion constraints among actions and propositions, which are propagated over the levels of the graph.

The search phase on a k-level planning-graph starts by searching, at level k, the propositions corresponding to problem goals. If all the goals are not present, or if they are present but a pair of them are marked as mutually exclusive, the search is abandoned right away, and the planning-graph is grown another level. For each of the goal propositions, an action from the level k action list, supporting it with its effects, is selected. Mutual constraints are considered in this step, in order not to select actions for supporting two different goals which are mutually exclusive. If they

are, backtracking is performed to select new pairs of actions. At this point, the search process is recursively called on the $k-1$ level planning-graph, with the preconditions of the actions selected at level k as the goals for the $k-1$ level search. The search succeeds when level 0 is reached and the selected actions for each level represent a *Partially Ordered Plan* (i.e. *POP*).

Fig. 3.8 reports on a simplified example (delete actions and mutual exclusion relationships are not shown for the sake of readability) describing both the planning-graph construction (Fig. 3.8a) and the planning-graph search phases (Fig. 3.8b), related to a car-parking problem. Specifically, the example describes how the goal "*at(car1, parking) and at(car2, garage) and empty(street1) and empty(street2)*" can be solved from the initial state "*at(car1, garage) and at(car2, parking) and empty(street1) and empty(street2)*".

3.3 Tool Architecture

The main composition process, described in the previous paragraph, has led to a specific architecture for composition tool that is detailed in Fig. 3.9.

The most relevant components are:

1. The **OWL-S Analyzer**;
2. The **Planner**;
3. The **Plan Validator**;
4. The **Plan Converters** (or serializers).

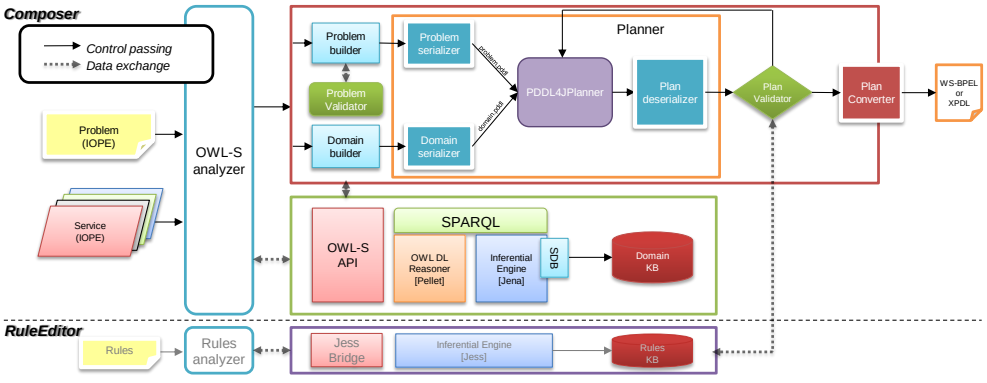


Figure 3.9: Detailed Composer Architecture

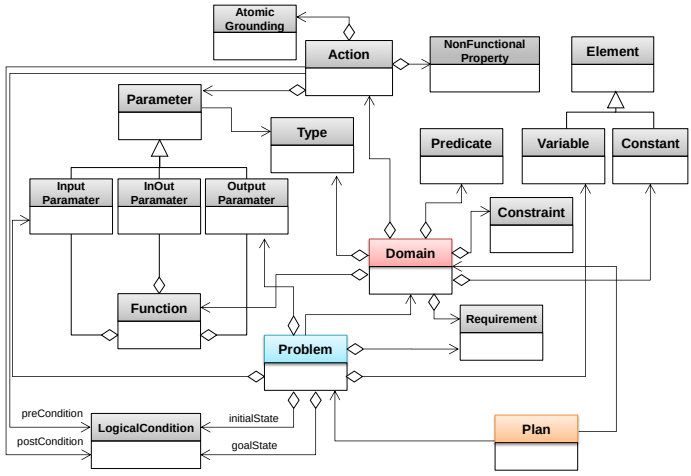


Figure 3.10: Problem and Domain Model

In order to introduce flexibility in the proposed planning-based approach to composition, we have defined a **Meta-model** for describing all concepts, and their relationships, which define our notion of autonomic workflow.

Fig. 3.10 represents a `Domain` and a related `Problem`, while Fig. 3.11 gives a detailed description of the concept of `Plan`, composed of a `Workflow`, which includes simple or complex (composite) `Activities` and several control flow structures as `Parallel` and `Sequence`. The `Plan` element joins the two models.

The meta-model is used to define abstract representations of the service domain, the problem under analysis and workflow plans for solving it in the domain. By defining converters (problem and domain serializers) from the meta-model to planning specific representation languages, it is easy to support several planners. This way, the composer is independent from specific planning tools and languages.

Since PDDL represents the de-facto standard for classical planning problems and there are several planning systems supporting this language, in the current implementation of our tool, we focused our attention on this language and implemented specific converters from the meta-model to the PDDL 3.0 specification and vice-versa.

The **OWL-S Analyzer** is the component responsible of analyzing both the OWL-S files, which describe the available services in the planning domain, and the OWL-S of the problem to solve. This component parses the provided inputs and converts them to an instance of the meta-model. It also integrates reasoning capabilities about semantic concepts referred in service and problem descriptions.

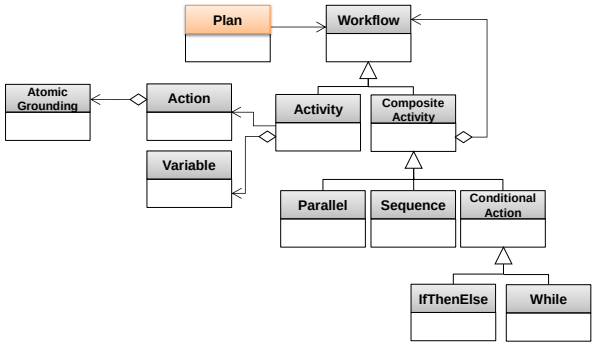


Figure 3.11: Workflow Model

The main rules used by the analyzer to generate a meta-model instance from OWL-S files are the following:

1. The service operation name (<*service:Service*>) defines the name of a new `Action`.

2. The name of input and output parameters (<*process:Input*> and <*process:Output*>) of the atomic process describing the service operation defines the name of the action `Parameters`. Associations between the `Action` instance and its input/output `Parameters` are introduced.

3. The types of input and output parameters (<*process:parameterType*>), possibly referring ontology concepts, define new `Types` of the domain object and are associated to the corresponding `Parameter` objects.

4. SWRL conditions, defining preconditions or effects of a service operation over its parameters and constants (<*process:hasPrecondition*> and <*process:hasEffect*>), are used to define domain `Predicates`, related to action `Parameters` and associated to `Action` objects as `LogicalConditions`, with the role of preconditions or effects respectively.

5. Any action input parameter, retrieved from an OWL-S process, generates a *hasKnowledge* predicate, added as precondition for the relative action object and having the input parameter associated as a predicate variable.

6. Any action output parameter, retrieved from an OWL-S process, generates a *hasKnowledge* predicate, added as effect for the relative action object and having the output parameter associated as a predicate variable. The *hasKnowledge*

predicates for the input/output parameters are added in order to consider parameter dependencies among planning actions during plan generation.

7. Any other element in the OWL-S description, which is not an input/output parameter, neither a type name, defines a new domain `Constant`.

8. References to WSDL information, required in the later phase of *Plan Serialization* for describing an action plan in an executable standard business process representation, are retrieved from the OWL-S grounding section (<*grounding: WsdlAtomicProcessGrounding*>). This section contains information like a WSDL document URI, a *portType*, an *operation*, an *inputMessageMap* and an *outputMessageMap*, which specifies how an OWL-S atomic process maps to a concrete Web service. This information is stored by the OWL-S Analyzer in an `AtomicGrounding` object associated to the action related to the OWL-S described service operation.

In a very similar manner, it is possible to build the meta-model *Problem* object, by applying the previous specified parsing rules to the problem OWL-S description. Obviously, no grounding section is supposed to be available for the OWL-S problem description, since the problem is abstract and still to be solved with a concrete combination of Web services. For this reason, the last rule does not apply to the case of the OWL-S problem parsing.

The **Planner** is the component deputed to the processing of domain and problem inputs in order to produce a plan of domain actions satisfying the problem. To this end, several solutions are available. We implemented a PDDL problem serializer, which take as input the meta-model representation of the problem and produce its PDDL 3.0 compliant serialization. Similarly, we implemented a PDDL domain serializer, which does the same on the domain meta-model object built by the OWL-S Analyzer. By having the PDDL representation of both the domain and the problem, a PDDL planner can be used for generating plans.

The current implementation of the tool uses PDDL4J [74] for planning, a product released under the GNU General Public License (GPL), which is based on a Java implementation of the Graphplan algorithm, described in Section 3.2. The PDDL4J output plan is represented in a PDDL-like representation, which is converted back to its meta-model representation (the `Plan` object in Fig. 3.11, through the *PlanDeserializer* component).

The grammar described in Table 3.1 defines the language supported by the Composer to express business rules that has to be satisfied by the generated plans. In

```
EXP ::= EXP_TYPE; | ;
EXP_TYPE ::= RULE | CONSTRAINT
CONSTRAINT ::= ACTIVITY <-> ACTIVITY |
                ACTIVITY ->! ACTIVITY
ACTIVITY ::= IDENTIFIER
RULE ::= ...
```

Table 3.1: An excerpt of the rule language grammar

the current implementation, the rule language supports the definition of dependency (<-> in the table) and mutual exclusion constraints (->!) between pairs of activity. As an example of these constraints, rule A <-> B means that, if the plan contains the A activity, B has to be present in turn and vice-versa. Rule A <-> B means that A and B activities have never to be both present in the same plan.

The **PlanValidator** component has the role to check if the business rules, specified by the user as input to the composer, are satisfied by a plan produced by the Planner.

If these rules are present and the plan does not satisfy them, a new plan has to be searched by the Planner component until rules are satisfied or there are no other feasible plans.

The behavior of the composer, in relation to the described components, is shown in the UML sequence diagram of Fig. 3.12.

The architecture described in this section has been implemented using the Java language. The meta-model in Figures 3.10 and Fig. 3.11 has been implemented through a set of Java interfaces and classes. The `Domain` and `Problem` classes have been equipped with proper methods for serialization into a PDDL 3.0 language representation (PDDL Problem/Domain serializers in Fig. 3.9).

A PDDL plan deserializer has been developed in order to analyze the PDDL plan produced by the PDDL4J planner and generate a corresponding instance of the `Plan` class. The associated instance of the `Workflow` class contains the meta-model representation of the control flow for the plan (see Fig. 3.11).

The workflow is generated by analyzing the partially ordered plans (POPs) produced by PDDL4J and, if the plan contains more than one action, by properly constructing a composite activity. This activity can be a sequence or a parallel, containing other parallels or sequences at any level. Since the POPs produced by Graphplan are simply lists of steps, each containing a list of actions to be performed, any list of actions for a specific step is analyzed in order to identify parallel branches possibly ranging over more than one single step. This condition occurs when, in the steps following the one currently being analyzed, there is no action whose input parameters or pre-conditions depend from the output parameters or post-conditions of the action

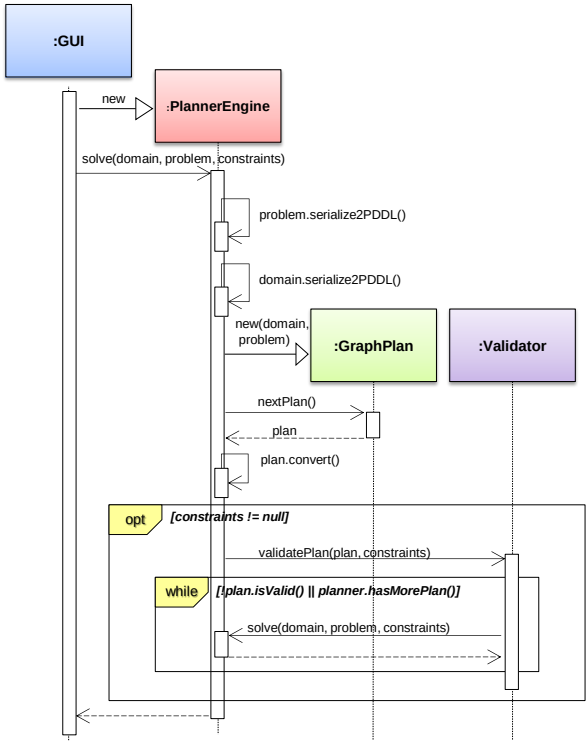


Figure 3.12: Plan generation and validation

in the current step. This analysis is done in order to optimize the plans produced by the PDDL4J planner (i.e. maximize the parallelism) in the final control flow of the business process (i.e. the produced `Workflow` instance for the `Plan` object).

The current implementation of the tool has been equipped with two main serializers: a WS-BPEL Serializer and an XPDL Serializer.

In the next section, the WS-BPEL Serializer is detailed. The XPDL serializer is quite similar, using the XPDL language for the process representation.

3.4 WS-BPEL Serializer

A serializer for a business process representation language is responsible to retrieve all the information required by the corresponding language specification, and possibly demanded by the business process execution engine, to generate a compliant representation of the plan, executable on that engine. For example, the WS-BPEL

representation of a process to be executed on the RiftSaw business process engine is composed of a *.bpel* file, a *.wsdl* artifacts file, a deploy *.xml* file and the set of imported *.wsdl* files.

The first file (*.bpel*) contains the WS-BPEL language description of the business process control and data flows. It also includes the *partner link* definitions. They are concrete references to services interacting with the BPEL business process. Links to WSDL-defined Web services that are invoked by a BPEL process are called *invoked partner links*. Similarly, links to WSDL-defined Web services that can invoke a BPEL process are called *client partner links*. Each BPEL process will have at least one client partner link, since a BPEL process can only be instantiated with a call from a client partner.

The second file (*.wsdl* artifacts file) contains any WSDL information needed to expose the WS-BPEL process as a Web service (e.g. *portType*, *operation*, *input/output messages*, *XML Schema types*, *partnerLinkTypes*). In particular, a partner link type describes the interface used for interaction between services. They are referred in the partner link specification to express the interaction roles. Each partner link type defines at least one role (one-way) or can specify two roles (bidirectional interaction). It references a WSDL port type for each role and can be included in a WSDL file by means of the specific WS-BPEL extension element.

The third one (*.xml* deploy file) is a file describing how to deploy the business process on the engine. It associates concrete endpoints to any provided or invoked service over a partner link and specifies other details (e.g. if the process is active or not) related to the deploy process.

Finally, the set of the *.wsdl* imported files refers to the Web service descriptions used within the *.bpel* file.

In the following, we describe the process of generating a concrete WS-BPEL process definition (concrete plan), executable on the RiftSaw engine, from a non-empty *Plan* object (abstract plan).

Coherently with the structure of a RiftSaw WS-BPEL process previously described, the class diagram in Fig. 3.13 contains classes for a `BPELFile`, a `BPELDeployFile`, a `BPELArtifactsFile` and a set of imported WSDL descriptions. These classes are the object representation of the files required for building a complete BPEL description. They provide methods for their own construction, whose invocation is coordinated by the `BPELSerializer` and, in turn, by the `BPELProcessDefinition` objects, according to a two-levels builder pattern approach.

In order to execute a plan generated by the planner engine on common business

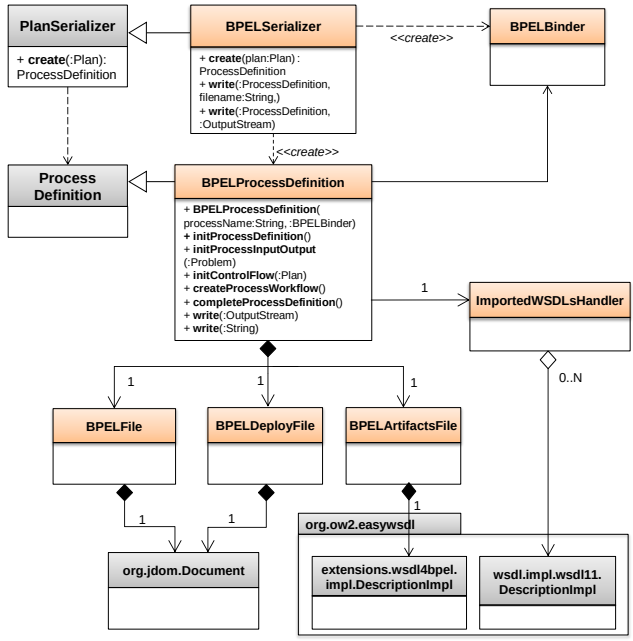


Figure 3.13: WS-BPEL serializer architecture

process execution engines, it is necessary to convert its meta-model representation (the `Plan` object) to a standard business process representation language (e.g. WS-BPEL, XPDL, etc.).

To this purpose, according to the architecture depicted in Fig. 3.9, several **Plan Converters** can be defined, which serialize the meta-model `Plan` object to a specific language representation.

In the design of our tool, the `BPELSerializer` class is in charge of creating the `BPELProcessDefinition` object (and the component objects) and invoking its methods in order to create a complete and executable specification of the BPEL process, corresponding to the activity plan found by the planner component. The `create` method triggers the process generation from the specified instance of the `Plan` meta-model class. The `write` method generates files for the process description, by triggering the `write` methods provided by the component objects. The main sources of information available to the `BPELSerializer` are:

1. the Plan meta-model object and, in particular, the associated Workflow instance;

2. the domain/problem OWL-S files;

3. the WSDL service files, containing the endpoints required to make executable the final business process representation.

The association between the plan actions, the OWL-S and WSDL files lies in the OWL-S grounding section. It is retrieved during the OWL-S parsing and is stored in the `Grounding` of the `Action` class in the meta-model (Fig. 3.10).

WSDL files have been handled (see Fig. 3.13) by using the *EasyWSDL* libraries[4], which offers support for parsing and generating WSDL 1.1 or WSDL 2.0 compliant service descriptions. In addition, the *EasyWSDL4BPEL* library enables the handling of partner links BPEL extensions in service description.

The main steps for the generation of the `BPELProcessDefinition` are described in the following (see Fig. 3.14).

1. `initProcessDefinition` initializes the object representing the concrete BPEL process. Specifically, the generated WS-BPEL process is exposed as a WSDL service, offering one *operation* and one *portType* for executing the activity plan, and the *partnerLink* and *partnerLinkType* for interacting with the WS-BPEL process are created in the proper files (.bpel and .wsdl artifacts files, respectively).

2. `initProcessInputOutput(:Problem)` by using the `Problem` meta-model instance, available input data are used to initialize the process input message, while requested output data are used to initialize the output message of the process. Since the WS-BPEL process is exposed as a Web service, input and output messages for the provided operation are associated with simple or complex XML types specified in the WSDL artifacts file. The concrete structure of the types for the input/output messages is created later (see the `buildInvokeActivity(...)` description), when analyzing the concrete services to be invoked by the process using as input (parts of) the process input message and producing as output (parts of) the process output message, respectively.

3. `initControlFlow(:Plan)` from the meta-model `Plan` instance, initializes the basic control flow of the process: an initial receive input message activity and a final reply output message activity are added to the main sequence of

[4]http://easywsdl.ow2.org

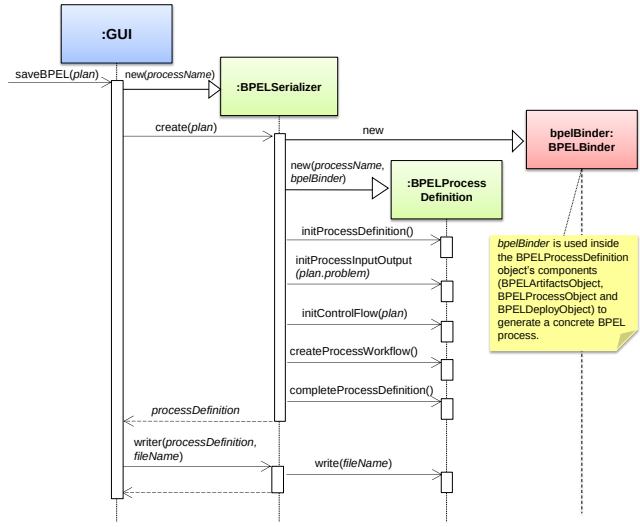


Figure 3.14: WS-BPEL process generation, main sequence of actions

activity of the process, which will contain also the whole workflow associated to the plan.

4. `createProcessWorkflow()` adds to the control flow skeleton, built by the `initControlFlow()` method, the BPEL activities corresponding to the plan instance. The activities can be sequences and parallels (in any combination) of invocation of Web services (BPEL <`invoke`> activities).

   In addition, the `createProcessWorkflow()` method handles the construction of the data flow among Web services. A process memory data structure, managed by the `BPELFile` object, has been used to progressively store, for each different semantically annotated input/output parameter discovered during the generation of the <`bpel:invoke`> activities, the references to the corresponding BPEL global variable and the type field of the WSDL message containing the most updated value for that parameter type. This information is used within <`bpel:assign`> activities preceding service invocations (i.e. <`bpel:invoke`>). They are aimed at preparing the input for the following service invocation, by transferring the correct and up-to-date values from the process global variables to the <`bpel:invoke`> activities input variable.

   Finally, by using the binding component (`BPELBinder`), which relies on the grounding information extracted by the OWL-S semantic descriptions of the

services to be invoked, partner link and partner link type definitions can be completed: partner links are included in the BPEL file, partner link types in the WSDL artifacts file.

5. `completeProcessDefinition()`, completes the definition of the process by generating the binding information for the invoked and client partner links of the process, inserting them into the deploy file (bpel-deploy.xml).

3.5 An Application Example

To evaluate the potential of the tool, we tested automatic service composition on the specific application scenario (*population alert*) introduced at the beginning of this chapter.

3.5.1 Domain Description and Problem Solution

The proposed scenario is about the handling of a hydrogeological disaster (e.g. extreme rainfall, flooding, inundations, etc.) that strikes a human community (city, town, rural areas, etc.). In this scenario, the tool can be used to plan emergency management flows of actions to be executed by a standard workflow engine.

Disaster response management is a particularly meaningful test bed for the tool since action flows must be timely planned and executed. Such actions may be concerned with the use of specific resources (such as telecommunication facilities) and/or the coordination of static and mobile resources (volunteers, policemen, ambulances).

Planning has to take into account resources capabilities, availability and readiness. To this end, the proposed tool is able to generate, automatically, quickly and almost effortlessly, all the planning processes needed. In addition, variations in services availability in the domain, changes in the state of resources, changes in the overall goal, can be immediately considered to get updated plans.

In the reported example, local population is alerted by exploiting different communication channels (e.g. mobile networks, SMS, MMS, automatic calls, TV, etc.). To design a realistic scenario, we have derived it from the analysis of real disaster management plans defined by the organization and emergency procedures of the Italian *Protezione Civile*, the national body in charge of prevention and management of disaster events. The *Protezione Civile* adopts a specific model ("metodo Augustus"); such model emphasizes operational flexibility by using to the largest possible

extent resources located close to the emergency area, and involving all organizations (institutional or not) that can be useful in the specific situation.

The assumption is that such cooperating organizations would have made some or all of their disaster management resources or capabilities available as Web services. Such Web services could be used to access resources, to retrieve useful information, to alert volunteers. Each service would have a WSDL and an OWL-S semantic description of its behavior. OWL-S description refers to a general domain ontology that describes the emergency context.

We considered as cooperating organizations local police, fire brigade, telecommunications companies, white pages, register offices, community volunteering, etc. In order to semantically annotate the available services offered by the different organizations, we defined a specific OWL domain ontology, classifying relevant concepts and their inner relationships. The whole set of domain services refer to several dozens of entities. Some concepts included in the ontology are: *citizen*, *address*, *personal data*, *message*, *mobile or fixed-line telephone number*, *message delivery status*, etc. The semantic OWL-S descriptions of each Web service refer to concepts described in the defined domain ontology. Some examples of the domain services are:

1. *register office service*, to get personal data of the people that must be warned;
2. *white pages service*, to get personal data of people that could be involved in the alert process;
3. *SMS/MMS broadcast*, to alert all the people in a specific neighborhood (available for use by telecommunications companies);
4. *SMS/MMS send*, to send direct messages to specific people;
5. *send phone calls*, using prerecorded messages or via human call center (available for use by telecommunications companies);
6. *alert local police or fire brigade.*

Fig. 3.15 graphically presents the OWL-S description of the *Send SMS* service, which is the archetype of a service made available by a mobile telecommunications company.
OWL-S descriptions are converted into PDDL, the internal language used by the PDDL4J planner. In Table 3.2, we show a sample of PDDL translation, related to the *Send SMS* service.

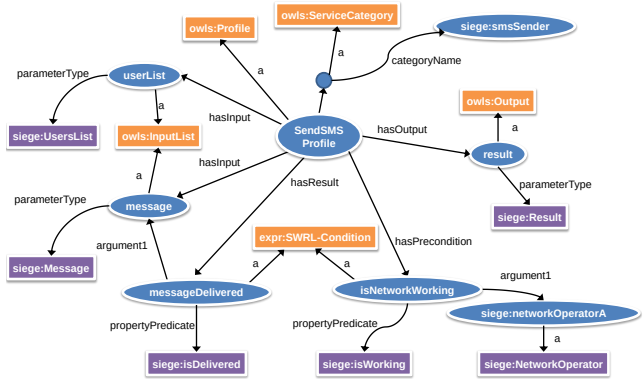


Figure 3.15: Ontological description of the service SendSMS

```
(:action SendSMS
     :parameters(?userList ?message ?result)
     :precondition (and(hasKnowledge ?userList)
                       (hasKnowledge ?message)
                       (isWorking OperatorANetwork))
     :effect (and (hasKnowledge ?result)
                       (isDelivered ?message))
)
```

Table 3.2: PDDL code generated by the PDDL Domain serializer

By using the backward strategy of the Graphplan algorithm, if the problem goal contains the effects of the *Send SMS* service, the PDDL4J planner may include the above PDDL action in the plan and try to reach the specified precondition through a single service, or a service chain, beginning with the specified initial state.

As the aim of the whole scenario is about alerting the population living in a specific area, the goal includes the following post-conditions: to get home phone numbers of all people living in the area and to alert them by phone calls, SMS, using TV channel, etc.

The resulting plan process includes several services and three different parallel branches. The three branches contain the following actions:

1. Alerting people by using landline phones:
    - get the list of home telephones in the area using a white pages service;
    - give such phone numbers to a call center for automatic call procedure;
    - get a list of citizens without home phone;
    - send the list of citizens that were not warned (as they do not have home phone or because they did not answered the call) to local police.

2. Broadcasting an alert message using a specific service made available by TV corporations.

3. Alerting people by using SMS channel:

   - use a service made available by the birth register to get the names of all people living in the area;
   - get mobile numbers (MSISDN) from telecom companies;
   - send SMS to all the retrieved mobile numbers.

Fig. 3.1, introduced at the beginning of this chapter, is the graphical representation of the solution (abstract) plan produced by the Graphplan component, which includes ten services (*WPGetUsers*, *GetHomePhoneUsers*, *GetNoHomePhoneUsers*, *SendToCallCenter*, *WPGetUsersNames*, *SendToPolice*, *EmergencyTVChannel*, *RegisterGetUsers*, *GetMsisdn*, *SendSMS*) to perform the activity described above.

In Fig. 3.16, an excerpt of the corresponding WS-BPEL process (concrete plan) automatically generated by the WS-BPEL serializer is shown.

The auto-generated code in Fig. 3.16 contains a main sequence composed of a receive activity, a flow, a variable assignment and a reply activity, coherently with the considerations about control and data flows generation reported in Section 3.4. The first receive activity (*receiveInput*) creates a new instance of the business process and stores the request message coming from the invoker in the *input* variable, declared in the <*variables*> section. The type of the variable is specified in the *.wsdl* artifacts file associated to the bpel file and includes the destination area and the message text, among other fields.

The final reply activity (*replyOutput*) returns a response message to the invoker, whose fields (in this case just the outcome of the alert) are properly prepared by the assign activity preceding it (*replyOutputPrepareActivity*).

The middle flow contains the <*invoke*> activities required to interact with external services, implementing the abstract tasks in Fig. 3.1. Coherently with the three-branches structure of the plan, the BPEL flow is composed of three sequences. One of them simply invokes a service for alerting people by means of a TV message (*EmergencyTVChannelService*). Another sequence contains the chain of service invocations for getting information about people living in the area (*RegisterGetUsersService*), retrieving their mobile numbers (*GetMsisdnService*) and sending SMS to them (*SendSMSService*). The last sequence of the flow is more complex, including an invocation of the *WPGetUsersService* followed by a flow (containing the sequence of *GetHomePhoneUsersService* and *SendToCallCenterService* in parallel with

```
<bpel:process ... name="alertingPopulationPlan" targetNamespace=
      "http://ing.unisannio.dslab.it/bpel/alertingPopulationPlan">
   ...
   <bpel:partnerLinks>...</bpel:partnerLinks>
   <bpel:variables>...</bpel:variables>
   <bpel:sequence name="mainSequence">
      <bpel:receive name="receiveInput" partnerLink="client"
            portType="tns:alertingPopulationPlanPortType"
            operation="alertingPopulationPlanOperation" variable="input" createInstance="yes" />
      <bpel:flow>
         <bpel:sequence>
            <bpel:assign name="WPGetUsersServicePrepareActivity" validate="no">...</bpel:assign>
            <bpel:invoke name="WPGetUsersService" partnerLink="WPGetUsersServicePL"
                  portType="ns1:WPGetUsersServicePT"
                  operation="WPGetUsers" inputVariable="WPGetUsersRequestVariable"
                  outputVariable="WPGetUsersResponseVariable" />
            <bpel:flow>
               <bpel:sequence>
                  <bpel:assign name="GetHomePhoneUsersServicePrepareActivity" validate="no">...
                  </bpel:assign>
                  <bpel:invoke name="GetHomePhoneUsersService" partnerLink=
                        "GetHomePhoneUsersServicePL" portType="ns2:GetHomePhoneUsersServicePT"
                        operation="GetHomePhoneUsers"
                        inputVariable="GetHomePhoneUsersRequestVariable"
                        outputVariable="GetHomePhoneUsersResponseVariable" />
                  <bpel:assign name="SendToCallCenterServicePrepareActivity" validate="no">...
                  </bpel:assign>
                  <bpel:invoke name="SendToCallCenterService" partnerLink="SendToCallCenterServicePL"
                        portType="ns3:SendToCallCenterServicePT" operation="SendToCallCenter"
                        inputVariable="SendToCallCenterRequestVariable"
                        outputVariable="SendToCallCenterResponseVariable" />
               </bpel:sequence>
               <bpel:sequence>
                  ...
                  <bpel:invoke name="GetNoHomePhoneUsersService" ... />
               </bpel:sequence>
            </bpel:flow>
            ...
            <bpel:invoke name="WPGetUsersNamesService" .../>
            ...
            <bpel:invoke name="SendToPoliceService" ..."/>
         </bpel:sequence>
         <bpel:sequence>
            ...
            <bpel:invoke name="RegisterGetUsersService" ... />
            ...
            <bpel:invoke name="GetMsisdnService" ... />
            ...
            <bpel:invoke name="SendSMSService" ... />
         </bpel:sequence>
         <bpel:sequence>
            <bpel:assign name="EmergencyTVChannelServicePrepareActivity" validate="no">...
            </bpel:assign>
            <bpel:invoke name="EmergencyTVChannelService" .../>
         </bpel:sequence>
      </bpel:flow>
      <bpel:assign name="replyOutputPrepareActivity" validate="no">...
      </bpel:assign>
      <bpel:reply name="replyOutput" partnerLink="client"
            portType="tns:alertingPopulationPlanPortType"
            operation="alertingPopulationPlanOperation" variable="output" />
   </bpel:sequence>
</bpel:process>
```

Figure 3.16: An (excerpt of) automatically generated WS-BPEL process for the alert workflow

GetNoHomePhoneUsersService) and a final invocation for the *SendToPoliceService*. It is worth to note that each <*invoke*> activity is preceded by an <*assign*> activity, aimed at preparing the input message of the service to invoke, as described in Section 3.4.

Some limitations in the auto-generated BPEL code of Fig. 3.16 derive from the assumptions described in Section 3.4. A synchronous request-response message exchange pattern is used for invoking external services from the BPEL process (each <*invoke*> has both the input and the output variables specified). The same synchronous model applies to the interactions between the BPEL business process and its invokers (the process starts with a receive and terminates with a reply activity). Correlation sets are not addressed in the current implementation of the tool and the generated flow is related to a normal execution flow. Finally, absence of structural mismatches among services with the same semantic input/output characterization is assumed, when generating the code by exploiting the binding information in the OWL-S groundings. These limitations are intended to be overcome as future work, for example by means of mediation services (for structural mismatches) and callback mechanisms (for also supporting asynchronous service interactions).

3.5.2 Performance Analysis

Execution times for solving the previously described "population alert" composition problem are reported in Fig. 3.17.

It is worth to note that architectural issues and flexibility of the proposed solution have been considered as the main drivers for tool implementation instead of performance or other non-functional requirements, which we address in Chapter 5, in the context of large-scale service domains, by means of a distributed and cooperative approach to service composition. In addition, the tool described in this chapter is still a prototypal implementation, which has been useful to demonstrate feasibility and utility of the proposed approach. Nevertheless, the performance measures acquired in our analysis have confirmed the potential of the tool, which can be effectively and efficiently used to support designers in service composition, when the service domain is not particularly large, and to implement adaptive re-planning related to sub-processes of autonomic workflows.

The "population alert" problem has been solved in ten test cases, each characterized by a different size of the service domain, i.e. a different number of OWL-S semantic descriptions available for solving the problem (from 10 to 100 descriptions, by tens). For each of the test cases considered, the ten services for composing a possible solution to the problem (see Section 3.5 and Fig. 3.1) were included in the domain and our tool was always able to find the correct composite solution. The BPEL generator has created, in all the test cases, a complete WS-BPEL process (.bpel file together with the other files required to deploy it), which was correctly executed on

the RiftSaw workflow engine.

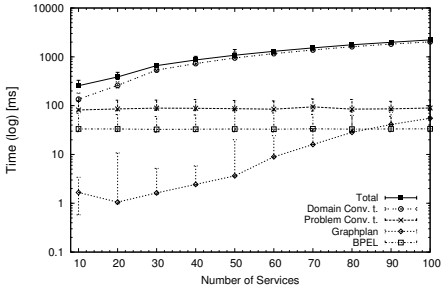
The total execution time is the sum of four main contributions:

1. *domain conversion time*;
2. *problem conversion time*;
3. *planning time*;
4. *WS-BPEL generation time.*

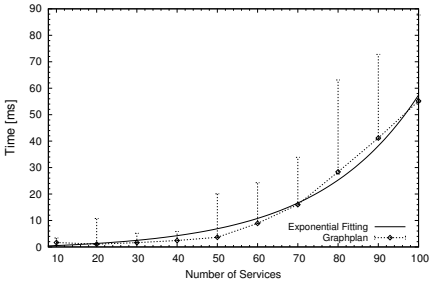
Domain conversion time represents the time needed to access, parse and convert to PDDL each OWL-S semantic service description included in the planning domain; *problem conversion time* is the time required to access, parse and convert to PDDL the OWL-S description of the problem to solve; *planning time* is the time required for Graphplan to find a solution to the PDDL problem in the PDDL domain; *WS-BPEL time* is the time required by the BPEL serializer to produce the set of files making up a complete and executable BPEL process from the abstract plan and the WSDL files referred in the OWL-S grounding section.

The four time contributions have been measured by using system time, with nanosecond resolution. The machine used for executing the test cases was an Intel Core 2 Duo CPU, with 3 GB RAM, running a Linux Debian distribution. Semantic descriptions and WSDL files were deployed locally to the machine on an Apache server.

The total execution time and the four contributions that compose it are reported, on a log scale, in Fig. 3.17a. The points in the figure are the median values obtained



(a) Total execution time and its breakdown



(b) Graphplan planning time, with exponential fitting

Figure 3.17: Time analysis for the population alert problem

after 100 iterations for each test case, while observed minimum and maximum values are depicted as error bars around the median value.

10 initial iterations per test were executed, whose times have not been accounted in order to evaluate tool performance during a steady-phase. This way, dynamic class loading, Java *Just-In-Time* (re-)compilations and other cold-start overheads due to the Java Virtual Machine only marginally affects the reported measured performance indexes.

As expected, the predominant contribution to the total execution time is *domain conversion time*, since it includes the access to a set of semantic descriptions and ontologies via an HTTP server (local to the testing machine) and their parsing by means of the OWL-S API (v. 2.0).

In relation to *problem conversion time* and *BPEL generation time*, we have observed an approximately constant time, since, in the case of OWL-S problem conversion, one description has to be converted into PDDL, while, in the BPEL case, the found solution plan to convert to BPEL is the same for all the test cases.

Finally, regarding planning time, Graphplan exhibits an approximately exponential trend with respect to the number of domain services, as shown in Fig. 3.17b. In other words, resolution times can be extremely high when planning is performed on a large-scale centralized registry. Therefore, as detailed in Chapter 5, such approaches can be feasible on small/medium-size service domains (in such cases, Graphplan presents very low time overhead to compute solutions, even lower than 100 ms), but become unpractical when a large or very-large service registry must be explored, demanding for more scalable and efficient approaches to service composition.

In Chapter 5, we report on our solutions for efficient distributed and collaborative discovery by composition, aimed at dealing with Internet-scale service registries.

Chapter 4

Context-awareness in Service Composition

A characterizing feature of SOA-based systems is their high dynamicity in selecting the functions that satisfy user requirements. Service composition plays a fundamental role in this kind of software systems.

In the previous chapter, we have reported on automatic generation of concrete service composition, starting from some results derived from the AI field and considering the classical scenario characterized by published services that are unable to adapt to the execution context.

However, the potential of SOA could be better exploited if such ability of building an application by composing (even on the fly) existing functions were augmented with the awareness of the surrounding context where the composition takes place. This way, services and related compositions could be forged to adapt their malleable aspects to the specificity and dynamic nature of the environment. This impacts the design phase of the services that characterize a given domain, but also the definition of the problem and the goal used to drive a specific composition.

Designing services with this new approach means to extend their semantic descriptions with new attributes and rules that are able to slightly change the structure and behavior of the services according to the needs emerging in the specific context where they will be used. This is a desirable property in SOA, where services, differently from components, should be implemented to be exploited in contexts that could change even during the same execution.

We describe in the following an application scenario, aimed at highlighting the

importance of context-awareness in Web service composition.

Bob likes TV-shows. He wants his home media server to automatically check the availability of new episodes every day, by connecting to the websites of the major broadcast companies (e.g. Fox, BBC, etc.) or authorized external media providers (e.g. YouTube, Google TV, etc.), to buy (if needed) and download them in order to watch the episodes when he likes. When download is not possible, since the provider only allows for streaming or disk space is limited, only minimal information about the show (e.g. title, air date, stream URI, etc.) is available for selection, while playing could be performed by connecting to the stream when required. For each retrieved TV show, if the language is different from Bob's native one, subtitles for that episode should be retrieved in that language and superimposed to the video while playing it.

Bob wants his media-player to prompt the list of available new TV-show episodes, allowing him to choose the one to play. He prefers subtitles be shown in a large font size. Moreover, the blinds and artificial lights in Bob's TV room should be automatically adjusted in order to guarantee the best light conditions for watching TV.

In this scenario, smart devices like media server, TV, light sensors, inter-connected by means of wireless connection, pervade the environment, and some of them are connected to the Internet. Moreover, a single service could not satisfy Bob's complex needs. In particular, a service is required to download TV-show episodes from some Web Site; another one for retrieving proper subtitles; other services are to be used to control the blinds in the room, check TV programs and so on. We assume that each smart device has a service interface, which is published in a registry, deployed in the pervasive environment. It is possible to achieve Bob's goals by composing together some of the services available in the registry.

Current context and user preferences have to be taken into account in order to fully implement the scenario above.

Some context elements regard the time dimension: every day new TV-show episodes are played on TV channels and, according to the fact they belong or not to Bob's favorite ones, they have or have not to be downloaded by the media server. Light condition changes during daytime, causing a strong or weak lighting of the room, which might require closing the blinds or switching on artificial lights to satisfy Bob preferences. Other context elements are related to the information associated with the user: if Bob is Italian, an Italian subtitle provider (or an additional service for translation) has to be found on the Web and subtitles have to be downloaded for each new episode whose spoken language is not Italian. Some other context elements regard download or reproduction rights for copyrighted content, which can change

according to the physical location of the user. Preferences are relevant in relation to subtitle language or font size while playing, the list of TV-shows, lighting levels, etc.

This chapter proposes an approach to design context-aware services by extending the OWL-S ontology with the context dimension. The services designed according to the proposed model can be discovered and composed by dynamically tailoring a (possibly distributed - see Chapter 5) service search space to user needs, preferences and the current situation of the environment where the services have to be executed. In particular, we present a model for context representation and its implementation in OWL, and an extension of OWL-S for allowing to use context-awareness expressions in semantic service descriptions and their adoption during composition.

The rest of the chapter is organized as follows. In Section 4.1, the conceptual model for context representation is described, while in Section 4.2, the OWL-Ctx ontology based on that model is detailed, together with the extension of the OWL-S ontology for service description. Section 4.3 describes our context-aware composition system and discusses it by means of a detailed application example.

4.1 Context Model

By the term *application state* we mean the set of variables and corresponding values that the application is able to access or modify. We distinguish between *internal* and *external application states*.

The *internal state* is the set of variables only visible to the application itself. They are created, used and eventually destroyed by the application (which can be an ensemble of components, having visibility of them) and are not accessible outside the application. Besides *input* and *output* parameters, the application may have predicates to be satisfied in order to execute it (*pre-conditions*) and predicates that are satisfied after the application has been executed (*post-conditions*).

The *external state* is the set of variables accessible also outside the application. They can be read or modified by users, devices or applications other than the one the state is referred to. This set of variables represents the ***context*** in our model: it includes every attribute that characterizes a user and/or the (smart) environment a distributed application interacts with.

An application may exhibit dependencies from the context and in this case we call it *Context-Aware Application* (CAA). In our vision, context dependencies are related to application pre- and post- conditions. A CAA may or may not change the current context during and after its execution. In the latter case, post-conditions

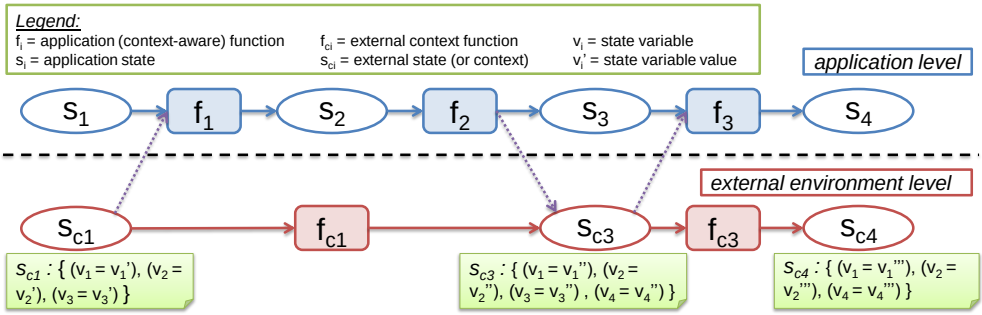


Figure 4.1: Contexts and Context-Aware Applications

include references to context variables. When the properties above are related to Web services, we speak about *Context-aware Web Services* (CAWS). Fig. 4.1 graphically represents our notions of context and CAA.

In the figure, two levels can be distinguished: the *application level* and the *external environment* or *context level*. At the first level, an application is represented as a set of software functions (f_i), or functional components (e.g. Web services), interconnected to implement the complex logic of the distributed application. At the second level, context may evolve by means of both external, usually manual, world activities (f_{ci}) and context-aware software components (f_i) from the application level.

The states belonging to the application level (s_i) are internal states, only relevant and visible to the specific applicative function f_i, connected to the states by means of input or output transitions. In general, a function f_i has the state s_i as its pre-condition and s_{i+1} as its post-condition. The states belonging to the external environment level (s_{ci}) are contexts. Both the application level and the context level functions may have context states s_{ci} as pre- or post-conditions, i.e. they can: (a) require a particular configuration of the external environment to be satisfied (external pre-conditions); (b) change the external environment by means of a software or manual action (external post-conditions). User preferences are attributes characterizing a specific user behavior and, in this sense, they are part of the user context.

In the figure, both f_2 and f_{c1} contribute to create the context s_{c3}, the post-condition associated to the two functions and the pre-condition required to execute f_3 and f_{c3}.

It is worth to note that each external state (i.e. a context) provides a partial view of the external world, describing only the external variables relevant with respect to a human actor or a software application. A complete description of the environment

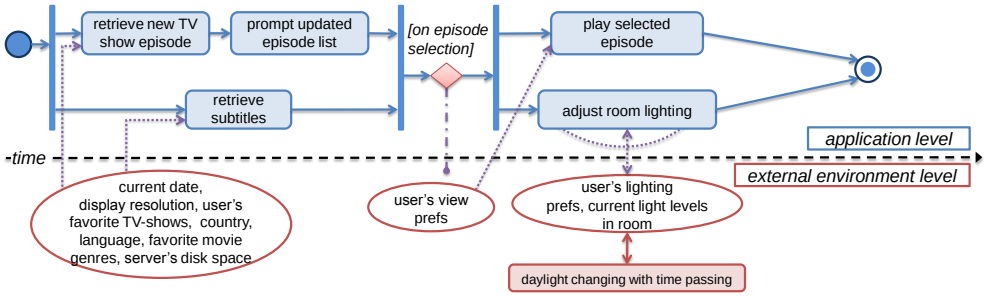


Figure 4.2: UML activity diagram for the pervasive environment scenario

would require a theoretically infinite set of variables, but, typically, only a small subset of it would be actually considered by an application.

Every context-aware function (both software and manual) can be seen as an operator that changes the values of some context variables in the input state (pre-conditions) with the updated values included in the output state (post-conditions). During this state update, some initially unknown or irrelevant variables may be assigned a value, appearing consequently in the output state (e.g. v_4 in s_{c3} in Fig. 4.1).

To clarify the context model, in Fig. 4.2, we apply it to the pervasive environment scenario, introduced at the beginning of this chapter, by using a UML activity diagram.

The services required to accomplish the user's goals are reported in the upper level of the figure, together with the control logic required to orchestrate the composite process. In the lower part of the figure, a few context states are shown, together with the interactions with the application level. Some other context states (e.g. availability of new episodes or subtitles, episode successfully played at the end of the service execution, etc.) and the internal states of the application level (e.g. TV-show database updated) are not shown in the figure for the sake of readability. Contextual events, which enact some of the activities in the scenario (e.g. *on episode selection*), are shown in the top part of the figure, i.e. the application level.

The figure shows some context elements, e.g. current date, country and user's language, that are essential to decide which episodes and subtitles to download, and the current light conditions in the room in order to adjust lighting according to the user's preferences.

4.2 Context-aware Semantic Service Design

To support design and composition of context-aware services we propose: an OWL ontology (OWL-Ctx), supporting the description of sets of contexts in specific domains (Subsection 4.2.1) and an extension of the OWL-S ontology for services (OWL-SC, Subsection 4.2.2), allowing for the specification of OWL-Ctx-based context adaptation rules.

Context-aware semantic descriptions can be exploited during the service composition process to automatically generate compositions better-tuned to the requestor's behaviors and preferences and to the particular situations of the surrounding environment.

4.2.1 OWL-Ctx: an OWL Ontology for Context Modeling

Fig. 4.3 shows OWL-Ctx, an extensible OWL ontology for describing contexts according to our model, reported in Section 4.1.

Context, *ContextDimension*, *ContextDimensionValue* and *ContextDimensionConstraint* are the main concepts in this ontology (Fig. 4.3a).

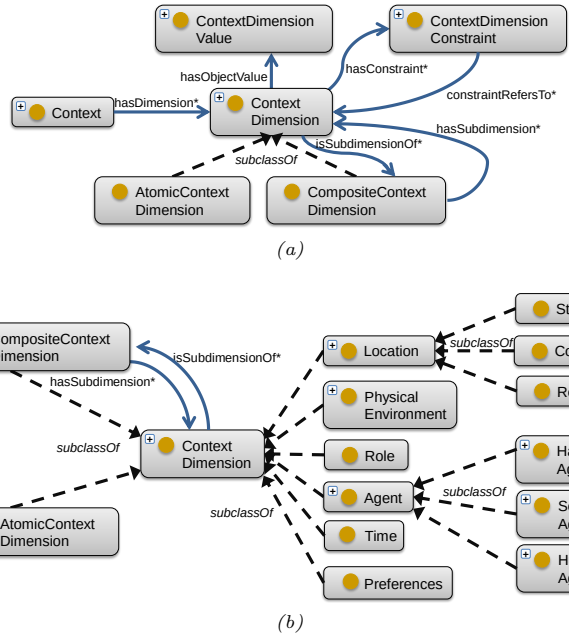


Figure 4.3: OWL-Ctx: ontology language (a) and (partial) middle ontology (b)

The *Context* related to a software system is composed of a set of *ContextDimensions*, each describing one relevant aspect, or dimension, of the environment enclosing the particular software system. Each dimension can be modeled at a different level of abstraction, according to the specific requirements related to the application domain to consider. In this sense, we can distinguish a *ContextDimension* as a composite or an atomic dimension. Differently from an *AtomicContextDimension*, a *CompositeContextDimension* can be further decomposed in one or more sub-dimensions that can be helpful for a better characterization of the associated context aspect. For example, a *HardwareDevice* concept may be sufficient in some domain for modeling the context aspects related to the device used for executing the software. However, in another domain, a finer-grained model, including *Display*, *Processor*, *Disk*, etc. as its sub-dimensions, could be useful for better modeling the context. In this case, an *AtomicContextDimension* for each of the above elements will be introduced and the range of the *hasSubdimension* properties for the *HardwareDevice* concept will be restricted to them.

When modeling contexts in a specific application domain, it is possible to specify the dimensions deriving from the *AtomicContextDimension*, those specializing the *CompositeContextDimension* concept and the relationships among them by means of multiple *hasSubdimension* properties. For each *ContextDimension*, a *ContextDimensionValue* can be defined to represent complex values associated to the specific dimension. However, if a concrete XML-Schema built-in data type (e.g. *string*, *int*) is sufficient for representing the values for a specific context dimension, the data type property *hasValue* associated to the *ContextDimension* (not shown in the figure) can be used, by defining a sub-property having the required xsd type as range. A *ContextDimensionConstraint* can be used to specify domain-dependent constraints applying to *ContextDimensions*.

In the partial context middle ontology of Fig. 4.3b, *Time*, *Agent*, *Location*, *Preferences*, *Role* and *PhysicalEnvironment* describe some domain-independent context dimensions, typically used for specifying context properties with respect to any software application. It is worth to note that the context dimensions shown in Fig. 4.3b are considered without any assumption about the relationships between them. This information depends on the application domain and should be defined by the designer.

Considering the scenario illustrated at the beginning of this chapter, a designer could specify the properties of possible contexts in a Media ontology, as shown in Listing 4.1.

MediaContext is a specialization of the *OWL-Ctx: Context* concept, with a restric-

tion on the range of the *hasDimension* property to values of *Agent*, *MediaContent* and *PhysicalEnvironment* classes. *MediaContent* is a composite context dimension introduced by the Media ontology to characterize any media resource like pictures, videos, etc.

Listing 4.1: Media Context OWL Ontology (excerpt)

```
<owl:Class rdf:about="&Media;MediaContext">
  <rdfs:subClassOf rdf:resource="&OWL-Ctx;Context"/> ...
  <owl:equivalentClass><owl:Class><owl:intersectionOf rdf:parseType="Collection">
    <rdf:Description rdf:about="&OWL-Ctx;Context"/>
    <owl:Restriction><owl:onProperty rdf:resource="&OWL-Ctx;hasDimension"/>
    <owl:someValuesFrom><owl:Class><owl:unionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="&OWL-Ctx;Agent"/>
      <rdf:Description rdf:about="&Media;MediaContent"/>
      <rdf:Description rdf:about="&OWL-Ctx;PhysicalEnvironment"/> ...
```

The reported context specification has been considered for modeling contexts in the TV-shows scenario, where the user (Bob), the devices (Bob's TV, Bob's PC), the media content (TV-shows) and the location (TV-Room) represent the relevant context dimensions.

4.2.2 OWL-SC: an OWL-S Extension for Describing Semantic Context-aware Services

The OWL-Ctx ontology is exploited by the OWL-SC ontology, our extension of the OWL-S service ontology for describing context-aware semantic services. The most relevant elements extending OWL-S are reported in Fig. 4.4a. Each of the three core concepts, provided by the OWL-S ontology to describe a *Service* (*Profile*, *Process* and *Grounding*), can be associated (Fig. 4.4b) to a *ContextAdaptationRule*, by means of the *hasContextAdaptationRule* object property, specialized by three sub-properties.

A service designer may be aware of a set of contexts or context dimension values, which can be used, during the design phase, to specify conditions for the service to change its basic features (i.e. profile, process or grounding properties). A reference to the current context may be used to relate context conditions to the situation at the moment the extended service description will be analyzed (relationship *executesIn* between *Service* and *Context*) for discovery or composition.

The *Ctx-Condition* concept is a sub-concept of *Condition* from the *Expression* ontology used in the original OWL-S. *Ctx-Condition* contains references to *Context* and *ContextDimension* since conditions on the current context may be specified in relation to other known (at design time) contexts or dimensions.

A context adaptation rule allows for specifying a semantic service as a context-aware service. Specifically, context-awareness can be injected at each level of the

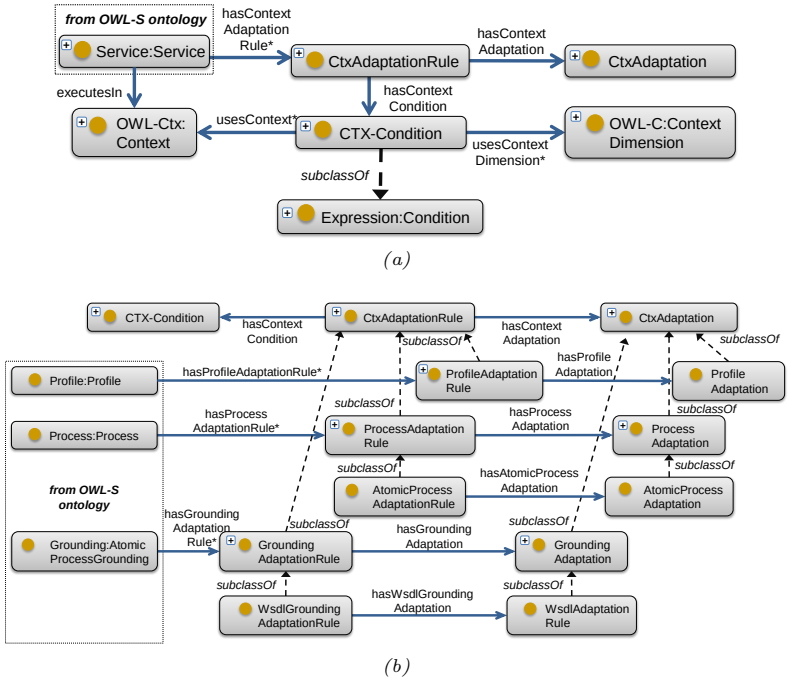


Figure 4.4: OWL-SC: ontology language (a) and (partial) middle ontology (b)

OWL-S ontology: at the profile level, it is possible to change the semantic description of the service IOPE interface (i.e. the service profile) according to some context condition specified in a *ProfileAdaptationRule*; at the process level, the process may be context-aware as specified by a *ProcessAdaptationRule*; finally, the grounding description may depend on the context as specified by a *GroundingAdaptationRule*.

A *ContextAdaptationRule* (and its sub-concepts) is defined by means of a context condition (*Ctx-Condition*) and a context adaptation action (*ContextAdaptation*). It prescribes the context-dependent condition to be satisfied in order to apply the associated adaptation for the specific section of the OWL-S description. If at least one *ProfileAdaptationRule*, *ProcessAdaptationRule* or *Groun-dingAdaptationRule* is specified in the description, the service is a CAWS. The *ContextAdaptation* concept is introduced to specify a context-aware adaptation for the specific section of the OWL-S service description and is, for this reason, specified by the sub-concepts *ProfileAdaptation*, *ProcessAdaptation* and *GroundingAdaptation*.

The most relevant context adaptations for the OWL-S sections provided by our language are described in the following.

For the profile/process section, on some context condition, it is possible to:

- Defaulting an input/output parameter (profile/process section):
  `when ctx_condition id.value = default_value`
- Nulling (i.e. removing) a parameter, not applicable for a specific context condition (profile/process):
  `when ctx_condition param_id = null_value`
- Changing the owls <process:parameterType> of an input/output parameter to a different ontology concept (process):
  `when ctx_condition param_id.parameterType = ontologyConcept`
- Replacing pre-conditions or effects of the basic OWL-S service description (profile/process):
  `when ctx_condition replace prec_id with new_precondition_id`

Regarding the grounding section, our language allows for:

- Changing the *WsdlAtomicProcessGrounding* input-output section of an atomic Process with a new `WsdlMessageMap`;
- Changing the *WsdlAtomicProcessGrounding* section of an atomic Process with a new WSDL operation and/or WSDL portType or interface;
- Replacing the entire *WsdlAtomicProcessGrounding* of an atomic Process with a different one.

Context-dependent conditions, currently supported by our OWL-SC ontology, include:

- `current_ctx matches ref_ctx`
- `current_ctx includes "concept hasValue ontology_individual"`
- `current_ctx includes "concept.datatype_property OP value"`
- `current_ctx includes "concept.object_property hasValue individual"`

where `current_ctx` is the context reference for the context at the moment in which the composition domain is explored for finding a solution to the submitted problem. Both `current_ctx` and `ref_ctx` specifications can be verified according to an approach like the one used in [15]. *OP* is a relational operator (e.g. $==$, $\leq$, etc.) or any other binary operator compatible with the data type property.

Currently, we are using SWRL to express the last three kinds of condition. An example of SWRL context condition, related to the *Media* context specialization previously defined, is the following:

```
OWL-Ctx:hasDimension(current_ctx, ?hwAgent) ∧ Media:HWAgent(?hwAgent) ∧
OWL-Ctx:hasSubdimension(?hwAgent, ?role) ∧ OWL-Ctx:Role(?role) ∧ sameAs(?role,
Media:DownloadServer) ∧ OWL-Ctx:hasSubdimension(?hwAgent, ?disk) ∧ Media:Disk(?disk) ∧
Media:diskMBSpace(?disk, ?avSpace) ∧ swrlb:lowerThanOrEqual(?avSpace, 500)
```

The condition verifies whether one disk of a media server has not enough space ($\leq$ 500MB) to download some file.

As an example of context adaptation based on the reported condition, if the available space is limited, the service for retrieving TV-show episodes does not download files to the local PC, but only retrieves minimal information (e.g. URI) for streaming.

4.3 Context-aware Service Composition

The composition tool described in Chapter 3 has been extended in order to support context-aware semantic descriptions of services and problems.

Fig. 4.5 describes the logic flow and the main architectural components of the extended Composer.

When performing context-aware composition, the set of semantically described candidate services may include CAWSs (the OWL-SC domain) and is provided as an input to the system (manually or automatically via a matchmaking process), together with the semantic description of the initial state of the environment and the goal, which may include context adaptation rules, in turn. OWL input files are analyzed and used by the *OWL Analyzer* to build the internal representation of the composition problem.

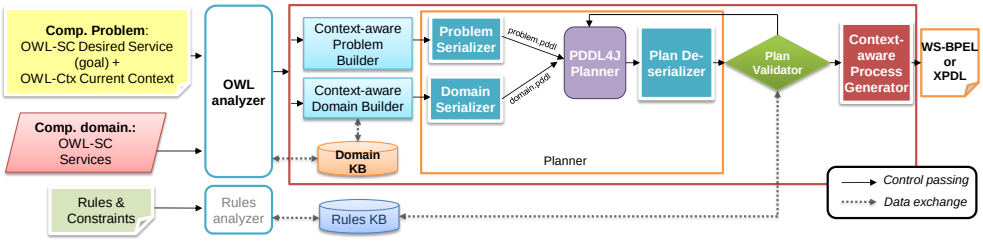


Figure 4.5: Composer Architecture

Then, context is exploited for preparing the planning phase. The *Context-aware Domain Builder* evaluates the context dependencies in pre-conditions, effects and adaptation rules of the service descriptions, with respect to the current context, and generates a contextualized instance of the domain, suited to be converted into the planner language (PDDL [66]).

Similarly, the *Context-aware Problem Builder* augments the problem representation given by the user by injecting into it relevant information from the context, which is derived by the monitoring support available in the SAWE engine (see Figure 3.4), and evaluates the context rules related to the profile/process sections of the specified problem.

It is worth to note that, in general, the context can be extremely dynamic, evolving so rapidly that it could be not possible to complete the composition process without taking into account the mutated environmental conditions. However, in this thesis, we assume context variations to be reasonably slower than the time required for a typical composition problem to complete (see Subsection 3.5.2 for details about performances of the composer).

By this assumption, the composition process may be completed in relation to the contextualized domain and problem generated from the context information available at the beginning of the process, being the final composition still meaningful.

Moreover, as reported in Chapter 3, in the current SAWE implementation, context changes, following the composition process, may be addressed by re-planning, according to a reactive approach to adaptation (the *Plan1* transition in Fig. 3.3). Alternatively, a proactive strategy (i.e. the *Plan2* transition in the post-mortem analysis state of Fig. 3.3), based on probabilistic techniques (e.g. Bayesian networks, Markov processes), might be used for learning context transition probabilities by analyzing the history of context changes and the consequent reactions. This way, alternative compositions could be computed in advance for highly probable context transitions, thus minimizing (with respect to reactive strategies) the time the running system is unavailable when context actually changes.

The PDDL4J *Planner*, described in Section 3.3, operates therefore on contextualized domain and problem inputs in order to produce the plans of domain actions satisfying the problem.

The *Context-aware Process Generator*, in charge of serializing a concrete (WS-BPEL) and executable representation of the plan, is an extended implementation of the plan serializer described in Section 3.3). It is able to contextualize the available grounding information by evaluating the context-aware adaptation rules related to

the grounding section of the OWL-SC files, thus producing a contextualized business process description.

4.3.1 Pervasive Scenario Example

In this section, a simplified version of the application scenario described at the beginning of this chapter is detailed, both in relation to the design of domain-specific OWL-Ctx ontologies and context-aware semantic Web services, and regarding context-aware compositions to generate executable business processes.

Fig. 4.6 shows the inputs for the context-aware composition process (i.e. problem and domain description by means of predicates over OWL-Ctx and its domain-specific extension, and OWL-SC annotated services). In Fig. 4.6a, a user requests a service for retrieving and playing the last episode related to some specified TV-show; subtitles are requested to be superimposed to the video. Besides the desired service IOPE description, our composer is fed with the current context representation as input. This information can be automatically acquired and updated by the system, without being controlled or explicitly known to the service requestor, and is represented according to the OWL-Ctx ontology language. In Fig. 4.6b, a subset of the OWL-SC service domain (Media domain) is shown together with the relevant dependencies on the context, used in the adaptation rules.

The OWL-SC service for retrieving a TV-Show episode provides the effect *newEpisodeAvailable(TV-ShowEpisode)* and, in the basic case, the download of the last *TV-Show* episode file (effect *downloaded(TV-ShowEpisode)*).

The profile and its service atomic process have been extended with a context adap-

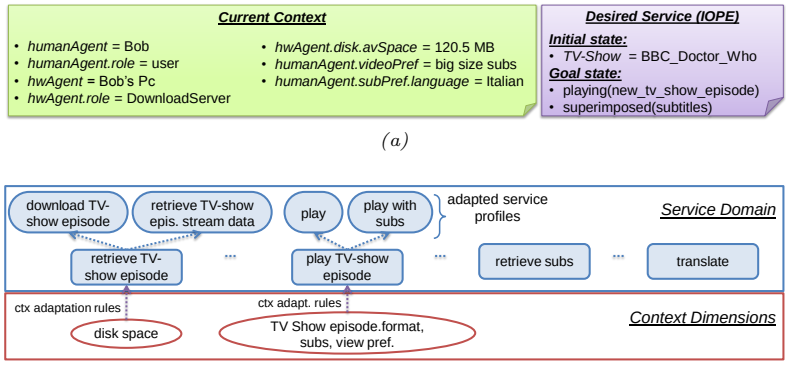


Figure 4.6: Problem description: current context, desired service IOPE (a) and domain (b)

tation rule, changing the process result in case the download server has not enough space to store the episode file. The context condition used to control this profile/process adaptation in the OWL-SC description is the SWRL condition reported at the end of Section 4.2.2. The adaptation replaces the downloaded result with the effect *streamDataAcquired(TV-ShowEpisode)*. The same condition also controls a grounding adaptation rule, changing the concrete service from the one used to download the file to the one for retrieving the stream information.

Since the disk space amounts to 120.5 MB in the current context of Fig. 4.6a, the rule is activated and the adaptation is applied by the *Context-aware Domain Builder*. The stream retrieval service (i.e. *RetrieveTV-ShowEpisodeStreamData* in Fig. 4.4) is included within the contextualized domain, while the download one is not.

When converting to PDDL, the stream retrieval service is the only one to appear as a PDDL action (i.e. the download tv-show episode action is excluded), thus reducing the actual size of the domain, with benefits over the composer performances.

The OWL-SC description of the *PlayTV-ShowEpisode* service includes the precondition *newEpisodeAvailable(TV-ShowEpisode)* and the input parameter *TV-ShowEpisode*. The effect *playing(TV-ShowEpisode)* is provided.

A context-dependent adaptation is included in its grounding section: depending on the space availability condition, a different concrete service will be grounded to the service process. The grounding adaptation rule does not generate different services in the domain. Instead, two groundings are stored for being later used by the Process Generator when the final WS-BPEL process has to be created. In other words, multiple WSDL descriptions can be associated to the same semantic service. The different WSDLs represent alternative concretizations of the semantic service, depending on the specified context conditions.

The profile and atomic process sections of the *PlayTV-ShowEpisode* service contains an adaptation rule for subtitle rendering. Depending on the availability of subtitles, they have to be considered as an additional input and their superimposition as an additional effect. Since at the moment of domain building it is not known whether the subtitles are going to be available or not, the Context-aware Domain Builder includes both the two services to the domain (the one including subtitles as input and the one not including them).

Moreover, since view preferences are another input of the service and part of the context, the Context-aware Problem Builder expands the set of known inputs for problem solution with them.

Regarding the *RetrieveSubtitles* service, we assume this service only downloads

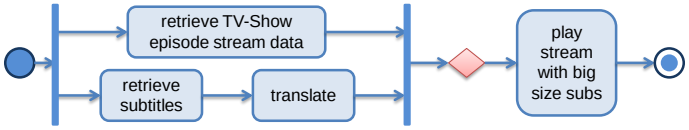


Figure 4.7: Resulting concrete service composition

English subtitles. This translates into the presence of the effect *language(subtitles, English)* in the service description. Such a service cannot satisfy the user's preferences included in the current context, i.e. *language(subtitles, Italian)*. This information is actually considered by the Context-aware Problem Builder to expand the goal, even though the user has not explicitly included it in the provided goal description. A *Translate* service to the user's language (i.e. Italian) is also available in the domain.

After contextualized domain and problem conversion to PDDL, the Graphplan planner generates an abstract PDDL plan. The plan considered in the example contains the sequence of *RetrieveSubtitles* and *Translate* in parallel with *RetrieveTV-ShowEpisodeStreamData*. The parallel is in sequence with the *PlayWithSubs* service. The translate service is included in the first sequence because the user's preference about subtitle language (Italian) has been added to the problem goal by exploiting available context knowledge.

Without context-awareness, the composer would have not been able to find such a suitable composition, according to the assumption that *RetrieveSubtitles* only retrieves English subtitles and the user has not explicitly indicated the Italian language preference in its goal.

Starting from the plan, the WS-BPEL generator produces the resulting concrete process of Fig. 4.7, by exploiting the available grounding information and knowledge about the current context.

Chapter 5

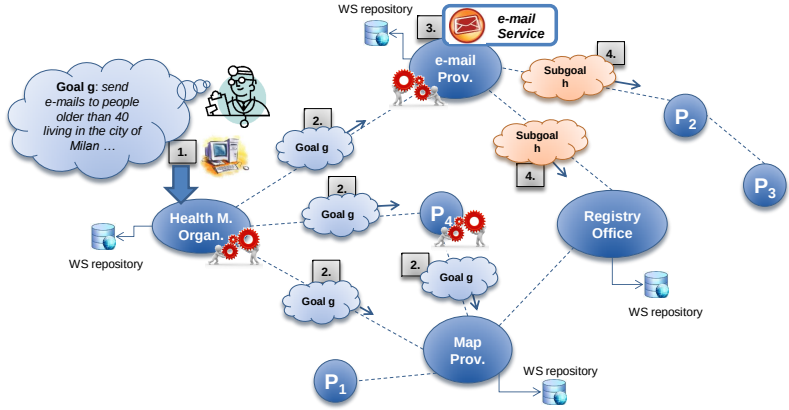
Scalable Composition with P2P Architecture

Discovery and composition in a large space of services are time consuming activities. It is unrealistic to handle such a space through a single public registry. As for DNS, distributed approaches based on a hierarchical architecture could represent a more efficient solution. However, service discovery may become a very complex process compared to DNS resolution because services can be semantically discovered and composed to create solutions for complex goals. Hierarchical architectures may be therefore inadequate to support semantic service composition.

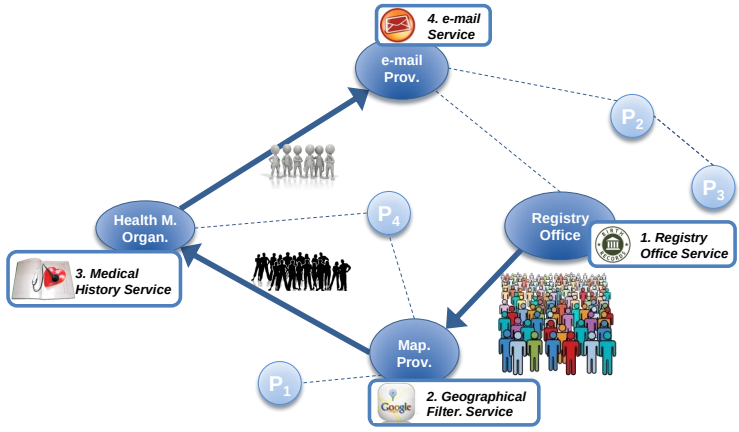
In addition, when dealing with semantically described services, it is difficult to properly define hash functions which are able to extract IDs from semantically annotated predicates, included in the IOPE service descriptions, in order to select the registries where to publish new services. Thus, structured architectures, which are based on the use of hash functions, cannot ensure both high functional and non-functional scalability for semantic service composition.

In this chapter, we address the problem of discovery and composition of services hosted by networked peers, according to a decentralized and cooperative approach, based on unstructured P2P overlays with superpeers. In particular, we expect that every peer issuing a query should be able to retrieve all the solutions (atomic and composite) available in the network.

As an example (see Fig. 5.1), a health maintenance organization could need to inform people suffering from some illness or at-risk patients to go to a hospital for a check-up within a specified date. The check-up notification has to be delivered to



(a) Service allocation to the peers and goal/sub-goals distribution



(b) The resulting composite service and its distributed execution

Figure 5.1: Collaborative composition example: graph of networked organizations (peers) with their services and known communication links (dashed lines). Gear symbols point out the peers involved in the composition stages (a)

people living in a specific geographical area, within the range of the health organization, and with some specific age requirement. A possible query, expressed in natural language, is the following: *alerting people living within a range of 5 kilometers, older than 40 years or suffering from heart diseases, to contact the nearest hospital and make an appointment for a cardiovascular check-up.*

In the following, we assume that Web services wrap some well-known Internet services, such as e-mail delivery (in the same way Web applications often wrap such systems). This way, inside an organization, like a hospital, it could be easier preparing

a message for a complex delivery instead of manually using different systems.

The query in the example can be solved by a service composition including (Fig. 5.1b): 1) a service for sending e-mail, hosted by different providers; 2) a service for handling people registry, hosted by a specific provider; 3) a service for filtering a list of people according to geographical information, like a geospatially characterized area in a specified city, hosted by different geographical service providers; 4) a medical history service to retrieve health information about people, hosted by a specific Health Maintenance Organization provider.

Both functional and non-functional requirements are semantically described as predicates in pre-conditions or post-conditions of the searched service. By exploiting information contained in the query, each peer is able to discard or accept the received message, based on the semantic service descriptions available in a local registry. For example, if no constraint is expressed in the query with reference to the e-mail service, several providers can satisfy the requirements and consequently one of them can be selected to send the notifications. On the other hand, the registry service to use for extracting the list of 40-year-old people must be the one that serves a specific geographic area. Moreover, queries may include information about the QoS to select specific providers according to non-functional requirements [24, 38]: for example, regarding the e-mail service, authentication mechanisms or cryptographic protocols, providing communication security over the Internet (e.g. SSL, TLS), could be explicitly requested by the user or implicitly deduced from the context where he/she operates, as described in Chapter 4. In either cases, the QoS requirements will be included as predicates in the query, together with the associated inputs (e.g. encrypted user credentials), to reduce the set of service peers to the ones satisfying the goal.

The remaining part of the chapter is organized as follows. In Section 5.1, we describe the main definitions and hypotheses adopted to deal with service composition in P2P networks. Section 5.2 details the concept of distributed and cooperative composition and the algorithms used for P2P discovery and composition. Section 5.3 presents the solutions used to create self-evolving Composition Overlay Networks (CONs), based on the use of a P2P with superpeers architecture. Section 5.4 describes the probabilistic strategy adopted for avoiding flooding in unstructured P2P networks when forwarding service requests. Section 5.5 reports on a comparison among CONs, Semantic Overlay Networks (SONs) and Distributed Hash Tables (DHTs). In Section 5.6, simulations and results are discussed; Section 5.7 presents our bidirectional search strategy, together with its performance evaluation in the simulated environment. Finally, in Section 5.8, we compare our dynamic forwarding strategy to other

gossip-based algorithms, typically used for implementing message propagation in P2P networks.

5.1 Definitions and Hypotheses

To deal with composition in P2P networks, we have defined a new fully distributed and cooperative planning technique, by considering the deterministic transitional model, the Closed World Assumption (CWA) and the offline planning hypothesis introduced in Section 3.2. Therefore, we assume to work with temporary knowledge bases when checking, via a matchmaking process, whether candidate services match the requested goal (containing the specification of the initial state and the desired final one) or only the final state (according to a backward planning approach). In the latter case, further chaining services are needed to satisfy the service pre-conditions. This approach may require the evaluation of different composition paths (without changing the real state of the system). Each path is monotonic (if executed) and refers to a potential evolution of the state. Since services are not actually executed during planning (CWA approach), at each stage from the desired final state towards the initial state, the planner only simulates the assertion of the effects associated to candidate services to infer whether the current pre-conditions are satisfied.

By following the same approach described in Section 3.2, a service and similarly a service goal are described through OWL-S profiles whose pre-conditions and effects are expressed by using SWRL conditions referring to concepts of OWL ontologies. An organization can semantically annotate its WSDL services by using any available ontology. However, in order to make its services easily discoverable by other peers in the network, it is reasonable for the organization to refer to concepts belonging to standard, shared and well-known ontologies, which have a higher probability to be queried and discovered by the peers.

By using the proposed model, discovery by composition in P2P networks is equivalent to perform a distributed state space search aimed at finding ordered collections of service operations (i.e. *composite solutions*) that solve the requested goal. Our backward strategy is described in Section 5.2, while a bidirectional alternative approach is reported in Section 5.7.

Since in this chapter we are mainly interested in the distributed aspects of composition, pre- and post-conditions will be represented, for the sake of simplicity, with tokens that abstract sets of concepts belonging to shared ontologies[1] (i.e. token ab-

[1] Unshared ontologies could be used in turn, by implementing ontology mapping techniques

straction). However, all the previous assumptions remain valid.

As will be detailed in Sections 5.2 and 5.3, we adopt an unstructured P2P with superpeers architecture to implement service discovery and composition according to a cooperative and fully decentralized approach. Unstructured P2P overlays allow for flexibility, fault tolerance and semantic matching capabilities in service discovery (see Subsection 2.3.2), while superpeers dynamically emerge in the network and can be exploited to manage knowledge, related to already retrieved compositions, that is progressively acquired by the P2P system and can be re-used for solving similar subsequent requests with increased efficiency.

5.2 Architecture and Protocols

Each peer of the network is involved in the concurrent execution of the following main protocols: 1) semantic service publishing in a local repository; 2) discovery of services satisfying user goals, allowing for collaborative and distributed composition; 3) query forwarding for efficiently propagating service requests in the network; 4) network reorganization for building semantic service overlays (CONs) from already solved service requests; 5) gossiping, as part of the forwarding strategy, for disseminating knowledge about the network structure among the peers; 6) consistency checking, to manage the CONs at both peer and superpeer levels.

These protocols are mapped to the component-based architecture reported in Fig. 5.2 together with some relevant methods for each component. Specifically, the planning logic component is responsible for the query solving process, by means of the backward or bidirectional solve strategies described in the following and in Section 5.7, respectively. The semantic matching component offers reasoning support for matching the service queries to the semantic service descriptions (i.e. the OWL-S

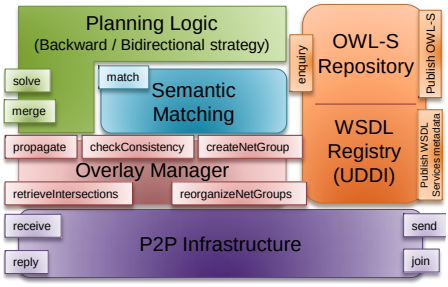


Figure 5.2: Component-based architecture of a peer

files) available in the service repository. The overlay manager is in charge of supporting overlay network formation and reorganization and provides the basic primitives for efficient query forwarding through the overlays. The P2P infrastructure (e.g. JXTA-based, or PeerSim-simulated) provides basic functionalities to join the P2P network and communicate with other peers. The service repository hosts OWL-S service descriptions, specifying the semantic characterization of the services published on the specific peer, and is an extended service registry offering support for semantic matchmaking. The WSDL service registry is a UDDI registry containing information about concrete services known to the peer. This information is exploited when editing the OWL-S service descriptions in order to properly annotate the concrete services and populating the OWL-S grounding section. Both the registry and the repository provide an inquiry and a publishing API.

In the following, we assume that all the modules reported in Fig. 5.2 are deployed on every node of the P2P network. This means that each single peer can: 1) locally publish service descriptions (i.e. within the local OWL-S repository) to semantically annotate known WSDL services, referred in the OWL-S grounding section, and make them discoverable and composable in the P2P network; 2) locally publish concrete WSDL services (i.e. within the local UDDI registry) to register known operations and use them in the OWL-S grounding section.

However, other configurations are possible. For example, some peers of the network may be deployed without the repository/registry module, by replacing it with a proxy to another peer's repository/registry module. In other words, within an organization some peers may share the registry/repository, by performing both publishing and inquiring in the shared service registry/repository.

The peer's publishing protocol is outside the scope of this thesis and will not be detailed in the following. The discovery protocol is described in this section, network reorganization in Section 5.3, forwarding and gossiping in Section 5.4.

The protocol executed by each peer (and superpeer) to perform P2P service discovery or composition is organized in two threads (active and passive) and described in pseudo-code in the following listings (5.1 and 5.3). The spawn keyword indicates the asynchronous invocation of the associated procedure (i.e. a new thread is spawned for the procedure).

The *active thread* of the discovery protocol (Listing 5.1) implements the application component of the discovery system and is active with respect to the query resolution process. This thread generates a new query object from the user specified goal (line 2), by setting information like the query's Time To Live (TTL), a query

identifier, the query goal and the source peer identifier. The query is then forwarded to other peers according to the `forward` procedure (line 3).

Listing 5.1: Discovery Protocol, active thread

```
1 discoveryProtocol(Goal g) [active thread]
2   Query q := new Query(g);
3   spawn forward(q);
4   spawn solve(q);
5   spawn reorganizeNetwork(g);
6   spawn collectSolutions();
```

After the query has been forwarded, the solving procedure (Listing 5.2) is started for the newly generated query (line 4).

The active thread spawns the `reorganizeNetwork` procedure (line 5) to start the network reorganization phase. This concurrent thread waits for query solutions coming from the network before starting the actual network reorganization according to the inferred knowledge (see Section 5.3 for details).

The `collectSolutions` procedure (spawned at line 6) aims at collecting the complete solutions that are locally notified by the passive thread (see the `locally notify` primitive in Listing 5.3) executed on the same peer. The current implementation (not reported due to its simplicity) cyclically waits for notifications and prompts the received complete solutions on the user interface.

The `solve` algorithm, spawned at line 4 in Listing 5.1, is reported in Listing 5.2.

Listing 5.2: Query solving procedure for simple peers

```
1  solve(Query q)
2    localSols := localSearch(q);
3    if localSols is not empty
4      reply (q, localSols) to q.source;
5    else
6      localPartialSols := localBackwardSearch(q);
7      add localPartialSols to partialSolsTable[q];
8      foreach partialSol in localPartialSols
9        partialGoalQuery:= new Query(q.goal, partialSol);
10       spawn forward(partialGoalQuery);
11       spawn solve(partialGoalQuery);
```

It is the implementation for simple peers of the search procedure, while superpeers perform a slight variant, described in Subsection 5.4.1.

The procedure `localSearch` in line 2 of Listing 5.2, is based on a semantic matching process (not shown in the following because outside the scope of this thesis) and explores the local peer repository, containing semantic service OWL-S descriptions, in order to find complete solutions to the requested goal. The matchmaking algorithms, used in the search phase, are extensively described in previous work of our research group [38, 96], but other algorithms or tools could be used [3, 56].

When no complete solution is available in the peer repository, *partial solutions* to the goal query are searched, by using a backward strategy (line 6). The `localBackwardSearch` is essentially a variant of the `localSearch` one: a partial solution to the submitted goal `g` matches its post-conditions, while having different pre-conditions. The same matchmaking algorithms are used to decide the presence of a match on post-conditions only.

Whenever a partial solution is found, a new query object is created (`new Query( ...)` in line 9) from the original goal (i.e. `q.goal`) and the retrieved partial solution (i.e. `partialSol`). The new query (i.e. `partialGoalQuery`) will contain the goal ranging from the pre-conditions in the original goal to the pre-conditions of the found partial solution. We call the query generated from a partial solution *partial goal query* and the corresponding goal (i.e. `partialGoalQuery.goal`) *partial goal*. The newly generated partial goal query is forwarded to other peers according to the `forward` procedure (line 10). The `solve` procedure is called recursively (line 11) to find other complete or partial solutions to the new query on the peer that generated it.

Partial solutions, when received, have to be merged with previously found local solutions related to the same partial goal query and sent back to the query source peer. This is performed in the passive thread of the discovery protocol, reported in Listing 5.3.

Listing 5.3: Discovery Protocol, passive thread

```
1  discoveryProtocol() [passive thread]
2    do forever
3      receive(q, sols);
4      if sols is nil  // case 1: handle new goal resolution request (query q)
5        and q.TTL > 0 and q has not already been solved
6          spawn forward(q);
7          spawn solve(q);
8      else // case 2: handle new solutions for query q
9        if q.refQuery is not nil // case 2a: merge partial solutions
10         localPartialSols := partialSolsTable[q.refQuery];
11         newSols := merge(q.refQuery, sols, localPartialSols);
12         reply (q.refQuery, newSols) to q.refQuery.source;
13       else // case 2b: notify complete solutions
14         add sols to completeSolsTable[q];
15         locally notify sols for q.goal;
```

The *passive thread* in Listing 5.3 is triggered by the main application component of the system, when it is started on some network node, and is passive with respect to the query resolution process. It simply consists in an infinite loop, which blocks on the `receive` procedure (line 3) until a new goal request is replied to the peer (e.g. line 4 in Listing 5.2 or line 12 in Listing 5.3). The goal query can be received along with solutions (in this case the `sols` parameter will not be `nil`).

Therefore, it is possible to distinguish two cases for the receive to return: 1) a new query has to be solved (lines 4:7, where `sols` is `nil`); 2) new solutions are available

(lines 8:15, where `sols` is not `nil`).

In the first case 1), the peer first starts the forward procedure (line 6) to effectively and efficiently spread the query through the network, and then starts the solve procedure (line 7) in order to find solutions.

In the second case 2), the peer verifies whether the received solutions are partial or complete and decides the next operations to perform. If the query refers to a partial goal (lines 9:12), partial solutions to the referred query are retrieved from the `partialSolsTable` and merged (`merge` procedure, line 11) with the newly received solutions to create a new composite solution, which will be complete for the referred query. The new merged solutions are sent back to the peer that has generated the referred query (line 12). Instead, if the solved query refers to a complete goal (i.e. referred query is `nil`, line 13), the peer has just received complete solutions to a local generated goal request. They are stored locally (`completeSolsTable`), in order to be used later in the network reorganization phase (Section 5.3), and notified (line 15) to the solution collector thread (the one executing the `collectSolutions` procedure on that peer).

5.3 Composition Overlay Networks

We call *Composition Overlay Network* any set of superpeer-managed groups generated from retrieved composite solutions and related to the same ontology concepts. CONs represent our specialization of SONs regarding service composition. To model the collaborative backward strategy and allow for the creation of CONs, we have adopted a P2P-superpeer architecture: a special kind of unstructured P2P networks where some special peers, called superpeers, act like proxies, forwarding queries to groups of managed simple peers.

Each simple node of the network runs the protocols introduced in Section 5.2, enabling publishing, discovery and composition mechanisms on it. Superpeers emerge when new service knowledge to manage exists. In fact, new composite solutions are continuously retrieved by the system as new queries are submitted and solved. Therefore, they represent useful knowledge to manage for effectively and efficiently solving successive similar queries.

In our vision, the peers holding pieces of retrieved composite solutions, related to the same service goal, form a cooperative group. Each group is handled by a superpeer, which manages links to control the cooperating peers, stores a semantic description of the group (i.e. the set of concepts included in the pre- and post-conditions

of the composite solutions) and properly forwards, as in a traditional SON, queries to the peers in the group when needed. Each peer may participate to several and possibly semantically unrelated composite solutions, by partially satisfying different service goals requested in the network, according to the distributed discovery protocol in Section 5.2. Therefore, each peer may belong to multiple superpeer-managed groups during system execution, as the network evolves and new goals are satisfied by means of distributed composite solutions.

A peer knows the CONs it belongs to, by maintaining links to group superpeers, labeled with the semantic description of the groups. Therefore, CONs are specific SONs, implemented by storing on each simple peer an index table whose entries associate the set of concepts included in the pre- and post-conditions of the composite solution service(s) to the superpeer managing that CON. The superpeer's table contains an entry for each managed group, which associates the set of concepts included in the pre- and post-conditions of the composite solution service(s) to the member peers.

Each peer in the network is associated to one specific node. A node is a computer, a virtual machine and any other software/hardware device that is able to execute the protocols described in our listings (5.1:5.9). On the same node, there can be, in general, multiple peer processes, running the proposed search and self-evolution algorithms. Superpeers emerge during the execution of the proposed protocols, after new composite solutions are discovered in the network to handle groups of peers that collaborated during past query resolutions.

Since peers are connected by logical links, we assume the existence of a first unstructured overlay network, called connectivity graph (i.e. *0-level overlay network*). In general, if peer P_i (running on node N_i) is connected to peer P_j (running on node N_j) on the connectivity graph overlay, it means that a link connecting nodes N_i and N_j exists. The link is used for propagating messages by means of the *send/reply* procedures. If P_i and P_j belongs to the same node, the link between the two peers is a local communication channel.

In the following, we assume a topology (arbitrarily complex, such as a mesh, a ring, a tree, etc.) originated by a topic-specific SON or by any uninformed set of neighbors initially known to the peers (example in Fig. 5.3). In case the overlay represents a SON, links may have been created by means of a semantic indexing process (such as the one described in [77]) performed over peers' repositories. Otherwise, they may have been retrieved by exploiting a traditional P2P bootstrapping phase or they may have been statically associated to the peers.

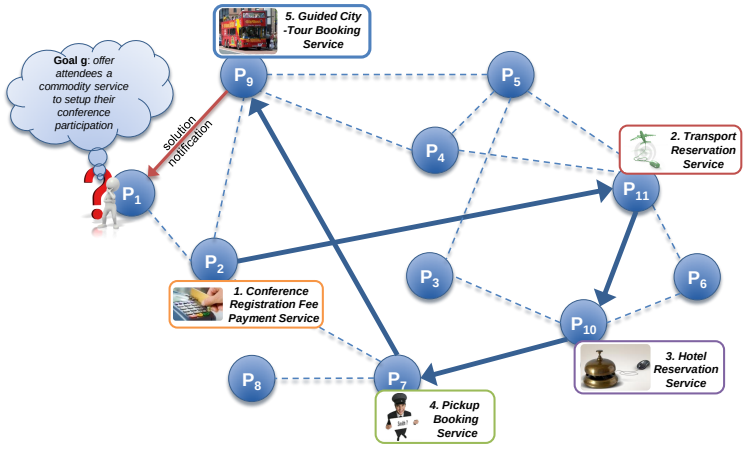


Figure 5.3: A conference-setup service composition example (resulting composite service)

In the following, we give a concrete description of our cooperative composition strategy by means of a comprehensive example. To this purpose, we propose a further scenario related to a conference organizer who wants to offer attendees a commodity service to setup their conference participation.

The goal specification (described in Fig. 5.3 and included in the query injected in the system by the conference organizer's peer) contains: predicates defining the desired final states or *effects* (i.e. conference fee paid, hotel room reserved, transport, pickup and guided city-tour booked); the set of known *pre-conditions* and available *inputs* (e.g. user generalities, conference identifier, departure and arrival time, source and destination cities, credit card number, preferred flight companies, etc.); the *output* (i.e. information about the status of each required effect).

Starting from the connectivity graph of Fig. 5.3 and by using the token abstraction (see Section 5.1) for both the service descriptions and the service goal, we report on the most relevant phases of our approach.

1) *Query forwarding and distributed backward partial resolution* (see Fig. 5.4): a peer (i.e. P_1) issues a query, containing the specification of the goal, i.e. the desired service transition from the specified pre-conditions (i.e. token A) to the specified post-conditions (i.e. token F). The query is progressively spread over an unexplored region (the one in Fig. 5.3) of the unstructured overlay. During this phase, there is no CON in the portion of the network under analysis and the only links known to the peers are the ones of the connectivity graph. These links are exploited by the forward procedure, used in Listings 5.1:5.3 and described in Section 5.4, to efficiently propagate

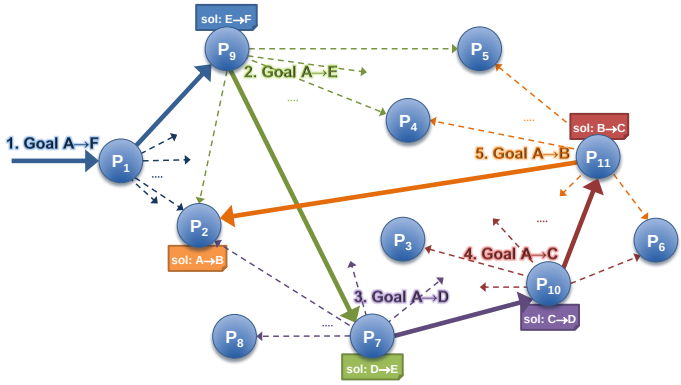


Figure 5.4: Distributed backward search (thicker lines represent links leading to partial or complete solutions)

the requested service goal (A→F) and the following partial goals (i.e. A→E, A→D, A→C, A→B). The distributed backward search algorithm, described in Listing 5.2, generates the partial goals above after discovering partial solutions (i.e. E→F, D→E, C→D, B→C, respectively);

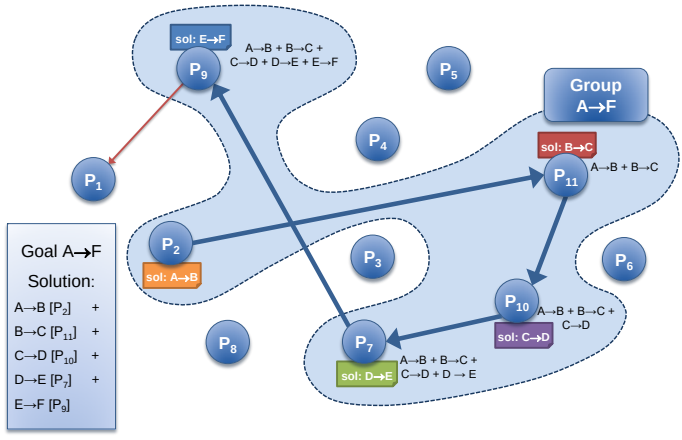
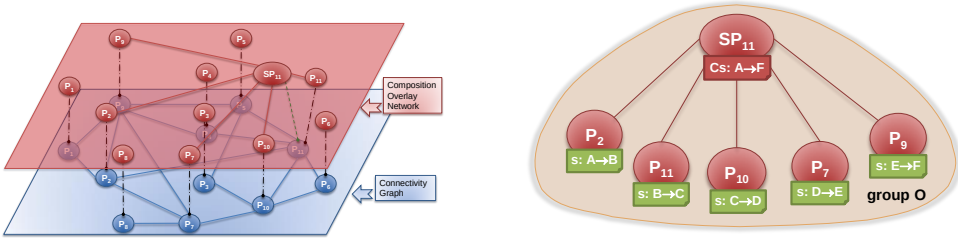


Figure 5.5: Partial solutions merging and solution notification

2) *Composite solution re-construction* (see Fig. 5.5): retrieved partial solutions are merged together by means of a forwarding distributed merging process. The node that has retrieved the first part of the composite solution (i.e. P_2, solution A→B) – the last to be found in temporal order – starts the merging process. The peer that has found the last part (i.e. P_9, solution E→F) terminates it. Finally, this peer notifies

the complete composite solution to the requesting peer (i.e. P_1). In the example, P_2 sends the partial solution A→B to P_{11}, which aggregates A→B with B→C. Partial solution A→B + B→C is sent back to P_{10}, which generates A→B + B→C + C→D and sends it to P_7. P_7 aggregates A→B + B→C + C→D with D→E and sends it to P_9, which generates the final complete solution A→B + B→C + C→D + D→E + E→F that is notified to P_1. It is worth to note that merged solutions are transferred as sequences of atomic services (e.g. A→B + B→C + C→D) and not as atomic services (e.g. A→D). We could provide A→D in only two cases: 1) an atomic service A→D is available in some peer's repository; 2) the system has previously retrieved the composite solution A→B + B→C + C→D (or any other combination that satisfies goal A→D) from a previous query and a superpeer has stored it in its local repository, exposed in a merged form, having A as pre-conditions and D as post-conditions.



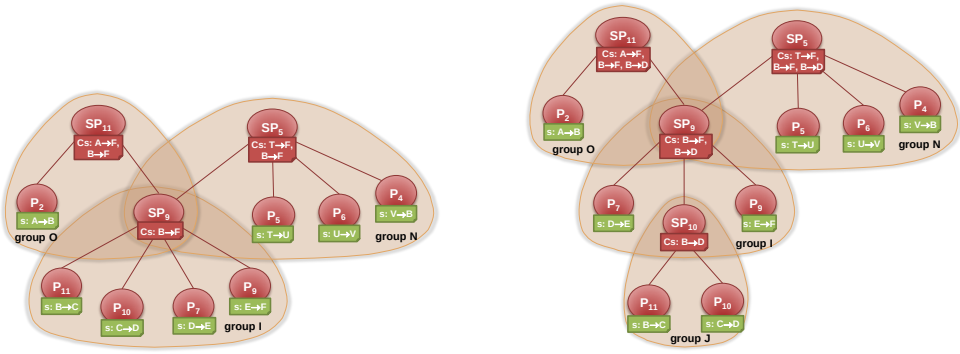
(a) (b)

Figure 5.6: Overlay network built from retrieved composite solution, two different representations

3) *Overlay network formation* (Fig. 5.6): the peers that cooperated in solving the query (P_2, P_7, P_9, P_{10}, P_{11}) are grouped together in a new CON, managed by a new superpeer (i.e. SP_{11}), which is started on one of the nodes running the cooperating peers. This superpeer will be in charge of managing the overlay network related to the found composite solution, by holding the links to all the other peers and exploiting them for solving successive similar queries. The peer that submitted the request (i.e. P_1) is in charge of starting and managing the network reorganization phase to build the overlay: messages to create the new group are sent from this peer to the ones that contributed to find the composite solutions. The superpeer is elected according to a distributed election algorithm (more details in the following), started on the selected node and asked to publish the found composite solution (i.e. A→F) in its local repository. The other peers join the new group and store a reference to the group superpeer, associated to the semantic characterization of the CON, by creating a new entry in their local CON table.

Fig. 5.6 gives two alternative representations of the built overlay network. Fig. 5.6a focuses on the presence of multiple levels of overlay networks; Fig. 5.6b on the structure of the new overlay network (a group of peers connected to one superpeer) and the services available in the repositories of peers and superpeers.

4) *Solution of a query by exploiting the composition overlay network*: a peer (e.g. P_2) issues the query A→F again. In this case, P_2 transfers the query following the semantic link associated to the previously emerged CON (Fig. 5.6), which is semantically equivalent to the submitted query. Thus, SP_{11} receives the query, finds a complete solution in its repository and returns it to P_2. In case another peer, not belonging to the overlay, submits the query A→F again, propagation over the connectivity graph is necessary to reach at least one peer (or the superpeer itself) of the overlay network, in order to quickly find the solution associated with the CON.



(a) Overlay network evolution after solving query T→F

(b) Overlay network evolution after solving query B→D

Figure 5.7: Overlay network evolution according to a tree-like structure

5) *Tree-structured overlay evolution* (Fig. 5.7): P_1 issues a third query (i.e. T→F), whose post-conditions are semantically equivalent to those of the previous query (i.e. A→F). In this case, the overlay network is exploited to find the partial solution A→F, which however does not lead to a complete solution. Finally, the T→F query is cooperatively solved by the peers P_4, P_5, P_6, P_7, P_9, P_{10} and P_{11}. After a timeout, the network reorganization process starts. Fig. 5.7a shows how the composition overlay network evolves according to a tree-like structure, by retrieving the intersections among the available groups and electing new superpeers to handle the new structure. The requesting peer (P_1, not shown in the picture) manages the overlay reorganization process.

By inspecting the received solution, P_1 retrieves the set of solving peers (i.e. P_4, P_5, P_6 from the T→B composite partial solution and P_7, P_9, P_{10}, P_{11} from the B→F one) and contacts all of them asking for the groups they participates to, semantically related to the solved goal. Peers P_7, P_9, P_{10}, P_{11} replies by sending back the only semantically relevant group they belong to, i.e. group O, generated in the previous phase for goal A→F, while peers P_4, P_5, P_6 sends back no group. A temporary group T, containing all the solving peers P_4, P_5, P_6, P_7, P_9, P_{10}, P_{11}, is considered for computing intersections. After having collected groups (a timeout is started to collect responses), the requesting peer analyzes them for finding intersections. P_1 identifies the sets of peers belonging to each distinct group (i.e. by contacting group superpeers) and evaluates intersections between them (e.g. P_7, P_9, P_{10}, P_{11} is an intersection between group O and group T).

The intersection represents a new group, I, whose superpeer is elected among its members (e.g. SP_9 in Fig. 5.7a). Similarly, the peers in group T, which do not belong to any intersection, represent a new group, namely N (i.e. P_4, P_5, P_6, with superpeer SP_5). The intersection group members have to be removed from their previous group O, while its superpeer (SP_9) has to be added as a member to both O and N. To this purpose, P_1 sends messages to the interested peers and superpeers asking them to remove themselves from their old groups (if any) and to join the new ones. Fig. 5.7b shows the evolution of the overlay network after another query (i.e. B→D) is solved.

Listing 5.4: Network Reorganization Protocol, executed by the query submitter peer

```
1  reorganizeNetwork(Query q)
2    wait networkReorganizationTimeout;
3    solutions := completeSolsTable[q]
4    solvingPeers := retrieveParticipants(solutions);
5    if solvingPeers.size > 1
6      foundGroups := new Set<Group>();
7      peersWithPreviousGroups := new Set<Peer>();
8      foreach peer in solvingPeers
9        otherPeerGroups := retrieveGroups(peer, q.goal);
10       if otherPeerGroups is not empty
11         add peer to peersWithPreviousGroups;
12         add otherPeerGroups to foundGroups;
13     peersWithoutPreviousGroups := remove peersWithPreviousGroups from solvingPeers;
14     if peersWithoutPreviousGroups is not empty
15       add peersWithoutPreviousGroups to foundGroups;
16       createNetworkGroup(peersWithoutPreviousGroups, solutions);
17     intersections := retrieveIntersections(foundGroups);
18     reorganizeNetworkGroups(foundGroups, intersections);
```

The `reorganizeNetwork` procedure, introduced in Section 5.2 and reported in Listing 5.4, is aimed at creating and managing the CONs, according to the definitions given at the beginning of this section and the mechanisms described in the previous example (phases 3 and 5). We remark that CONs are managed through index tables of peers and superpeers, whose entries (semantic links) maintain the association between

a set of concepts (those included in the pre- and post-conditions of the composite solution service of a specific CON) and the superpeer managing that CON (for simple peers) or the member peers (for superpeers).

Starting from the connectivity graph, when a composite solution is identified, the network implicitly aggregates the participating peers to form a new group. A timeout mechanism is used for stopping the reception of solutions for the locally generated query and starting network reorganization (line 2). The process is executed continually in the network, giving rise to several virtual layers of links (CONs), placed over the connectivity graph and managed at the superpeer level. Superpeers maintain both pointers (peer ID) to the peers making up the new composite solution and a semantic characterization of the solved goal, which identifies the CON.

Simple peers maintain pointers to their superpeers and a semantic characterization of the CON. The links of the semantic overlay networks, stored at the superpeer level, make it possible the routing of new queries along the most convenient directions for resolution (i.e. the most semantically similar links). The self-evolution algorithm is periodically started and orchestrated by the peer that submitted a service request (line 5 in Listing 5.1). Thresholds can be defined in order to avoid that the reorganization is performed too frequently.

In the first phase, the peer orchestrating the reorganization protocol identifies those peers that cooperated to retrieve composite solutions by providing partial solutions (line 4). In lines 6:12, the cooperative peers are requested to provide information about previously formed groups they belong to, which are semantically related to the solved query. The ones that do not belong to any previous group have to be organized in a new group (lines 13:16). To create the group, the peer performing network reorganization has to exchange group-creation messages with the member peers and orchestrate a distributed superpeer election algorithm (`createNetworkGroup`, line 16). At the end of the group creation phase, the superpeer receives the composite solutions and publishes them in its local repository.

In our current implementation of the peer protocol, superpeer election is based on the evaluation of two parameters for any network node N: 1) degree of node N on the connectivity graph (d_N); 2) number of already running peer/superpeer processes on the node N (p_N). The node with the highest degree and the lowest number of already running peer processes is chosen (highest value of the difference d_N - p_N), with a random choice as tie-breaking rule. The d_N parameter has been considered to make the superpeer easily accessible from other peers in the network; the p_N parameter

is a balance factor that aims at not overloading the node[2]. The new superpeer will be reachable by other peers through the newly added semantic links and the original connectivity graph links, pointing to the node the superpeer is running on.

In order to maximize the reuse of previously acquired knowledge, overlapping groups are reorganized by finding their intersections and storing them into a proper collection of tree-structured data (the `intersections` variable in line 17). To this purpose, the `retrieveIntersections` procedure is used. Any intersection found has to be turned into a new group with its semantic description and superpeer, which is elected according to the previously described election algorithm. The superpeers of each intersection group have to appear as members in any other group owning that intersection. The orchestrating peer gives the final structure to the CON (`reorganizeNetworkGroups`, line 18) by: 1) sending group reorganization messages (`add/resign` from group messages) to peers and superpeers; 2) sending group creation messages to create the intersection groups.

When a node X of a group Y finds new composite solutions, it informs the superpeer of its original group Y, which publishes them on its local repository. The repository will be explored by the superpeer when new queries are received, allowing the re-use of already performed service compositions. This way, it is possible to more quickly solve queries which are semantically similar or identical to previously solved ones, without generating another distributed backward search (even though propagation in the group would be performed efficiently by exploiting group semantic links).

Peers and superpeers may fail, leave the system or crash during system operation. Conversely, new peers may join the system, connect to a set of neighbors and eventually their services will be discovered by means of the searches performed over the connectivity graph.

In order to preserve consistent information in the network, at the CON level, we have designed a proper consistency protocol, detailed in Listing 5.5.

The protocol is executed periodically by each peer in the network that belongs to some CON and by the CONs' superpeers. Since we perform offline composition, the `consistencyCheckTimeout` is chosen in relation to the maximum (configurable) time to find a service composition in the network.

This timeout represents the time for a found composition to be still valid, i.e. the time slot during which we assume the state of the environment (i.e. the context which

[2]An alternative approach could use the actual machine load (i.e. the number of all processes running on node N) as p_N. According to this view, if the number of peers/superpeers running on each network node is low, it is possible to avoid that hubs are selected frequently as superpeers.

the service composition depends on) does not change. The `receive` procedure, used in the following listing, is blocking and allows for specifying both the type of the expected message and a wait timeout. It returns a `status` value indicating whether the receive call was successful or not. Messages are created by means of the `Message` procedure, by indicating the type of the specific message.

Listing 5.5: CON consistency protocol, executed by each peer/superpeer in the network

```
1  consistencyProtocol()
2    do forever
3      wait consistencyCheckTimeout;
4      if isPeer(this)
5        foreach group in this.groups
6          group.isAlive := false;
7          send (new Message(KEEP_ALIVE)) to group.superpeer;
8          status := receive(groupAliveMsg, [GROUP_ALIVE, GROUP_INVALID], waitTimeout);
9          if status == TIMEOUT_EXPIRED or groupAliveMsg.type == GROUP_INVALID
10           deleteNetworkGroup(group);
11         else
12           group.isAlive := true;
13     else if isSuperpeer(this)
14       foreach group in this.managedGroups
15         missingPeers := false;
16         aliveList := new List<PeerId>();
17         foreach peer in group.peers
18           status := receive(keepAliveMsg, KEEP_ALIVE, waitTimeout);
19           if status == TIMEOUT_EXPIRED
20             missingPeers := true;
21           else
22             add keepAliveMsg.source to aliveList;
23         groupIsConsistent := true;
24         if missingPeers
25           groupIsConsistent := checkGroupSolutions(alivePeers);
26         if not groupIsConsistent
27           msg = new Message(GROUP_INVALID);
28           deleteNetworkGroup(group);
29           broadcast(msg, group.peers);
30           // SP does not send any keep alive message to intersection's superpeers
31         else if local failure or SP is leaving
32           delegateSuperpeer(group);
33         else
34           foreach upperLevelGroup in this.groups
35             send (new Message(KEEP_ALIVE)) to upperLevelGroup.superpeer;
36           msg := new Message(GROUP_ALIVE);
37           broadcast(msg, group.peers);
```

After the consistency timeout has expired (line 3), each simple peer notifies it is alive by sending a `KEEP_ALIVE` message to the superpeer of each group it belongs to (line 7). Then, the peer waits for a message from the superpeer (line 8) to establish whether the group and the associated superpeer are still alive (lines 9:12). If the superpeer or the group members are not alive, the peer can delete this group (lines 9,10). According to a pessimistic behavior, during the group checking phase (lines 5:12), the peer temporary disables (line 6) the group, in order to avoid that inconsistent group knowledge is exploited for query forwarding (see Section 5.4). The group is enabled again by the peer after receiving the `GROUP_ALIVE` notification from the superpeer (lines 11,12).

Each superpeer will collect the `KEEP_ALIVE` messages coming from the managed

peers (lines 17,18) to establish whether the managed group is still alive or not. If all the group members are alive, the group is still consistent; otherwise, the superpeer assumes that some member peers left the system or crashed.

In the latter situation (lines 24,29), group solutions will be inspected with respect to the missing peers. Superpeers remove from the group those solutions depending on the missing peers and, if no valid solution is still associated with the group, they mark it as invalid (lines 26,27), delete it (line 28) and notify this status to the member peers (line 29). It is worth to note that in this case, the upper-levels superpeers, managing the invalid group, will remove the sub-group in turn because they will receive no `KEEP_ALIVE` message from the lower-level superpeers.

If the group is still consistent, the local state of the superpeer is checked to verify whether it is still in condition to act like a superpeer (e.g. it can be overloaded or in a faulty state) and, if required, a delegation mechanism will start (lines 31,32). The delegation mechanism is useful to keep composition overlay trees valid, even if one superpeer of the tree is going to leave the network. As an example, in Fig. 5.7b, if SP_9 leaves, it will give its knowledge to one of its managed peer and properly update the managed data. This change will be then propagated to SP_{11} and SP_5. The delegation mechanism is based on the execution of a new round of the superpeer election algorithm previously described, by excluding the node where the leaving super-peer is running as a candidate for the election.

Finally, if delegation is not required, (lines 33:37) the superpeer informs the upper-level superpeers it is alive (lines 34,35) and the member peers that the group is still valid (lines 36,37).

To execute the consistency protocol, simple peers exchange $\mathcal{O}(g \cdot \overline{p})$ messages with superpeers, where g is the number of groups available in the system and $\overline{p}$ is the average number of member peers per group[3]. In addition to these messages, superpeers exchange $\mathcal{O}(g \cdot \overline{h})$ messages for maintaining the CONs tree-like structure, where $\overline{h}$ is the average height of the composition overlay trees. Finally, $\mathcal{O}(g \cdot \overline{p}^2)$ messages are required for leader election inside the CONs group (in the worst case, i.e. Bully algorithm complexity). The final message complexity of the consistency protocol is therefore $\mathcal{O}(g \cdot (\overline{p}^2 + \overline{h}))$.

[3] $\overline{p}$ depends on the network size and the maximum allowed composition length (configuration parameter). In general, $\overline{p} \ll n$.

5.4 Query Selective Forwarding

In an unstructured P2P network, avoiding flooding is essential to reduce the number of messages exchanged for solving discovery requests and to ensure, at the same time, lower composition times without significantly affecting the ability of finding all the existing solutions in the network.

Listing 5.6: Query forwarding procedure

```
1 forward(Query q)
2   if isSuperpeer(this)
3     foreach group[matching q.goal] in this.groups
4       reply (q, nil) to group.superpeer;
5   else
6     propagateQueryToNeighbors(q);
7     if (isNeighbor(q.sender) and not isSuperpeer(q.sender)) or query submitted locally
8       foreach group[matching q.goal] in this.groups
9         reply (q, nil) to group.superpeer;
```

To this end, we propose a novel probabilistic forwarding algorithm (that can be considered as an informed gossip technique) that exploits knowledge about both the connectivity graph (such as density) and the CONs. The overall forwarding algorithm (Listing 5.6) distinguishes two cases:

1) Superpeers reply queries (the `reply` primitive is used with no solutions, acting therefore like a regular send) to the other known superpeers, managing semantically goal-related groups (lines 3,4). This way, we facilitate query diffusion to the groups with more useful information for query resolution. When such groups do not exist, superpeer propagation can be completed by forwarding the query to neighbors over the connectivity graph, during the execution of the superpeer implementation of the solve procedure (Subsection 5.4.1). It is invoked by the passive thread (see Listing 5.3, line 7) or the active one (see Listing 5.1, line 4), in case the superpeer itself submits the goal request;

2) Simple peers forward queries to their neighbor nodes, by means of the `propagateQueryToNeighbors` procedure (see Subsection 5.4.2) on the connectivity graph (line 6), and to the superpeers of the groups they belong to (line 9). Specifically, peers propagate a query to superpeers only in case the request has been created locally or received from a neighbor simple peer (line 7). The only groups considered for propagation are those semantically relevant with respect to the goal query (line 8).

5.4.1 Superpeer Propagation Algorithm

The implementation of the solve algorithm (see Listing 5.7) for superpeers is a slight variant of the search procedure executed by simple peers (see Listing 5.2).

Listing 5.7: Query solving procedure for superpeers

```
1  solve(Query q) [Superpeers variant]
2    groupSols := localSearch(q);
3    if groupSols is not empty
4      reply (q, groupSols) to q.source;
5    else
6      spawn propagateQueryToNeighbors(q);
7      groupPartialSols := localBackwardSearch(q);
8      add groupPartialSols to partialSolsTable[q];
9      foreach partialSol in groupPartialSols
10       partialGoalQuery:= new Query(q.goal, partialSol);
11       spawn forward(partialGoalQuery);
12       spawn solve(partialGoalQuery);
13     spawn forwardToManagedGroups(q.goal);
```

The only differences with the simple peers' solve are: 1) searching (lines 2 and 7) is performed over the group solutions stored in the superpeer repository (i.e. previously found composite solutions); 2) the algorithm includes propagation of the received query to the superpeer node's neighbors, when no complete solution is found (line 6); 3) the peers belonging to groups managed by the superpeer, and semantically related to the query goal, are directly forwarded (`forwardToManagedGroups`, line 13) to search for new (partial) solutions that are still unknown to the superpeer.

In relation to point 2), superpeers do not perform propagation over the connectivity graph immediately upon receiving the query, as in the case of simple peers. This consideration comes from the main forwarding algorithm in Listing 5.6, where the `else` body (lines 6:9, related to the simple peer case) contains the propagation to neighbors at line 6, while the `if` body (lines 3,4 for the superpeer case) contains none. Superpeers perform propagation over the connectivity graph only after they have verified that there is no complete solution in their repository (line 6 in Listing 5.7). This avoids message overhead when a complete solution can be directly found on the superpeer, by exploiting the larger knowledge available at this level.

5.4.2 Propagation over the Connectivity Graph

The algorithm `propagateQueryToNeighbors`, described in the following, implements the forwarding mechanism adopted by every peer on the connectivity graph and represents the core of the probabilistic forwarding mechanism.

In a traditional flooding approach, each peer forwards every message it receives to all of its neighbors. Even though most of the works addressing service composition in unstructured P2P networks uses flooding for propagating messages, we compare our approach to an optimized variant of the basic flooding technique, which we call *optimized flooding* in the following. With optimized flooding, the query message to propagate maintains the list of neighbors already forwarded by the sending peer. When the receiving peer has to decide about propagation, the list is considered to

exclude local neighbors that already received the query from the sender. The list of neighbors in the propagated query is then updated with the new information about the actual new forwarded peers.

Our probabilistic forwarding algorithm is based on the introduction of a propagation threshold, denoted as τ in the following, limiting the number of neighbors to which the query should be forwarded (*forwarded peers*). In other words, the propagation threshold represents the fraction of neighbors that a peer can select for propagation whenever forwarding over the connectivity graph is demanded. It is dynamically computed by using updated and locally available network and query-related information. The number of forwarded peers (f) acts as a dynamic fan-out (if our technique is compared with a traditional gossip algorithm) and is computed by each peer as part of the `propagateQueryToNeighbors` procedure according to the formula:

$$f = \lceil \tau \cdot \lambda \rceil, \text{ where: } \lambda = \begin{cases} l-1, & \text{if sender is a neighbor} \\ l, & \text{otherwise} \end{cases}, \tag{5.1}$$

being l the neighborhood size for the current peer. Variable f represents the maximum number of neighbors the peer will contact for query propagation (i.e. *propagation fan-out*). As in the optimized flooding strategy, the information related to the previously forwarded peers is coded in the query message by the sender. By excluding the possible common neighbors that already received the query from the sender, up to f neighbors are randomly selected among the whole neighborhood and stored as the new list of propagated peers in the query message. Finally, the message is forwarded to them.

As an example, let us assume peer P sends a query message m to an *f-size* subset of its neighbors at stage S_i. The query will contain the list ℓ of peers going to be forwarded by P (e.g. f equals to 4 and ℓ is [*Q, R, S, T*]), together with other information (e.g. the semantic description of the goal to solve). At stage S_{i+1}, peer Q (like R, S and T) receives message m from P, computes an updated value for f according to its local knowledge (e.g. f equals to 3) and explores the list ℓ of already forwarded peers contained in the m query, in order to decide the new neighbors for propagation. If Q shares some of the neighbors in ℓ, e.g. *R, S* or *T*, the random selection of neighbors for propagation will not include these peers, in order to reduce the probability that peers receive the same message multiple times. Thus, P will select other neighbors to reach the required f upper limit (e.g. [*U, V, W*]). Finally, Q will replace the old list in the query with the new one ([*U, V, W*]), before sending it to the new neighbors.

We have identified three different kinds of network information that can be computed at any time by a peer for the dynamic evaluation of τ: 1) Availability of relevant service composition overlays (CONs); 2) Global density of the network; 3) Number of peers (hops) crossed by the goal query from the source to the current peer. The information above makes it possible to distinguish three contributions to threshold τ, which we denote as τ_{Groups}, $\tau_{Density}$ and τ_{Hops}.

The value of threshold τ is evaluated as a weighted (the weights ω_{Groups}, $\omega_{Density}$ and ω_{Hops} are defined at configuration time) and normalized (weights sum up to 1) sum of the three contributions. Each contribution is a threshold itself, is defined in the range [0, 1] and is dynamically computed when the `propagateQueryToNeighbors` algorithm is executed by a peer, using the currently available data. Additional thresholds could be considered if some other knowledge may be accessed in the service P2P networks and used to improve forwarding, by means of a proper strategy and a specific tuning process. We considered the most relevant properties of service networks and kept low the number of thresholds to reduce the overhead tied to the network knowledge retrieval.

In the following, we explain the semantics of the thresholds, the formulas and mechanisms used by the system to compute them, to help understanding the potential impact on the metrics that are evaluated in Section 5.6 through several simulations performed by using different weights for the thresholds above.

Semantic Group Threshold (τ_{Groups})

τ_{Groups} is computed by considering the number of groups participated and/or controlled by the forwarding peer. Each of these groups is related to previously found service compositions (i.e. it is part of a CON) and has a semantic characterization, which makes possible to filter the ones not relevant to the goal resolution. The number of relevant and reachable groups is denoted as η. It represents the independent variable of the τ_{Groups} evaluating function.

The rationale behind the evaluation of this threshold is simple: the more semantic links are available for query forwarding, the less it is needed to propagate to neighbors on the connectivity graph. Propagation can be mainly handled through superpeers' connectivity information. Therefore, we identified the following requirements for the evaluating function: if there is no useful group ($\eta = 0$), τ_{Groups} has to be 1, since the query forwarding cannot rely on the semantic forwarding; τ_{Groups} has to be decreasing with respect to η; variable η has theoretically no upper bound, hence τ_{Groups} has to asymptotically decrease to 0 when η increases in the range $[0, +\infty[$.

By taking into account these requirements, we have considered the equilateral hyperbola function reported below, which has been tested, in terms of performance indexes, in the simulations of Section 5.6:

$$\tau_{Groups} = 1/(1+\eta) \tag{5.2}$$

Density Threshold ($\tau_{Density}$)

$\tau_{Density}$ is computed by means of a standard gossip protocol, continually performed by every peer of the P2P network in parallel with the composition/discovery protocol (Listings 5.1 and 5.3). An anti-entropy protocol, based on the push-pull strategy described in [50], has been defined in order to allow any peer to compute the average number of neighbors on the connectivity graph (i.e. average degree): 1) each peer P stores a local approximation of the average number of neighbors in the network as its $state_P$. The initial value is chosen as the number of peer's neighbors on the connectivity graph; 2) each peer P performs a random selection of the neighbor Q to gossip with from its neighborhood; 3) when receiving the gossip information $state_P$ from neighbor P, peer Q replies its state to neighbor P and updates it to the value: $(state_P + state_Q)/2$. This state update has been proved to converge to the global state average in [50]. The period between gossip iterations has to be quite low with respect to the average period of goal query submission, in order to make gossip quickly converge to reliable density information when the discovery/composition process is inactive.

Our gossip-based anti-entropy protocol has been designed to work during the inactive phases of our system, i.e. when there are no active goal queries on the peer running over the network peers, in order not to introduce message overhead during the stages of P2P discovery. To this purpose, the anti-entropy protocol is temporally suspended on a peer that is busy in solving queries, thus avoiding the proliferation of gossip messages. However, since the gossip message elaboration overhead is typically low, network density information is also included by each peer within the messages regularly exchanged during query forwarding (piggybacking). This way, it is still possible to update the peer's local density information, even when the composition/discovery protocol is active, without generating specific gossip messages.

Finally, in order to know how much the current state is a reliable representation of the global network density, we incrementally compute, on each peer, the standard deviation (σ) of the local density information (δ) at any state update. At the beginning of the gossip protocol, we assume an infinite standard deviation. Hence, we

introduce a parameter ($\overline{\delta}$), defined as:

$$\overline{\delta} = \begin{cases} \delta - \sigma, & \text{if } \sigma < \delta \\ 1, & \text{otherwise} \end{cases}, \tag{5.3}$$

which we use to compute the $\tau_{Density}$ threshold. The rationale is that the denser is the network (i.e. higher $\overline{\delta}$), the less propagations will be necessary for the query to reach the different peers in the network, because of the presence of many alternative paths. To allow a finer grained control of the density threshold, we have introduced the following parameterization of $\tau_{Density}$:

$$\tau_{Density} = K/(\overline{\delta} + K - 1), \tag{5.4}$$

with $K \geq 1$. By increasing the K coefficient, it is possible to use a smoother hyperbole in order to softly reduce the number of messages as $\overline{\delta}$ increases. This can be useful to manage the selective behavior when the network is sparse and a proper number of messages has to be injected to increase the probability of finding solutions.

Traversed Hops Threshold (τ_{Hops})

τ_{Hops} is computed by considering the number of hops that a query has already crossed. The idea is that the more hops the query has already crossed, the less it is likely that propagation will help to solve it. Hence, we have introduced the independent variable ρ, representing the number of traversed hops. It is set to 1 when a goal query is created on some peer from the user specified goal or from retrieved partial solutions; its value is transported by the query itself and is increased by one every time the query is received on a peer and relayed to its neighbors. By using ρ, we have defined the following evaluating function for τ_{Hops}:

$$\tau_{Hops} = 1/\rho \tag{5.5}$$

5.5 Comparison with DHTs and SONs

In Table 5.1, we propose a comparison among DHTs, SONs and CONs that reports on the advantages and disadvantages of each of them, extending the considerations reported in [7, 77].

In DHTs, semantic service discovery can be performed by using a hash function applied to the query post-condition. The hash function returns the *id* of the peer

Property	**Architectures**		
	DHTs	**SONs**	**CONs**
Content placing	Fixed, based on hashing	Each peer is responsible for its content	Each peer is responsible for its content
Content retrieval	Exact match	Semantic (based on similarity)	Semantic (based on similarity)
Network structure	Fixed	Variable	Variable
Discovery cost	$\mathcal{O}(log(N))$	Variable	Variable, but optimized
Topology management	Stabilization protocol	Semantic links need to be rearranged	Composition semantic links need to be rearranged
Semantics	None	Exploit a semantic artifact (e.g. ontology)	Exploit a semantic artifact (e.g. ontology) for atomic and composite service matching
Composition cost (for m *k-service* compositions)	$k \cdot m \cdot DiscoveryCost_{DHT}$ (orchestrated case) or $k \cdot c \cdot DiscoveryCost_{DHT}$ with $1 \leq c \leq m$ (cooperative case)	$k \cdot DiscoveryCost_{SON}$	$k \cdot DiscoveryCost_{CON}$ (worst case) or $DiscoveryCost_{CON}$ (best case)

Table 5.1: DHT(Chord), SON and CON-based semantic service composition

hosting the services, if any, whose post-condition exactly matches the query post-condition. We remark here that the exact match is an important limitation of DHT-based approaches, whereas SONs and CONs support more flexible semantic matching functions. In particular, the discovery cost depends on the finger table associated to each peer (we assume a Chord implementation). The node's finger table contains the Chord ids of its neighbors, which are located on the ring at exponentially increasing distances. This guarantees that any query is answered in $\mathcal{O}(log(n))$ hops, where n is the number of nodes in the network. In SONs, discovery time depends on the overlay topology, on the semantic matching similarity function and on the forwarding algorithm (typically flooding). In the case of CONs, we have instead an optimized discovery cost, stemming from the use of our informed selective forwarding strategy, described in Section 5.4.

Composition cost in DHTs depends on two different possible implementations: 1) *Orchestrated*, in which composition processing is performed only by the submitter node, considering that the query is not propagated to other peers. 2) *Cooperative*, in which the query is propagated to each looked-up peer, which in turn is able to continue the search for the other component services. In the former, the overall cost to find m

k-size service compositions (i.e. composition cost) is $k \cdot m \cdot DiscoveryCost_{DHT}$. In the latter, the overall cost to find m *k-size* service compositions depends on the distribution of the component services in the network. In fact, since in a DHT one single peer (P_i) hosts the set S of services that match the query at the i^{th} composition step (i.e. with the same post-condition in our case), that peer is responsible for solving the remaining partial goal queries. For each service in S, a partial goal query having the service pre-condition as its post-condition can be defined. If the partial goal queries of the $(i+1)^{th}$ composition step have all the same post-condition (worst case), the hash function returns one single peer again (P_{i+1}) which is responsible in turn of solving the remaining part of the queries. At the end, if m *different* compositions exist in the system, the overall composition cost will be $k \cdot m \cdot DiscoveryCost_{DHT}$. On the other hand, if the partial goal queries of the $(i+1)^{th}$ composition step have different post-conditions (best case), the hash function returns m different peers which are able to solve the remaining part of the queries in parallel. At the end, if m different compositions exist in the system, the overall composition cost will be $k \cdot DiscoveryCost_{DHT}$. Therefore, in general, the overall composition cost is $k \cdot c \cdot DiscoveryCost_{DHT}$, with $1 \leq c \leq m$.

In CONs, composition cost is equal to the CON discovery cost (optimized by our probabilistic forwarding technique) if the compositions already exist in the overlay. In fact, since they are available in superpeers' repository, it is only necessary to reach one of the peers in the overlays to get access to the superpeers and thus to the available solutions. If the compositions do not already exist in the overlay (worst case), the overall composition cost will be $k \cdot DiscoveryCost_{CON}$, because m searches for *k-size* service chains will be performed concurrently over the connectivity graph (each one costing $k \cdot DiscoveryCost_{CON}$).

5.6 Evaluation

To evaluate the techniques and the algorithms discussed in the previous chapters, we relied on the PeerSim [71] simulator. PeerSim is a Java open-source P2P network simulator, including an event-based engine, completely driven by events and node messages, offering concurrency simulation and a simulated transport layer.

On top of PeerSim, we implemented the overlays, the related management algorithms and the composition logic. The evaluation of the proposed techniques has been performed by considering several configurations and focusing on three performance indexes:

1. *Recall*: system's ability to find simple or complex solutions (among the existing ones) to a goal request, when services published on peers' repositories make it possible to satisfy it. In the following experiments, the recall information is computed as the ratio of the number of solutions found by our system during the simulation cycles to the number of existing ones.

2. *Message overhead*: the number of request and response messages exchanged among the peers in the network to find solutions to a goal request. Messages are counted considering the whole network until the available solutions to the query under test are received.

3. *Resolution time*: the time required for the submitter to receive the available solutions to the requested goal.

We initialized the P2P network with a specific number of nodes and a connectivity graph. At the beginning of simulations, each node of the simulated network hosts one single peer process, able to publish, discover and compose services; no CON is available. Service descriptions are published on each peer in the simulation initialization phase or cyclically. The latter is useful to evaluate the behavior of the system with respect to different service allocations on the peers, obtained by shuffling service descriptions on the available peers. Shuffling is performed before the beginning of a new simulation cycle. To make simulations consistent with realistic usage scenarios, we introduced three kinds of delay:

1. *Semantic elaboration delay*: models the delay introduced by a realistic semantic matching between the query received by the peer and a service description available on that peer. This delay includes the reasoning time required in order to semantically compare post- and/or pre-conditions. For backward partial resolution, we considered half this delay, since only post-conditions are compared.

2. *Transmission delay (td)*: models the delay for placing a message from the application layer (peer sending a message) to the network layer (the node), when no multicast communication is available. If a peer on node A has to send at time t_0 a message to peers on nodes B, C and D, the message for B will be sent at t_0, for C at time $t_0 + td$ and the one for D at time $t_0 + 2 \cdot td$.

3. *Network latency*: is the latency delay for the simulated transport protocol, used for message exchanges among the network nodes. It represents the time required for a message sent from a node to reach the destination one, in case the sender and the destination nodes are direct neighbors.

We simulated our P2P network using the following values for the parameters above:

1. *Semantic elaboration delay*: 400 ms for local complete solution; 200 ms for partial backward solution. These delays have been computed by executing a number of queries towards some of the matchmakers used in the S3 contest, the annual contest on Semantic Service Selection [3]. In particular, we focused on ISeM [55], since it offers matching capabilities based on IOPE (Inputs-Outputs-Preconditions-Effects) descriptions.

2. *Transmission delay*: 1 ms.

3. *Network latency*: uniform random variable in the range [10ms, 130ms]. This range refers to latency measured on the Internet when sending small messages to very distant destinations (130 ms) or very close (10 ms) ones [4].

We evaluated the techniques with different sizes of the network (and number of published services). For the connectivity graph, we used several topologies, like a star and random meshes, focusing our attention on a dense topology of the network, Δ, and a sparse one, Σ. In the first case (Δ), we used a random mesh where each node has at least *network_size/10* neighbors (the network become denser as the size increases); in the second one (Σ), we used a random mesh with a fixed average number of neighbors (between 2 and 4 neighbors).

Configuration 1:	$\{\omega_{Groups} = 0.4$, $\omega_{Density} = 0.3$ and $\omega_{Hops} = 0.3$, *density hyp. coeff. (K)* $= 1$ $\}$
Configuration 2:	$\{\omega_{Groups} = 0.4$, $\omega_{Density} = 0.3$ and $\omega_{Hops} = 0.3$, $K = 2\}$
Configuration 3:	$\{\omega_{Groups} = 0.4$, $\omega_{Density} = 0.5$ and $\omega_{Hops} = 0.1$, $K = 1\}$
Configuration 4:	$\{\omega_{Groups} = 0.4$, $\omega_{Density} = 0.5$ and $\omega_{Hops} = 0.1$, $K = 2\}$
Configuration 5:	$\{\omega_{Groups} = 0.4$, $\omega_{Density} = 0.3$ and $\omega_{Hops} = 0.3$, $K = 3\}$
Configuration 8:	$\{\omega_{Groups} = 0.4$, $\omega_{Density} = 0.5$ and $\omega_{Hops} = 0.1$, $K = 5\}$

Table 5.2: Algorithm configurations for evaluation

As a reference for the reader, Table 5.2 lists the configurations adopted to set weights and coefficients of the probabilistic forwarding algorithm in our experiments[5].

[4]Since our connectivity graph is an overlay network (built at the edge of the Internet), direct connections between peers are not necessarily faster than indirect connections. The considered time interval has been evaluated by approximating a Gaussian distribution generated by running the ping command (to measure the RTT) towards a set of worldwide-distributed hosts. This way the obtained times are realistic. Moreover, we have performed several runs and reported statistical aggregations of our experiment results (min-max intervals, percentiles, etc.) that make them (statistically) significant and comparable.

[5]The weights and coefficients in Table 2 are the result of an empirical process, performed by collecting and evaluating simulation data in several simulations. Only the most relevant configurations (i.e. in particular, those leading to good recall values) have been reported in this thesis.

K is the coefficient used in the definition of the hyperbolic function, described in Subsection 5.4.2. Other configurations, different from the one listed above, are explicitly mentioned in the text when used.

5.6.1 Scenario 1: Service Discovery

One peer submits a query for a specific goal; each peer hosts one service in its repository and there is only one service (published in some peer's local repository) in the network solving the query.

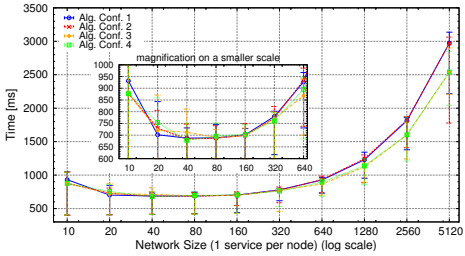
In the following, performance indexes are presented as 90^{th} percentiles of the values observed over 100 simulation cycles, executed with different allocations of services to peers and always submitting a query for the same goal. This scenario has been useful for verifying functional correctness of our distributed discovery algorithm (for single service discovery only) and testing different weights configurations, used in the evaluation of the propagation threshold τ.

Since the network contains only one simple solution to the query and no partial solution is present, the composition strategy does not apply to this scenario.

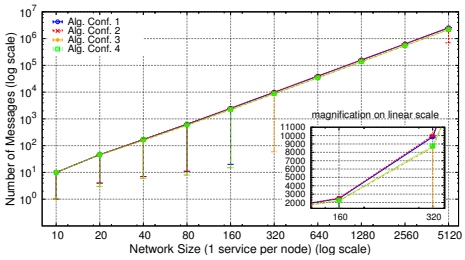
Together with the peer issuing the query for the goal to be solved, we have also considered a number of other peers concurrently querying the system for some goals that cannot be satisfied. This way, noise is introduced in the network to have a more realistic simulation (simulated elaboration delays are increased according to the number of concurrent activities on the peer performing simulated matchmaking). In our experiments, the number of peers producing noise increases with the logarithm of the network size.

Fig. 5.8 is related to topology Δ. Figures 5.8a and 5.8b show resolution time and message overhead when using four different weight configurations (Alg. Conf. 1 to 4). The network is denser when considering larger sizes of the network (degree per node is *network_size/10*). It is worth to note that configurations with a larger weight for the density threshold (Alg. Conf. 3 and 4) tend to exhibit lower resolution time and number of exchanged messages (90^{th} percentiles) than the other ones (Alg. Conf. 1 and 2), since they allow for a larger reduction of messages among neighbors during propagation. Alg. Conf. 3 produces a lower number of messages than Alg. Conf. 4 (see magnification in Fig. 5.8b), especially on large networks, since it has a smaller K coefficient for the density hyperbole.

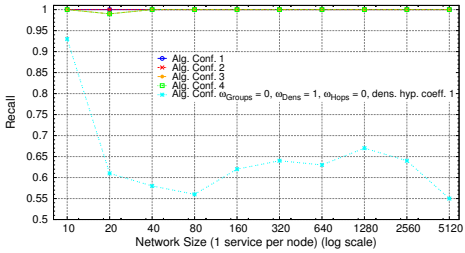
Recall (Fig. 5.8c) is always to the highest possible value 1). It is always reached with the optimized and simple flooding approaches. The only exception with our probabilistic forwarding is related to small networks when using Alg. Conf. 3 or 4,



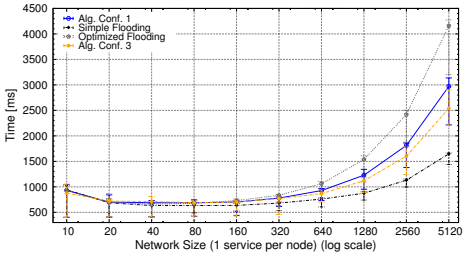
(a) 90^{th} percentile resolution time



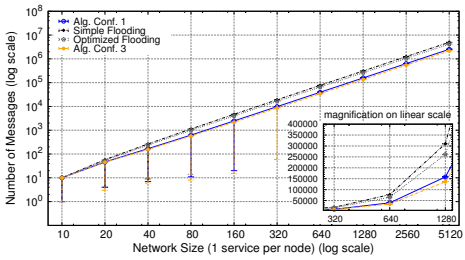
(b) 90^{th} percentile exchanged messages



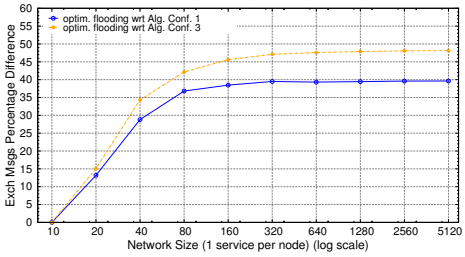
(c) recall



(d) Comparison of 90^{th} percentile resolution time with flooding and optimized flooding



(e) Comparison of 90^{th} percentile exchanged messages with flooding and optimized flooding



(f) Percentage difference of 90^{th} percentile exchanged messages between opt. flooding and probabilistic forwarding

Figure 5.8: Topology Δ, P2P service discovery: performance evaluation

where only a couple of solutions were not found (on a number of 100 experiments). This is essentially due to the reduced density of the network when considering small sizes (for a 20-nodes network there are only two links per node) and the larger weight on the density threshold. In the figure, values related to the algorithm configuration $\{\omega_{Groups} = 0, \omega_{Density} = 1$ and $\omega_{Hops} = 0$, K = 1$\}$ are also reported, showing very poor recall values due to its excessive selectiveness.

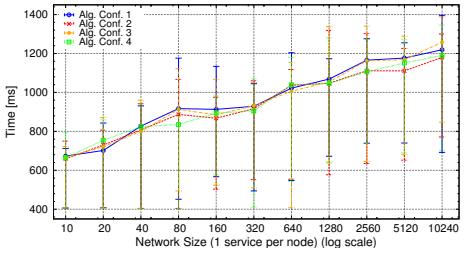
In Fig. 5.8d, a detailed comparison with flooding (both optimized and simple) is

shown for resolution times when using configurations 1 and 3. The figure confirms that, when compared to optimized flooding, the probabilistic algorithm presents lower resolution times (especially on large size network). In addition, Alg. Conf. 3 tends to be quicker than Alg. Conf. 1. When using simple flooding, resolution time can be lower because the larger number of messages exchanged in the network allows for quickly reaching the peer hosting the solution. Even though this may represent a benefit, the larger message overhead has to be taken into account, since, in a real system, limitations on link bandwidth, saturation of queues at routers and hosts (not modeled in the simulated environment) may significantly reduce performances when the number of messages in the network is high.

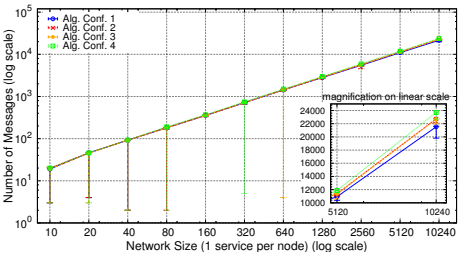
Figures 5.8e and 5.8f reports on the advantages coming from our probabilistic forwarding approach, by showing a comparison with optimized flooding in relation to the number of exchanged messages. In Fig. 5.8e, the 90^{th} percentiles of exchanged messages in the simulation cycles are presented on a logarithmic scale. The same information is described, as percentage difference between optimized flooding and our probabilistic algorithm (both in Alg. Conf. 1 and Alg. Conf. 3), in Fig. 5.8f. It is worth to underline the 35% to 50% gain that can be reached with the probabilistic approach, even when performing simple discovery on middle or large-size networks.

Fig. 5.9 is related to the evaluation of scenario 1 in topology Σ (i.e. average degree per node is between 2 and 4). There are no appreciable differences among the considered configurations in terms of resolution times (Fig. 5.9a), while Alg. Conf. 1, 2 and 3 present a lower message overhead than Alg. Conf. 4 (see magnification in Fig. 5.9b). Recall (Fig. 5.9c) is always in the range [0.9, 1] for all the algorithm configurations considered (Alg. Conf. 1 to Alg. Conf. 4), with a small linear decrease when the network size increases (together with some oscillations due to randomness in simulations). Alg. Configurations 3 and 4, which have the highest message overhead curve in the Fig. 5.9b, exhibit the highest and most stable recall values with different network sizes. It is worth to note that, differently from the case of topology Δ, where the number of links per node increases with the network size, in topology Σ, the network is sparser on larger networks. This is the main reason for the small reduction of recall that can be observed with a size between 1280 and 10240 nodes.

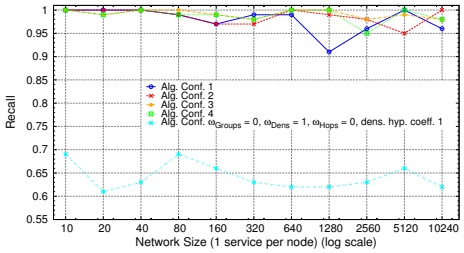
In Fig. 5.9d, resolution times for Alg. Conf. 1 and 3 are compared with flooding and optimized flooding. Their values are very similar, but on larger networks, as in the case of topology Δ, it is theoretically possible to achieve, at the cost of more messages injected in the network, lower resolution times with flooding and optimized flooding, since they maximize concurrency in the network. However, problems related



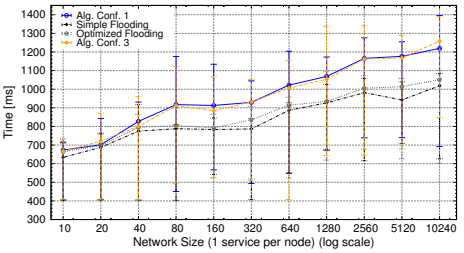
(a) 90^{th} percentile resolution time



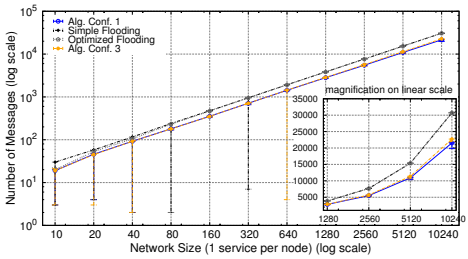
(b) 90^{th} percentile exchanged messages



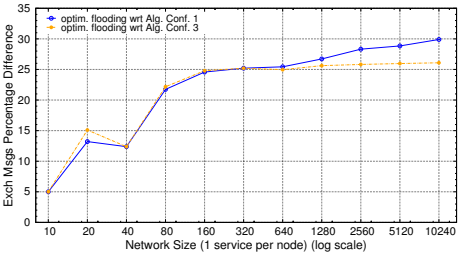
(c) recall



(d) Comparison of 90^{th} percentile resolution time with flooding and optimized flooding



(e) Comparison of 90^{th} percentile exchanged messages with flooding and optimized flooding



(f) Percentage difference of 90^{th} percentile exchanged messages between opt. flooding and probabilistic forwarding

Figure 5.9: Topology Σ, P2P service discovery: performance evaluation

to bandwidth and queue saturation may quickly reduce or eliminate this ideal benefit in resolution time, while solutions with a lower number of messages (probabilistic forwarding algorithm) may exhibit better resolution times consequently. Figures 5.9e and 5.9f detail the benefits in terms of exchanged messages when using the probabilistic forwarding algorithm (configurations 1 and 3) instead of simple or optimized flooding.

5.6.2 Scenario 2: Service Composition (10 services)

One peer submits a query for a specific goal; each peer hosts one service in its repository and there is only one composite solution in the network, specifically a chain of 10 services published on different peers of the network.

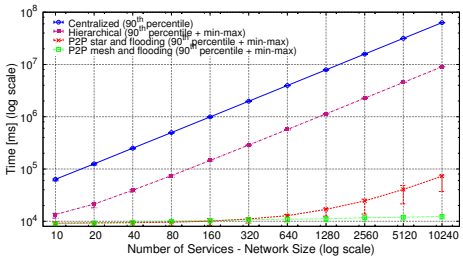
This scenario has been used firstly to the purpose of comparing P2P configurations of the service repository with centralized and hierarchical ones, in order to evaluate the potential benefits of the P2P approach over them (Fig. 5.10).

In the centralized configuration, one node contains all the services available in the network and discovers or composes services to satisfy requests coming from the other nodes, acting like a public registry in the network.

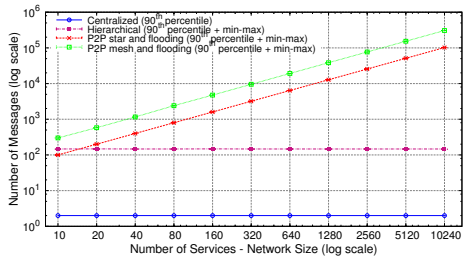
In the hierarchical configuration, services are distributed on a limited (with respect to the network size) number of nodes (7 in our simulations); registry nodes are interconnected according to a hierarchical topology and queries are generated from one node connected as a leaf of the hierarchy.

For the P2P configuration, together with the mesh topology Σ, described before, we have also considered a star topology X of the connectivity graph, where one single node has links to all the other nodes in the network, as in the centralized configuration, while services are distributed among peers. The simulated delays described at the beginning of this section have been used.

In order to perform a meaningful comparison, the same composition algorithm has been used in each of the above configurations (backward resolution). In order to stress benefits and costs for service composition coming from the use of our cooperative P2P approach only, we have disabled network reorganization and adopted traditional flooding for query propagation instead of probabilistic forwarding.



(a) 10-service composition, resolution time (no semantic reorganization)



(b) 10-service composition, messages (no semantic reorganization)

Figure 5.10: Resolution times (left) and number of exchanged messages (right) - comparison of repository architectures

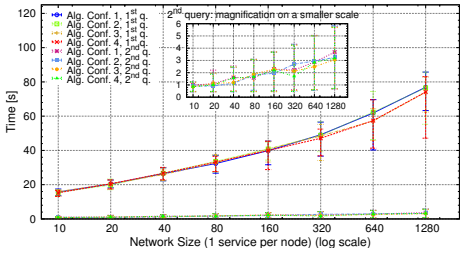
100 cycles of simulation have been executed. In each cycle, the same query has been issued by one submitter and statistical data acquired. Measured performance indexes are presented as 90^{th} percentiles over the different values collected at the end of each simulation cycle, together with min-max confidence intervals.

Fig. 5.10a compares the considered configurations, showing the remarkable improvement that can be obtained, in terms of resolution time, by increasing the degree of distribution of the service repository: the centralized configuration presents the highest values; an improvement can be observed when using a hierarchical registry; P2P configurations offer the best resolution time. It is worth to note that the mesh topology Σ exhibits lower values of resolution time, since there is no bottleneck as in the case of the star topology X. Obviously, the more distribution is introduced in the composition process, the higher is the number of messages to be exchanged to find a solution (Fig. 5.10b).

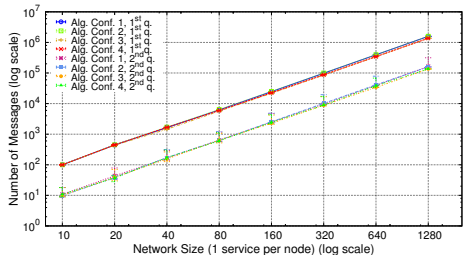
To evaluate the impact of our probabilistic forwarding technique, the P2P backward composition strategy has been used in conjunction with the probabilistic forwarding algorithm. Network reorganization has been activated to build overlay networks. Simulations were performed again in multiple cycles, with one peer issuing a query for the same goal at each cycle. Every two cycles, services were shuffled on the peers. In computing our performance indexes, we have distinguished the average computed over data collected in odd cycles (indicated as 1^{st} query in the following figures) from the one related to even cycles (2^{nd} query). These data are resolution time, number of messages exchanged and the presence or absence of a solution for the query.

The 1^{st} query (the one related to odd cycles) has been submitted when no overlay network has still emerged. Therefore, the only information available to each peer, for efficiently performing query propagation, is the one related to the connectivity graph. If the solution has been found, a corresponding CON is built at the end of the odd cycles, one superpeer emerges and the 2^{nd} query (which refers to the same goal of the first one) can be solved by taking advantage of the overlay network knowledge. After each pair of cycles, the P2P network is brought back to the original configuration: the overlay network is cleared, superpeers are removed, and services are shuffled among peers, introducing a variance element at each pair of cycles to collect statistically more relevant data.

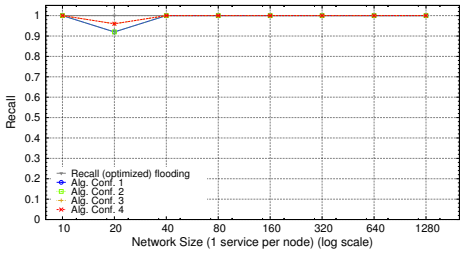
Together with the main query, to be solved by a 10-services composition and issued by one of the network peers, other nodes concurrently query the system for an unsolvable goal, in order to produce noise. In the figures, average values are reported



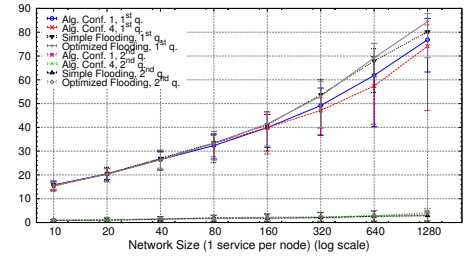
(a) Average resolution time - 1^{st} and 2^{nd} query



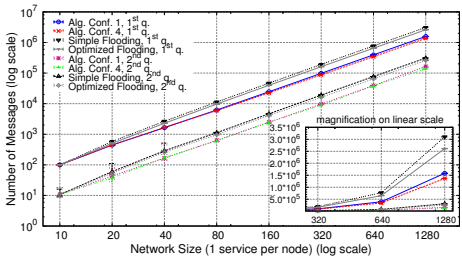
(b) Average exchanged messages - 1^{st} and 2^{nd} query



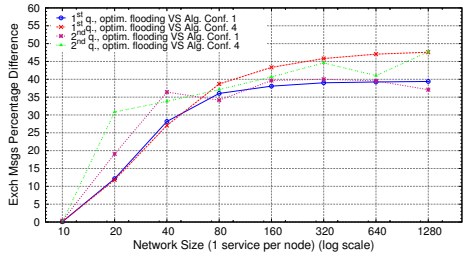
(c) Recall 1^{st} query



(d) Average resolution time - comparison with flooding and optimized flooding



(e) Average exchanged messages - comparison with flooding andoptimized flooding

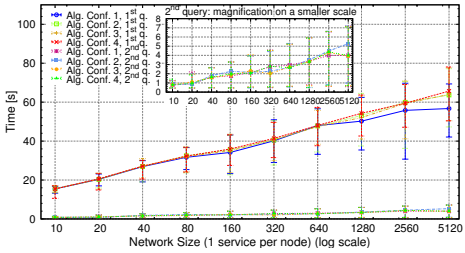


(f) Percentage message difference between opt. flooding and probabilistic forwarding

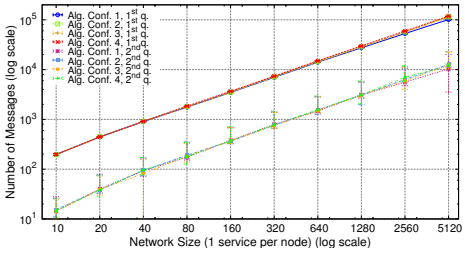
Figure 5.11: Topology Δ, 10-service P2P composition: performance evaluation

together with min-max confidence intervals.

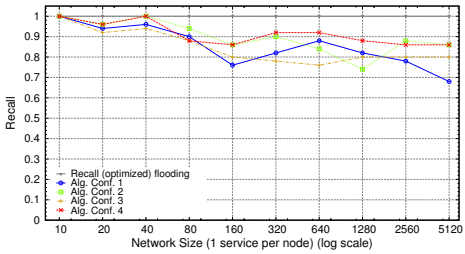
Fig. 5.11 is related to topology Δ and compares the case in which the goal is solved without the overlay network (1^{st} query) to the one in which the overlay network, built from the previous composition, is exploited (2^{nd} query). The curves in the figures confirm that, in topology Δ ($\approx$ *network_size/10* neighbors per node), with any of the algorithm configurations considered (Alg. Conf. from 1 to 4), the service composition is effectively re-used for solving more quickly (lower curves in Fig. 5.11a) an already solved goal (higher curves in Fig. 5.11a). Resolution time for the 2^{nd}



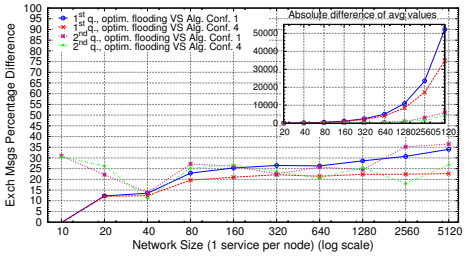
(a) Average resolution time



(b) Average number of exchanged messages



(c) Recall 1^{st} query



(d) Percentage message difference between opt. flooding and probabilistic forwarding

Figure 5.12: Topology Σ, 10-service P2P composition: performance evaluation

query is lowered by the presence, at the superpeer's repository, of the composite service solution previously found for the 1^{st} query.

The presence of the composition overlay network also reduces the number of messages exchanged during the resolution of the 2^{nd} query (Fig. 5.11b), preserving high levels of recall. Fig. 5.11c only shows recall for the 1^{st} query. The 2^{nd} one, submitted in presence of the service composition overlay built after resolution of the 1^{st} query, has always been solved in our experiments (recall is 1, relatively to the number of solutions found for the 2^{nd} query). In Fig. 5.11c, recall is always equal to the highest possible value (i.e. 1, with the exception of size 20, where the network is quite sparse, 2 neighbors per node). Figures 5.11d, 5.11e and 5.11f compare composition based on probabilistic forwarding with composition based on flooding. The curves show the evident benefits of our probabilistic forwarding approach, with respect to flooding and optimized flooding, in relation to resolution time and number of messages exchanged (both for the 1^{st} and for the 2^{nd} query), due to the selective propagation mechanism, described in Section 5.4. Our probabilistic discovery is able to reduce message overhead up to 50% by taking into account network density, semantic grouping and hop traversing, without affecting recall and improving resolution times.

The same network configuration has also been evaluated in the case of topology Σ (Fig. 5.12), obtaining good performance results as in the case of topology Δ.

Figures 5.12a and 5.12b compares the resolution time and the number of messages exchanged before (1^{st} query) and after (2^{nd} query) the construction of the overlay network. Fig. 5.12c shows a higher and more stable recall (close to 0.9 on large networks) when using an algorithm configuration with a higher coefficient for the hyperbole density evaluation function (specifically, Alg. Conf. 4 gives the best result). Fig. 5.12d shows the percentage message reduction with respect to optimized flooding, when using probabilistic forwarding configurations, in case of both absence and presence of the service composition overlay network.

5.6.3 Scenario 3: Service Composition on Overlay Networks (40-services)

Ten *queries for different goals (i.e. A0→A4, A4→A8, . . . , A36→A40) are submitted by different peers in the network; each peer hosts* ***four*** *services in its repository and there is only one composite solution in the network for each requested goal. After the ten queries have been solved and the respective overlay networks built, one peer submits a new query for* ***the whole goal*** *(i.e. A0→A40).*

Fig. 5.13 describes the percentage difference of the number of messages exchanged for solving the query A0→A40 with the optimized flooding approach and the probabilistic forwarding in two algorithm configurations.

As the figure shows, the proposed strategy introduces very positive effects also when large compositions are built by composing the services published on superpeers because of previous compositions.

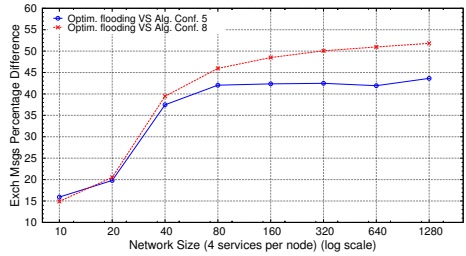


Figure 5.13: Percentage differences of average exchanged messages with optimized flooding and probabilistic forwarding in two configurations – Query A0→A40, 40-services P2P composition on overlay networks, topology Δ

5.6.4 Message Overhead in CONs Building and Management

In the following, we present two experiments aimed at evaluating the performance of the overlay network building/re-structuring process, described in Section 5.3. The analysis is focused on measuring, in a 1000-peers network, the communication overhead related to the overlay mechanisms enacted by service queries submitted and solved by exploiting our P2P composition approach.

Experiment 1: the experiment is organized in two stages. During the first stage, one peer submits a query that is solved by a composition of atomic services distributed over n different network peers; the associated overlay network is built according to the algorithm reported in Listing 5.4, by grouping the n peers that contribute to the service goal solution. In this scenario, we have considered simulations with different values for n (i.e. 10, 20, 40, 80 and 160[6]). In the second stage, one peer issues a second query that is equal to the one submitted during stage 1. This query is solved by exploiting the presence of the overlay network, built at the end of stage 1, and enacts the overlay management protocol (see Listing 5.4). We have measured both the communication overhead required to build the initial composition overlay network and the message overhead required to manage it after the second query has been solved (no need to re-organize, because the overlay did not change).

As shown in Fig. 5.14, the overlay network communication overhead linearly depends on the number of solving peers, both in stage 1 ($2 \cdot n$ messages to collect the previous groups of the solving peers, n messages to create the new group managed by a new superpeer) and during stage 2 ($2 \cdot n$ messages to collect the previous groups of the solving peers).

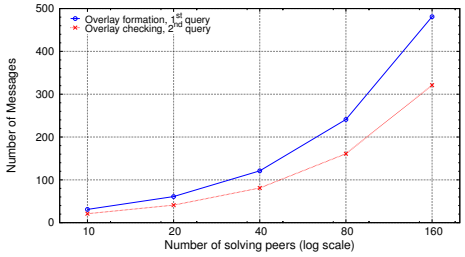


Figure 5.14: Number of messages exchanged for building and checking an overlay network composed of n solving peers

[6]We considered a high number of solving peers (e.g. 80, 160) in order to analyze the message overhead trend, even though it is unlikely to have such a scenario in a real context. However, the reported analysis applies also to an equivalent and realistic situation where multiple smaller solutions are hosted by n different network peers.

Experiment 2: the second experiment is organized in three stages and is inspired by the scenario reported in Section 5.3 (Figures 5.6 and 5.7), used to introduce composition overlay networks. Three different queries are submitted sequentially (one per stage) by one peer of the P2P network. As in Exp. 1, the first query is solved in stage 1 by one composite solution, made up of atomic services distributed over n different network peers; the associated overlay network is built according to our distributed algorithm. The second query is solved in stage 2 by n peers of the network. Differently from Exp. 1, $n/2$ of the solving peers are the same ones that solved the first query, while the remaining $n/2$ peers are different ones, which do not belong to any previous overlay network. Thus, the overlay network built at the end of stage 1 is re-structured by detecting an intersection group of $n/2$ peers and by creating a new group for the remaining different $n/2$ peers. This second group shares with the re-structured one the intersection group's superpeer as one of its children. In the last stage, one peer submits the third query, which is solved by $n/4$ peers already part of the intersection group. Similarly to experiment 1, we have considered multiple simulations with different values for n (i.e. 10, 20, 40, 80 and 160).

Fig. 5.15 reports on the message overhead for overlay (re-)structuring at the end of each phase and the total message overhead (i.e. the sum of the three single contributes). Similarly to experiment 1, the overlay network communication overhead linearly depends on the number of solving peers. Overlay re-structuring at the end of stage 2 exhibits a higher message overhead, due to the need for: (1) retrieving previous groups from the n solving peers; (2) creating two new groups (the intersection and the one for the $n/2$ different peers related to solutions for the second query); (3) demanding the intersection peers to resign from the group created at the end of stage 1.

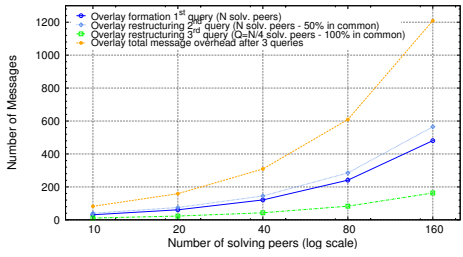


Figure 5.15: Number of messages exchanged for building and re-organizing (with intersections) overlay networks

5.6.5 Robustness Analysis

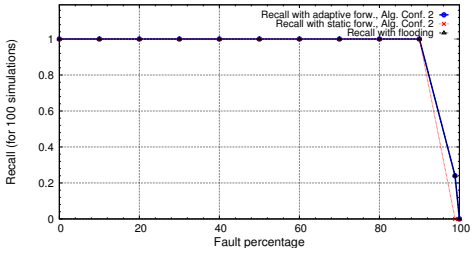
To analyze the robustness of the proposed approach, we focused our attention on three different random mesh topologies, each with the same initial number of peers ($n = 1000$) but with different average network degrees. The first analyzed topology is a strongly connected one (i.e. the Δ topology introduced at the beginning of this section); the second one presents an intermediate level of connectivity (i.e. the Ψ random mesh topology, having ≈ 20 as the average degree); the last one is weakly connected (i.e. the Σ topology introduced at the beginning of this section).

In particular, for each considered topology, we have simulated a periodical increase in the number n of faulty peers ($n = 0\%, 10\%, 20\%, \ldots, 90\%$, up to 98.9%), randomly selected among those peers that do not host pieces of the solution. One simulation cycle is organized as follows: at the beginning, 10 peers are randomly selected in the network to host the pieces of the only available solution to the test query. Then, during the same simulation cycle, the test query is issued periodically by another client peer 11 times: the first time when all the peers are up and reachable, the second one when 10% of them are faulty, the third one when 20% of them are faulty, and so on. In the last query submission, only the client peer and the solving ones are up and reachable.

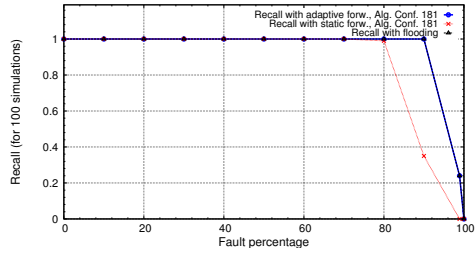
We have performed 100 different simulation cycles, each corresponding to a different random allocation of the solving services to the network peers, and verified whether the solution has been found or not. Recall has been considered as a robustness index and computed as the ratio of the number of cycles in which the solution has been retrieved to the number of simulation cycles (i.e. 100).

In order to verify the robustness of our approach with respect to network dynamics, we have compared our adaptive propagation algorithm, described in Section 5.4, to its static version. In the latter, propagation to neighbors is only limited by the average network degree computed in the initial configuration (i.e. when all the peers are up), without taking into account that faulty peers are not reachable and the corresponding links are down. In other words, we disabled the anti-entropy protocol used to update the mean degree approximation known to the network peers (see Subsection 5.4.2). In addition, we disabled the overlay mechanisms in our simulations, in order to test the effects on robustness coming only from the topology-driven adaptive behavior of the proposed forwarding approach.

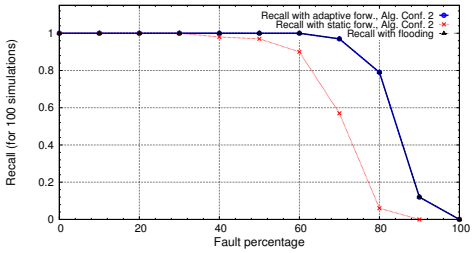
We considered two weight configurations for the propagation algorithm: (1) Configuration 2 of Table 5.2; (2) a weight configuration $\{\omega_{Groups} = 0.1, \omega_{Density} = 0.8$ and $\omega_{Hops} = 0.1$, K = 2$\}$, referred as *Configuration 181* in the following, extremely



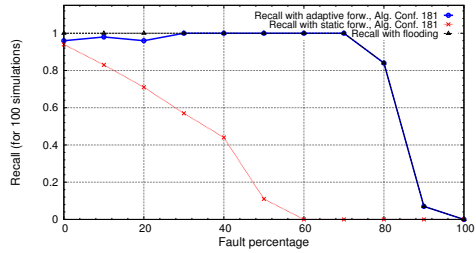
(a) Comparison of recall values for topology Δ using configuration 2



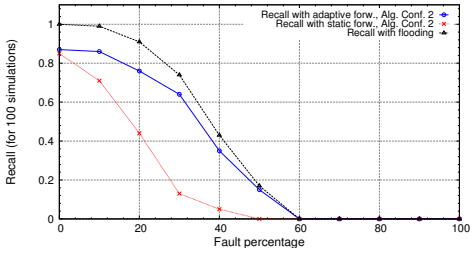
(b) Comparison of recall values for topology Δ using configuration 181



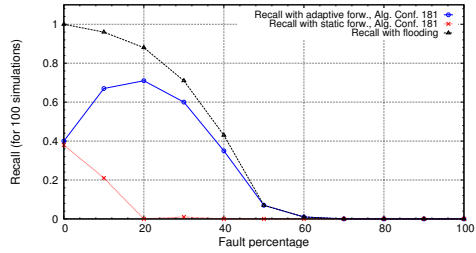
(c) Comparison of recall values for topology Ψ using configuration 2



(d) Comparison of recall values for topology Ψ using configuration 181



(e) Comparison of recall values for topology Σ using configuration 2



(f) Comparison of recall values for topology Σ using configuration 181

Figure 5.16: Comparison of recall values for several topologies of the connectivity graph, when using probabilistic forwarding, static-threshold forwarding and flooding

sensitive to topology dynamics but able to keep high recall levels in a configuration with 100% of the peers up.

Fig. 5.16 shows the benefits (i.e. higher recall) in terms of robustness coming from our probabilistic forwarding approach with respect to the static-threshold forwarding technique. The benefits are more relevant in case of less-connected topologies (Figures 5.16c, 5.16d, 5.16e and 5.16f) and, particularly, when using highly selective weight configurations (e.g. *configuration 181* in Figures 5.16d, and 5.16f).

In most of the performed simulations, our probabilistic forwarding technique allows for recall values that are very close to the maximum recall, reachable with a flooding strategy.

In addition, it is worth to point out that, whenever the number of faulty peers exceeds a certain threshold ($> 90\%$ in Fig. 5.16a and Fig. 5.16b, $> 70\%$ in Figures 5.16c and 5.16d and $> 10\%$ in Figures 5.16e and 5.16f), it becomes highly likely that the sub-graph containing the solving peers becomes disconnected, thus making impossible to find solutions, even with a flooding strategy. In this sense, the recall values related to flooding represent the reference values (i.e. the highest possible recall) for the comparison with the other two forwarding strategies.

5.6.6 Discussion

The evaluation reported in this subsection aims at showing the (self-) scalability of our P2P composition approach according to four main dimensions: memory (i.e. service repository), computation, network (i.e. messages and topology) and composition.

Once chosen the proper settings for weights configuration (Scenario 1), we have demonstrated (Scenario 2 and following) how cooperative P2P composition is effective in working with medium/large-size and dynamically growing service repositories.

Regarding memory scalability, a unique centralized repository easily becomes a limitation. On the other hand, a decentralized P2P model ensures memory scalability by exploiting the resources of the organizations that join the system.

Regarding computation scalability, as shown in scenario 2 - Fig. 5.10a, the centralized approach does not allow for efficient service composition (regarding resolution time) in common scenarios dealing with hundreds or thousands of services published in the central repository. Hierarchical repositories - supported in the last UDDI specification (UDDI v.3.0) or used in DNS - reduce the resolution time to $\mathcal{O}(log(n))$ only for resources identified by structured names. However, this is not the case of semantically described services, where IOPEs' predicates are difficult to structure. Better results can be obtained by relying on an unstructured P2P approach, which can guarantee very low resolution times at the cost of higher communication overhead (Scenario 2 – Fig. 5.10b). These results stem from the inherently distributed nature of P2P architectures, allowing peers to share their physical resources inside the network and obtain computational scalability. Shared resources can be exploited to cooperatively and efficiently carry out complex tasks, like service composition: multiple discovery/merge processes run in parallel over the network peers, thus removing bottlenecks of both centralized and hierarchical solutions (Scenario 2 - Fig. 5.10a).

In addition, the unstructured approach naturally supports a flexible and dynamic interpretation of the SOA roles: each peer may act both as service provider and as service consumer, organizations may manage their services in their own repositories and each peer can collaboratively participate to the service composition process.

Regarding network, flooding over unstructured P2P networks has poor performance in terms of message overhead (complexity is $\mathcal{O}(e)$, since the number of exchanged messages per query is proportional to the number of edges (e) connecting the peers on the connectivity graph). By exploiting information about network density, our selective forwarding mechanism has proved to mitigate such overhead (see scenario 2 – Fig. 5.11b). It reduces the number of exchanged message by up to 50% in dense topologies (see scenario 2 – Fig. 5.11f) and up to 40% in sparse topologies (see scenario 2 – Fig. 5.12d) with respect to the optimized flooding strategy (topology Δ – Figures 5.11e and 5.11f and topology Σ – Fig. 5.12d). This is obtained with slightly better resolution time (topology Δ – Fig. 5.11d) and without affecting the probability of finding solutions when present (see results for topology Δ in Fig. 5.11c and for topology Σ in Fig. 5.12c). In other words, our strategy works like the flooding approach over a less dense virtual connectivity graph, containing only a reduced number of connections among the peers than the actual connectivity graph. The links contained in the virtual graph are sufficient to spread the query effectively and efficiently for finding solutions. Therefore, the message complexity of our solution is $\mathcal{O}(e^*)$, where e^* represents the cardinality of the set of edges in the virtual connectivity graph, obtained by exploiting our probabilistic forwarding technique, and $e^* < e$ (specifically, we observed $e^* \approx e/2$ in the dense topology and $e^* \approx 0.6 \cdot e$ in the sparse one). In other words, we achieve a topology independent message complexity $\mathcal{O}(e^*)$, being $n < e^* < n \cdot (n-1)/2$ and $e^* < e$. In this sense, our strategy autonomously offers network scalability (both topology and message).

The effect of the CONs is shown in scenario 2 (lower curves of Figures 5.11:5.12, related to 2^{nd} query resolution). The resolution time and the number of exchanged messages are significantly lower for the 2^{nd} query than for the 1^{st} one. As soon as one of the peers in the overlay network receives the query message containing the similar goal, it forwards the query to the overlay superpeer, which immediately solves it, without generating a new distributed backward search and a new merging process to compose the same solution again. Therefore, our approach is scalable with respect to the composition size and independent of the connectivity graph as the system evolves, thus achieving composition scalability. No matter how complex the connectivity graph topology is and how the solving peers are far from each other in

terms of number of hops: if overlay networks, built from previously found compositions, exist, it will be extremely quick to solve the new similar ones by exploiting superpeers' knowledge, coded in the CONs and the superpeers' repositories. In other words, the more composite solutions are discovered by the system, the more CONs will be available and the quicker will be to solve new queries, with very low message overhead. Scenario 3 regards longer composite solutions (i.e. 40-service compositions) and is consistent with the previous observations. Even though it is not completely realistic to have a 40-service composition, scenario 3 is important to observe how our approach is effective and efficient in borderline cases.

5.7 More Concurrency: Distributed Bidirectional Search

In some cases, backward search (as well as forward search) is not able to find a solution due to the absence of one or more gap services in the network. In these circumstances, performing composition by searching in both directions could help to find the gap services that are needed to compose the desired service. For example, a partial solution for goal A→Z may range from the goal pre-conditions A to some state X, while another one can range from some state Y to the goal post-conditions Z, being X and Y different from one another and from goal post-conditions Z and pre-conditions A, respectively. In this case, X→Y represents the goal gap to be satisfied in order to have a complete solution for the original goal A→Z.

We introduce *distributed bidirectional search* as the search in the service space that performs concurrently both forward and backward chaining over a network of distributed peers. This search becomes able to find either the desired composite services, if any, or gap services; these are useful for providers to define new business opportunities. This feature can be extremely relevant in social and cloud environments, where new services can be easily deployed by users and new unpredictable business needs emerge frequently. Due to the distributed execution, bidirectional search is also able to significantly improve the time needed for service composition, since forward and backward searches can be performed in parallel.

A realistic example could help to appreciate the benefits of bidirectional search: in a community, people might be interested in flight, hotel and car reservations to organize journeys. Proper services may be available and composed to satisfy requests for travel organization. People might start to be interested in the possibility of sharing their apartments for hosting people who want to come and visit the place they live in

for short periods. A service for managing apartment sharing could still be not available in the community, since no user has already developed such a service. However, people might start to introduce apartment sharing as part of their goals for journey organization, instead of hotel reservation. In such a scenario, the complete goal can only be partially solved with the available services, in relation to the flight and car reservation conditions, while the objective "staying at a place by means of place sharing" would not be satisfied.

The proposed bidirectional strategy is able to retrieve a partial solution (flight reservation + car reservation) and the goal gap, automatically suggesting the need for a service for managing apartment sharing to the users in the community. The consequent benefit for all the parties involved (who needs the service and who provides it) is evident.

The "place sharing" service described before may have some interesting post-condition (e.g. make a donation to the hosting person), which should be accomplished by means of some service still unavailable. In addition, it could be useful to compose services not thought to be used together, which require therefore some intermediate service to be added in order to allow their interaction (e.g. a mediation service). In any case, the bidirectional approach can identify such gaps and be exploited by a framework to propose suggestions in terms of new services to be published in the community.

The following subsections describe our strategy. It has been implemented to the purpose of investigating the potentiality of a bidirectional exploration of the P2P service network both according to the performance perspective of improving response time in retrieving service compositions and according to the functional perspective of identifying goal gaps in partial solutions.

5.7.1 Bidirectional Search Strategy

In the bidirectional search, a service is considered as a partial solution to a goal request when one of the two following conditions holds: 1) service post-conditions semantically match goal post-conditions, but pre-conditions do not match (as in the backward strategy); 2) service pre-conditions semantically match goal pre-conditions, but post-conditions do not match.

Distributed search of composite solutions to a goal can be performed by peers with two concurrent service composition processes: the first one, analyzing available local services for pre-condition matches and eventually generating queries to fill the gap from post-conditions of the matching services to the goal post-conditions (*forward-*

search process); the second one, analyzing local services for post-condition matches and generating queries to fill the gap from goal pre-conditions to the pre-conditions of the matching services (*backward-search process*).

However, in order to take advantage from the two concurrent search processes, they do not proceed separately until a complete solution is found in one direction only. In the backward approach (Section 5.2), whenever a goal can be only partially solved on one peer, a query to fill the gap is issued. That peer will then receive, from the other peers, only complete solutions to the goal gap. These solutions are finally merged with the local partial solution to build the composite solutions to the original goal. Conversely, in the bidirectional approach, partial solutions, which can be found according to the two search processes described above, are immediately sent back to the peer that submitted the query, in order to merge them on the submitter and detect useful compositions as soon as possible.

Fig. 5.17 shows a graphical example of the proposed approach: each phase of the bidirectional search is labeled with a progressive number and, for the sake of simplicity, we assume that the only links available to the peers are those belonging to the connectivity graph.

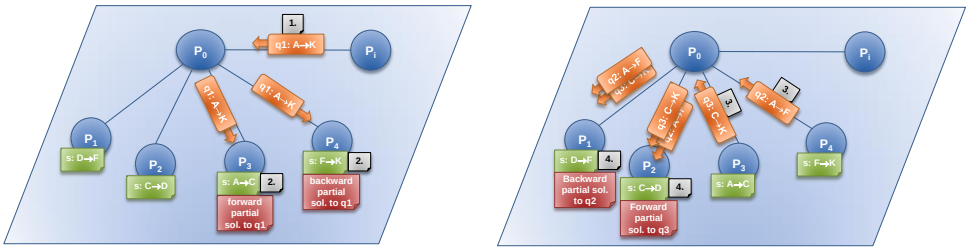


Figure 5.17: An example of distributed bidirectional search

The goal is submitted by peer P_i and spread in the network (*phase 1* - Fig. 5.17a). The implication A→K represents the goal, meaning that K is the desired set of post-conditions to achieve from pre-conditions A, as in the examples of Section 5.3.

None of the peers reached by the query has a local complete solution for it. The distributed bidirectional search, concurrently started by each of them, finds out two local partial solutions in *phase 2* (Fig. 5.17a) on peer P_3 and P_4 (A→C and F→K respectively).

During *phase 3* (Fig. 5.17b), peers P_3 and P_4 reply the partial solutions to the source of the initial query (peer P_i) and generate a new query for the goals C→K

(query q_3 on P_3) and A→F (query q_2 on P_4), respectively. Retrieved partial solutions are replied to the query source with the aim of finding a possible composition satisfying the initial goal A→K (not found in this case and not shown in the picture),

During *phase 4* (Fig. 5.17b), P_2 and P_1 discover partial solutions C→D and D→F to q_3 and q_2 respectively (together with the duplicates F→K on P_4 and A→C on P_3), which are notified back to P_3 and P_4 respectively, the sources of the partially solved queries.

By merging the new partial solution with the one previously found, the composite solution A→C + C→D can be found on P_3, partially solving the A→K original query. The same happens on P_4, which finds the composite solution D→F + F→K. These composite solutions are sent back to P_i, which is able to compose the solution (A→C + C→D) + (D→F + F→K), completely satisfying the goal A→K.

The bidirectional approach, previously described, is performed by means of the algorithm `bidirectionalSolve` in Listing 5.8.

The algorithm is intended to be used in place of the solve algorithm described in Section 5.2. The forwarding procedure is handled through probabilistic forwarding, as described in Section 5.4. The active thread is the same used for the backward approach (i.e. Listing 5.1).

Listing 5.8: Bidirectional search strategy

```
1  bidirectionalSolve(Query q)
2    localSols := localSearch(q);
3    if localSols is not empty
4      reply (q, localSols) to q.source;
5    else
6      localPartialSols := localBidirectionalSearch(q);
7      if localPartialSols is not empty
8        add localPartialSols to partialSolsTable[q];
9        reply (q, localPartialSols) to q.source;
10       foreach partialSol in localPartialSols
11         if partialSol.pre matches q.goal.pre and partialSol.post not matches q.goal.post
12           newGoal := new Goal(partialSol.post, q.goal.post);
13         else if partialSol.post matches q.goal.post and partialSol.pre not matches q.goal.pre
14           newGoal := new Goal(q.goal.pre, partialSol.pre);
15         newQuery.goal := newGoal;
16         newQuery.refQuery := q;
17         newQuery.source := this peer;
18         spawn forward(newQuery);
19         spawn bidirectionalSolve(newQuery);
```

The first part of the algorithm is very similar to the solve algorithm of Section 5.2. When no local complete solution is found (lines 5:19), locally available services are explored for searching partial solutions to the requested goal (line 6), according to the definition of partial solution given at the beginning of this section. Any local partial solution found is stored in a table, local to the peer (`partialSolsTable`), in the entry associated to the query, in order to make possible the subsequent merging process with other partial solutions that will be received from other peers of the network

or found by the peer itself (lines 7,8). As explained in the previous section, partial solutions are notified to the query source peer (line 9), in order to find possible compositions from partial solutions generated from the two concurrent search processes provided by the bidirectional approach.

Finally, for each found partial solution, a new goal request (the *partial goal*) is created and submitted to fill the existing gap between the partial solutions and the goal originally received (the *complete goal*) (lines 10:18). Lines 11,12 and 13,14 distinguish the two cases (pre-conditions match or post-conditions match) characterizing the bidirectional approach. It is worth to note that the partial goal queries store a reference to the originating complete goal query (line 16). This information is used in the passive thread of the modified discovery protocol, shown in Listing 5.9, to understand whether a received result is related to a query explicitly requested on that peer by the active thread or generated to fill the gap of a partial solution to the requested goal. As in the backward approach, the `bidirectionalSolve` procedure is called recursively to find other complete or partial solutions to the new queries on the same peer that generated them (line 19).

Listing 5.9: Bidirectional Discovery Protocol, passive thread

```
1  discoveryProtocol() [passive thread]
2    do forever
3      receive(q, r);
4      if r.isComplete and q.refQuery is nil
5        notify r.solutions;
6      else if r.isComplete and q.refQuery is not nil
7        reply (q.refQuery, r.solutions) to q.refQuery.source;
8      else if q.refQuery is nil
9        mergedSols := merge(r.solutions, partialSolsTable[q]);
10       add r.solutions to partialSolsTable[q];
11       foreach sol in mergedSols
12         if sol.pre matches q.goal.pre and sol.post matches q.goal.post
13           add sol to completeSolutions;
14         else
15           add sol to partialSolsTable[q];
16       notify completeSolutions;
17     else
18       mergedSols:= merge(r.solutions, partialSolsTable[q.refQuery]);
19       foreach sol in mergedSols
20         if (sol.pre matches q.goal.pre and sol.post not matches q.goal.post) or (sol.pre not
               matches q.goal.pre and sol.post matches q.goal.post)
21           add sol to partialSolsTable[q.refQuery];
22       reply (q.refQuery, mergedSols) to q.refQuery.source;
```

The passive thread contains the operations performed by a peer when a partial or complete result to a goal query is received, according to the bidirectional approach. When receiving a result (i.e. a solution to a goal query), a peer has to establish whether the received solutions completely or partially satisfy the goal the result is related to. In the case of a complete solution (`r.isComplete`), there are two cases to distinguish:

1) the received result relates to a complete goal requested on that peer (lines 4,5)

and has to be notified as a composite solution to the peer's collector thread, spawned by the active thread in Listing 5.1;

2) the received result is a complete solution to a query for a partial goal and has to be propagated, as a partial result, to the source peer of the reference query, i.e. the query from which the partial goal request originated, without storing it in the local `partialSolsTable`.

In case of a partial result for a complete goal request, the peer (lines 8:16) has to merge the received partial solution with previously received or found partial solutions, related to the same query and stored in the local `partialSolsTable`. Merged solutions are inspected (lines 11:16) to understand whether they are complete or still partial goal solutions. In the former, they are notified to the peer's collector thread; in the latter, they are stored in the `partialSolsTable`. Partial received results are stored in the `partialSolsTable` (line 10), in turn. Finally (lines 17:22), if a peer receives a partial result to a partial goal query, merging is performed among partial solutions related to the same query (the one identified as `refQuery`). Merged solutions are analyzed and, only if they are still partial solutions, they are stored in the `partialSolsTable` in association to the reference query. Finally, merged solutions are sent back to the reference query source (line 23).

The merge procedure is executed by the peers to compose the newly received partial solutions to the ones previously found. It consists in a trivial exploration of the local `partialSolsTable`, where partial solutions are progressively stored, aimed at sequentially aggregating partial solutions referring to the same goal query, based on the match of post- and pre-conditions. Both the received solutions and the locally available partial solutions may have been found according to the forward or the backward search directions. Hence, merging is possible when:

1. Pre-conditions of previous partial solutions match post-conditions of the new partial result (*backward composition*);

2. Post-conditions of previous partial solutions match pre-conditions of the new partial result (*forward composition*).

When a new merged partial solution is found, it is necessary to verify the possible presence of other partial or complete solutions, by recursively invoking the merge procedure.

Partial Solution Gaps Detection

Bidirectional search makes it possible to detect goal gaps in partial service compositions found when a goal query cannot be completely satisfied. The search algorithm has been designed to leave traces within the `partialSolsTable`, local to each peer, of partial solutions found during the distributed bidirectional resolution of a query. If no complete solution is found, the tables above can be inspected to identify the sub-goals that were not satisfied during query resolution.

The bidirectional search algorithms allow for a progressive propagation of partial solutions to the goal query submitter. Hence, the `partialSolsTable` of the submitter is, among the other ones associated to the network peers, the specific table to be inspected in order to find those state transitions (i.e. sub-goals or solution gaps) required to satisfy the goal. The identified gaps should be notified to the peers in the network to have some of them able to publish a new service or operation satisfying them. This way, the original queries will be completely solved, when submitted again to the system.

It is reasonable to consider periodic inspections of these tables, as well as periodic cleaning (or TTL-based expiration mechanisms) of peers' partial solutions tables. In other words, we can consider that, if the goal has not been satisfied after an adequate amount of time (e.g. a few days), the entries in the table are removed to release resources on that peer.

5.7.2 Scenario 4: Bidirectional Composition (10 services, topology Δ, Perfect Concurrency)

Additional simulations have been performed to verify the collaborative bidirectional composition strategy, used in conjunction with the probabilistic forwarding. To this end, *Scenario 2* has been re-used, considering the topology Δ for the P2P network and the following configurations (Table 5.3) for the forwarding algorithm adopted by the bidirectional search:

Configuration 9:	$\{\omega_{Groups}= 0$, $\omega_{Density} = 0.8$ and $\omega_{Hops} = 0.2$, *density hyp. coeff.* $(K) = 1$ $\}$
Configuration 10:	$\{\omega_{Groups}= 0$, $\omega_{Density} = 0.9$ and $\omega_{Hops} = 0.1$, $K = 1\}$

Table 5.3: Algorithm configurations for the evaluation of P2P bidirectional

We have tested these configurations in Scenario 2 with the backward algorithm. We have observed very low recall, if compared with the one of the bidirectional strategy in the same configurations, as shown in Fig. 5.18. This is due to the extremely

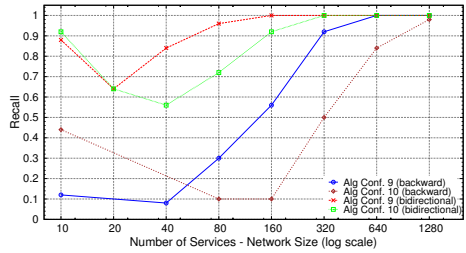
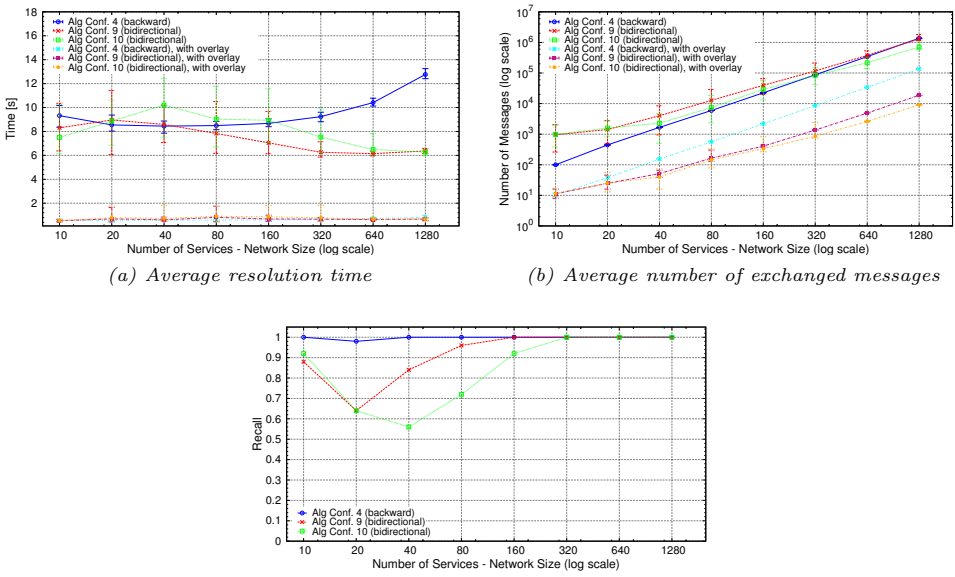


Figure 5.18: Comparison of backward and bidirectional strategies: 10-services P2P composition, topologyΔ, perfect concurrency

selective behavior of the chosen configuration and the absence of the additional forward search mechanism. Therefore, we have used configuration 4 for the backward algorithm in the following comparison with the bidirectional one. Resolution times and messages exchanged are shown in Fig. 5.19.

As Fig. 5.19a shows, the resolution time decreases when network size and density increase (reaching the expected speedup, equal to 2, since two concurrent composition processes execute). On the other hand, the number of messages (Fig. 5.19b) is higher when the network sizes and densities are small or medium, but it converges to the same number obtained with the backward approach when network size is high. The same behavior has been obtained in presence of the CON generated from the previous composite solution (lower curves in Fig. 5.19a and Fig. 5.19b). Fig. 5.19c shows similar effects on recall. When the network size is small, the extreme selective forwarding, driven by the value of the $\omega_{Density}$ coefficient, negatively affects the recall. On the other hand, when the network size is higher, the selective behavior of the forwarding algorithm is positively contrasted by the two query diffusion processes fed by backward and forward searches.

From the results of our experiments, we have also observed that the number of messages could be further reduced by differentiating the ω coefficients for each partial goal query issued by the distributed backward and forward chaining algorithms during the composition. This way, the first query could be forwarded with a limited selective effect in order to ease the diffusion of the query in the network. The next partial goal queries, which could originate from several points in the network, could be forwarded with a more selective behavior, since similar queries are probably replicated in the network. However, this effect is already partially ensured, since, at the beginning of the algorithm, network density is unknown and consequently it is assumed to be very low (see Subsection 5.4.2). This forces the algorithm to initially consider a high value



(a) Average resolution time

(b) Average number of exchanged messages

(c) Recall, 1^{st} query

Figure 5.19: Comparison of backward and bidirectional strategies: 10-services P2P composition, topologyΔ, perfect concurrency

for $\tau_{Density}$ and consequently a high percentage of links to use to forward the query. When the gossip algorithm computes the average network density, $\tau_{Density}$ becomes small for highly dense networks and the number of selected links for query forwarding is strongly reduced.

The same configurations have also been evaluated in the case of topology Σ, observing lower resolution times when using our bidirectional strategy with respect to the backward one, confirming the good results obtained for topology Δ.

5.7.3 Scenario 5: Gap Inspection

Services A→B, B→C, E→F have been published on 3 randomly chosen peers, while service X→Y has been published on the remaining peers. At the beginning of the simulation, the query for goal A→F has been submitted by one peer of the network. The specific goal has no atomic or composite service solution in the service network that can satisfy it.

This scenario has been considered to test gap detection from partial solutions, when the bidirectional search strategy is used and no complete solution is available

in the service network to satisfy the goal query.

Gap services detection has been implemented by defining a new PeerSim Control component, the `SolutionGapsInspector`. This component has the role to retrieve information from the partial solution tables, associated to the peers submitting queries, and to inspect partial solutions in order to find possible incomplete combinations of them. To this purpose, partial solutions found in both the search directions have to be analyzed together and, by semantically comparing them, it is possible to detect the state transitions required for having a complete solution. They represent the gaps to be filled in order to completely satisfy the originally requested goal.

We performed one simulation on a network of 1000 nodes, each with a single peer running on it. At the end of the simulation, no solution has been found as expected. Because of the presence of the two partial solutions A→B + B→C and E→F, the SolutionGapsInspector, started just before the end of the simulation, has correctly detected the following gaps: A→E, C→F and C→E. It is worth to note that the first partial solution (i.e. A→B + B→C) would not have been found by using the backward composition strategy. To reduce the number of retrieved gaps the current implementation of the inspector returns only the gaps related to the longest forward partial solutions (i.e. gap C→F), the longest backward partial solution (i.e. gap A→E) and the longest combined partial solution (i.e. gap C→E). However, the tool can be easily configured to detect other gaps, like B→E or B→F.

5.8 Comparison with Other Gossip Strategies

In Section 5.6 we have evaluated our dynamic gossip-based approach, detailed in Section 5.4, with respect to flooding and optimized flooding, in a simulated environment, considering two different topologies (i.e. a dense one, Δ, and a sparse one, Σ) in several scenarios.

In this section, we compare our proposal to other typical gossip strategies.

We perform the comparison in the PeerSim-simulated environment to observe resolution time, recall and message overhead on large-size and evolving networks while searching for service compositions. We have adopted Bernoulli, Random Geometric and Scale-Free graphs to model different network topologies. The experimental results show that our approach is able to adapt to network changes and preserve high levels of recall. In particular, it reduces message overhead, with respect to both optimized flooding and the analyzed gossip-based strategies, or improves the recall, whereas resolution time remains almost unchanged.

5.8.1 Gossip-based Query Forwarding

Algorithms 1 to 4 describe three well-known gossip protocols (*GossipFF* [28, 33, 54], *GossipPE* [36, 89] and *GossipPB* [14, 41]), commonly used to disseminate information over large-scale networks. These protocols are able to reduce message overhead by limiting the number of neighbors for propagation, but they exploit statically defined thresholds, which usually do not change during execution.

Algorithm 1 describes the general gossip framework for spreading information over large-scale networks. We have used this protocol as an alternative to our probabilistic forwarding to diffuse queries containing the specification of the desired semantic services among the peers of the network.

Algorithm 1 Generic Gossip algorithm

```
1.  procedure BROADCAST(msg)
2.      for all s_j ∈ Λ_i do
3.          Send(msg, s_j);
4.      end for
5.  end procedure

6.  procedure RECEIVE(msg)
7.      if msg ∉ MsgHistory then
8.          Deliver(msg);
9.          msgHistory ← msgHistory ⋃ {msg};
10.         Gossip(msg, parameter);        ▷ Gossip procedure and parameter are decided at configuration time
11.     end if
12. end procedure
```

To start information dissemination, the query source sends a message to all of its neighbors (lines 2 and 3). When one node receives a message, which has not been previously received, it is simply re-transmitted according to the specific gossip strategy, otherwise the message is discarded.

In the following, we consider a large-scale simulated P2P network P_N comprised of N sites s_1, s_2, ..., s_N. Node s_i's neighborhood is denoted as Λ_i and $V_i = |\Lambda_i|$ represents its degree. The `Gossip()` procedure at line 10 may be implemented as [47]: **(1)** *GossipFF*, **(2)** *GossipPE*, and **(3)** *GossipPB*.

All these algorithms receive the message to gossip and one strategy-specific *parameter* (i.e. $fanout$, p_e or p_v), which is used to control the dissemination of the received message and whose value, decided at configuration time, is the same for all the nodes.

In GossipFF (Algorithm 2), node s_i sends its message (msg) to a fixed number (denoted as $fanout$) of randomly selected nodes in Λ_i (lines 6-8). If $fanout \geq V_i$, s_i transmits msg to all of its neighbors (lines 2, 3 and 11, 12). Particularly, if $fanout \geq$

$max\{V_1, V_2, \ldots V_N\}$, Algorithm 2 is a pure flooding algorithm.

Algorithm 2 Fixed Fanout Gossip (at s_i)

```
1.  procedure GossipFF(msg, fanout)
2.      if fanout ≥ V_i then
3.          toSend ← Λ_i;
4.      else
5.          toSend ← ∅;
6.          for f = 1 → fanout do
7.              random select s_j ∈ Λ_i \ toSend;
8.              toSend ← toSend ⋃ s_j;
9.          end for
10.     end if
11.     for all s_j ∈ toSend do
12.         Send(msg, s_j);
13.     end for
14. end procedure
```

In GossipPE (Algorithm 3), site s_i randomly chooses those edges over which msg is transmitted according to a fixed probability p_e (lines 2-4, in which the `Random()` procedure generates a random number in the interval $[0, 1]$). When $p_e = 1$ for all sites, we obtain the flooding algorithm.

Algorithm 3 Probabilistic Edge Gossip (at s_i)

```
1. procedure GossipPE(msg, p_e)
2.     for all s_j ∈ Λ_i do
3.         if Random() ≤ p_e then
4.             Send(msg, s_j);
5.         end if
6.     end for
7. end procedure
```

Unlike Algorithm 3, in GossipPB (Algorithm 4), each site, except the source, diffuses msg to all its neighbors with fixed probability p_v (lines 2-3). In particular, when $p_v = 1$ this protocol becomes the flooding algorithm.

Algorithm 4 Probabilistic Broadcast Gossip (at s_i)

```
1. procedure GossipPB(msg, p_v)
2.     if Random() ≤ p_v then
3.         Broadcast(msg);
4.     end if
5. end procedure
```

5.8.2 Simulation Test Bed

By following the same approach of the evaluation reported in Section 5.6, we have compared our probabilistic forwarding strategy and the gossip algorithms of Subsection 5.8.1 with respect to large P2P networks, by exploiting the PeerSim simulator.

As in Section 5.6, we focused on measuring three performance indexes in our experiments: *Recall*, *Message overhead*, *Resolution time* and we exploited the following delays *Semantic elaboration delay* (i.e. 400 ms for local complete solution; 200 ms for partial backward solution), *Transmission delay (t_d)* (i.e. 1 ms), *Network latency* (i.e. uniform random variable in the range [10ms, 130ms]), in order to make simulations consistent with realistic usage scenarios.

Each node of the simulated network hosts one single peer process (therefore, the terms peer and node will be used interchangeably in the following). Each peer is able to publish, discover and compose services. Service descriptions are published on randomly selected peers before the beginning of each simulation cycle in order to evaluate system behavior with respect to different assignments of the services to the peers. The peers adopt the backward approach for performing cooperative composition.

Several configurations have been considered in simulations to evaluate the performance of the proposed algorithms, both in relation to the parameters of the forwarding algorithms and regarding the topologies used for the connectivity graph.

Configuration of the probabilistic forwarding strategy

The proposed forwarding mechanism on the connectivity graph, described in Subsection 5.4.2, is detailed in the pseudo-code of Algorithm 5. In our comparison, we used a slightly different definition of $\tau_{Density}$, with respect to equation 5.4.

$$\tau_{Density} = \frac{(K/\bar{\delta})}{(\bar{\delta} + (K/\bar{\delta}) - 1)}, \tag{5.6}$$

with $K \geq 1$. The $(K/\bar{\delta})$ factor (*inverse delta factor*) allows for better controlling density-based message reduction. The presence of this factor transforms a simple parametric hyperbole into an auto-tuning one.

In case the locally estimated average density decreases (i.e. the network becomes sparser), the inverse delta factor will allow for a softer reduction of the number of messages with respect to a simple hyperbole, in order not to lose solutions. Conversely, if the network becomes denser, lower values of $\tau_{Density}$ will cause more messages to be cut. K is a dumping factor, tuned according to the specific topological properties of the connectivity graph.

We recall from Subsection 5.4.2 that our gossip-based anti-entropy protocol works only during the inactive phases of our discovery system or by piggybacking regular messages exchanged during query forwarding. Thus, peers only need to access local up-to-date information, without additional message or computational complexity, to

Algorithm 5 Propagation over the connectivity graph (at s_i)

```
1.  procedure PropagateQueryToNeighbors(query)
2.      τ_Groups ← EvaluateGroupsThreshold()
3.      τ_Density ← EvaluateDensityThreshold()
4.      τ_Hops ← EvaluateHopsThreshold()
5.      τ ← ω_Groups * τ_Groups + ω_Density * τ_Density + ω_Hops * τ_Hops;
6.      f ← ⌈τ · λ⌉;                          ▷ λ = number of s_i's neighbors
7.      count ← 0;
8.      i ← 0;
9.      newForwarded ← {};
10.     selectedNeighbors ← ∅;
11.     while i < f ∧ count < λ do
12.         random select s_j ∈ Λ_i \ selectedNeighbors;
13.         selectedNeighbors ← selectedNeighbors ⋃ s_j;
14.         if s_j != query.sender ∧ s_j ∉ query.forwardedPeers then
15.             newForwarded ← {newForwarded|s_j};
16.             i ← i + 1;
17.         end if
18.         count ← count + 1;
19.     end while
20.     query.forwardedPeers ← newForwarded;
21.     for all s_i ∈ newForwarded do
22.         Send(query, s_i);
23.     end for
24. end procedure
```

dynamically compute the value of $\tau_{Density}$ in Algorithm 5. In addition, we assume in this section the absence of composition overlay networks (i.e. τ_{Groups} always equal to 1). Thus, the overall τ computation in Algorithm 5 requires negligible time and no additional message overhead, if compared with the other gossip strategies (Algorithms 1 to 4).

As reported in Table 5.4, the query propagation mechanism (Algorithm 5) has been configured in order to give relevance to the knowledge about density available in the P2P network.

Query Propagation Parameters	
ω_{Groups}	0.1
$\omega_{Density}$	0.8
ω_{Hops}	0.1
K	$[70, 100]$

Table 5.4: Parameters for the probabilistic forwarding algorithm

This is due to our intention of evaluating our forwarding algorithm with respect to other gossip strategies only by considering the topological properties of the network. To this purpose, we strongly limited the effects of the mechanisms for caching already found solutions, based on peer grouping, and for stopping query propagation based on traversed hops counting.

Regarding the K dumping factor in $\tau_{Density}$ evaluation, values in the range [70,

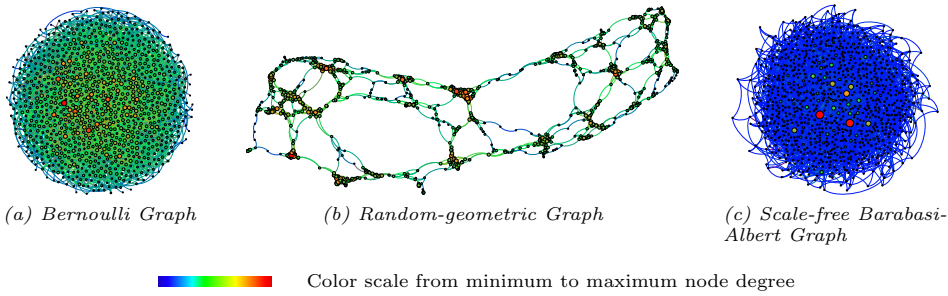
100] showed good performance in our simulations. The results reported in Subsection 5.8.3 are related to the mean value 85.

Connectivity graph configuration

The P2P network is initialized with a specific number of nodes and a connectivity graph.

To perform our simulations we considered three topologies: Bernoulli (or Erdős-Rényi) $B(N, p_N)$ [27], Random Geometric $G(N, \rho)$ [75], and Scale-Free graphs $S(N, m)$ [2], which are typically used to model peer-to-peer systems [54], wireless sensor networks [41] and ad hoc networks [36], respectively.

Figures 5.20a, 5.20b and 5.20c give a visual representation of each different topology.



(a) Bernoulli Graph *(b) Random-geometric Graph* *(c) Scale-free Barabasi-Albert Graph*

Figure 5.20: Examples of different P2P Network topologies: 1000 nodes and average degree per node ≈ 6

They have been obtained by analyzing some of the connectivity graphs used in simulations with the Gephi graph visualization tool [10]. Nodes with degrees larger than the average are shown with larger size and hot colors, while the ones with smaller degrees are presented with smaller size and cold colors. Edges are depicted with the same color of the node they depart from. The figures appear consistent with the theoretical definition of these topologies.

In the case of the Bernoulli graph, most of the nodes present the average degree (green nodes in the figure), while there are few nodes with very high or low degrees. In the random geometric graph, instead, we may observe many high-density zones, containing nodes that are spatially close. Nodes belonging to different areas have few connections to one another because of the large spatial distance among them. Finally, according to a power law distribution, in the Scale-Free graph there are very

few nodes with the highest connectivity (*hubs*), while the majority of the network has the lowest density (*peripheries*).

As in [47], we have considered topologies composed of $N = 1000$ sites. Table 5.5 reports on the values of the parameters used to configure the three connectivity graph topologies.

Topology	**Parameters**
$B(N, p_N)$	$p_N = 0.014$
$G(N, \rho)$	$a = 7500, b = 3000, \rho = 330$
$S(N, m)$	$m_0 = 9$ $(m_0 - clique)$, $m = 7$

Table 5.5: Topology parameters

This configuration is related to the case of a mean degree approximately equals to 14 ($\bar{V} \approx 14$). In the simulations, we have also considered other values for the mean degree and changed the parameters according to the properties of the corresponding topology.

5.8.3 Evaluation

In the following evaluation, we refer to the *main simulation scenario*: one random peer requests the discovery of a specific service goal (i.e. transition from state X to state Y); each peer hosts one service in its repository and there is only one composite solution in the network, specifically, a chain of 10 services published on different peers of the network.

To configure the different gossip algorithms described in Section 5.8.1 and compare them with our probabilistic forwarding strategy, we have used the effectual fanout parameter F_{eff}, defined in [47]. It enables the accurate analysis of the behavior of a gossip algorithm over a topology and simplifies the theoretical comparison of different gossip algorithms on this topology. For a fixed topology and gossip algorithm, the effectual fanout characterizes the mean dissemination power of infected sites. In the following, we report on the definition of F_{eff} for the GossipPE, GossipPB, and GossipFF algorithms.

$$F_{eff} = \begin{cases} p_e \cdot \bar{V} & \text{GossipPE} \\ p_v \cdot \bar{V} & \text{GossipPB} \\ \sum_{k=1}^{fanout-1} P(k) \cdot k + \sum_{k=fanout}^{N-1} P(k) \cdot fanout & \text{GossipFF} \end{cases} \tag{5.7}$$

Given the effectual fanout, we have derived the values of the parameters used for the configuration of the specific gossip strategy (i.e. p_v, p_e and $fanout$), by using the

inverse formulas of the ones in Eq. 5.7. Therefore, in the following results, we refer to a particular gossip strategy by its name and the chosen effectual fanout value.

For each specific topology of the connectivity graph (i.e. one of Bernoulli, Random Geometric or Scale-Free), the network size has been fixed to 1000 nodes, while the average node degree has been increased or decreased according to the specific scenario. Each degree configuration has been simulated over 50 cycles, before changing the graph topology parameters in order to have a new average degree. At each cycle, the services composing the solution have been shuffled over the peers of the network and the same query has been issued by one random submitter. Resolution times are presented as the average over the different values collected at the end of each simulation cycle, together with min-max confidence intervals. Recall is the ratio of the number of found solutions to the number of available solutions (i.e. 50) in the range $[0, 1]$. Percentage difference of exchanged messages has been computed according to the formula:

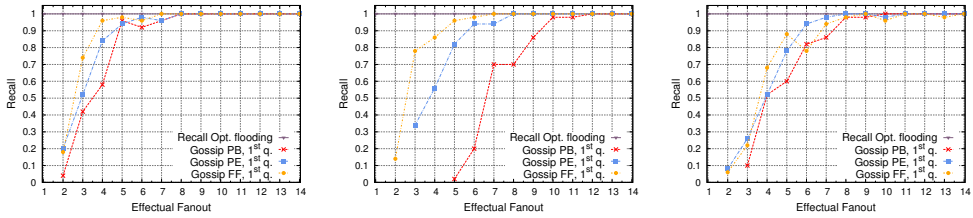
$$\%_{diff} = \frac{M_{\text{dynamic forw.}} - M_{\text{other forw.}}}{(M_{\text{dynamic forw.}} + M_{\text{other forw.}})/2} * 100 \ , \tag{5.8}$$

where M_{X} represents the average number of exchanged messages required to find the solution in the network when using the specified forwarding strategy X.

Effectual fanout for gossip strategies

The first experiment was related to the analysis of the gossip strategies over Bernoulli, Random Geometric and Scale-Free topologies when searching for a 10-service composite solution distributed in the network.

In particular, Fig. 5.21 graphically reports on the recall values observed after 50 simulation cycles performed over the networks of 1000 nodes, with an average node



(a) Bernoulli Graph *(b) Random Geometric Graph* *(c) Scale-Free Graph*

Figure 5.21: Recall of gossip strategies for different topologies as a function of the effectual fanout

degree equal to 14, a reasonable value for large scale networks [47].

The results confirm that GossipFF is particularly good in terms of infection capability on $G(N, \rho)$, but not on $S(N, m)$ (in several cases it has the lowest recall), and that all the algorithms have almost the same infection power on $B(N, p_N)$ [47].

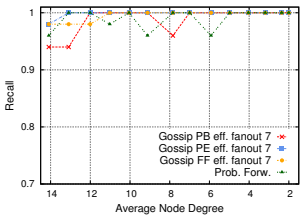
The experiment was also useful to choose the effectual fanout to use in the comparison of the different gossip strategies with our probabilistic forwarding technique, detailed in the next subsections. Since the number of messages increases when using larger effectual fanouts, we decided to consider an effectual fanout equal to 7: this value assures an almost optimal recall (greater than 93% for at least two gossip strategies over the three topologies) at the lowest message overhead.

Simulations over Bernoulli Graph

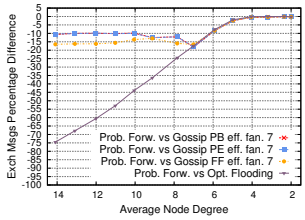
Figures 5.22a, 5.22b and 5.22c graphically reports on the experimental results related to the first variation of the main simulation scenario: starting from an initial average node degree of 14, we progressively decreased it, by selectively removing edges, in order to preserve topological properties, to the smallest possible value to have the graph still connected (i.e. ≈ 2). This scenario simulates a typical situation for P2P networks in which many links disappear because of address changes or congestion.

The probabilistic forwarding strategy presents a stable recall (Fig. 5.22a), higher than 95% for all the average node degrees in the range $[14, 2]$ and is comparable to those related to the gossip strategies having effectual fanout equal to 7. This is due to the ability of our density threshold mechanism to adapt to network changes, by recognizing the need for more neighbors to be forwarded (i.e. by increasing the dynamic fanout), due to density decrease. In fact, the gossip protocol used to compute the average node degree quickly converges to the new lower value. Thus, the peers will use an increased value for $\tau_{Density}$ (see equation 5.6) and select more neighbors for propagation. Therefore, even the partial solutions distributed over less connected areas of the Bernoulli topology can be reached and complete composite solutions can be created from them.

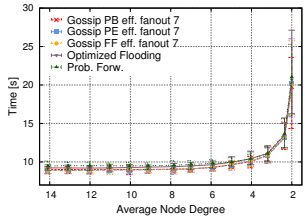
This effect can be better appreciated in Fig. 5.22b, which focuses on percentage difference of the average message overhead between our probabilistic forwarding and optimized flooding or gossip strategies. When the Bernoulli graph is denser, the density threshold cuts up to 20% of the messages produced by the gossip strategies. Message reduction has the largest absolute value when the average degree is equal to 7, where the gossip strategies degenerate into a flooding approach. If the average degree is lower, message reduction decreases in order to not lose solutions, down to



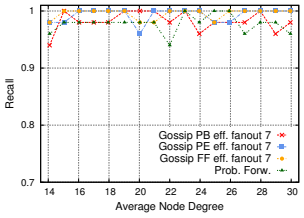
(a) Recall (decreasing degree)



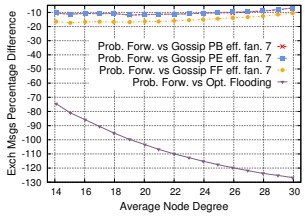
(b) Percentage difference between average exchanged messages (decreasing degree)



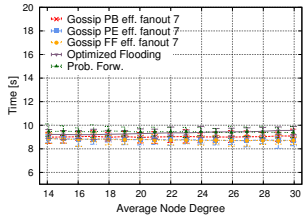
(c) Average Resolution Time, min-max confidence (decreasing degree)



(d) Recall (increasing degree)



(e) Percentage difference between average exchanged messages (increasing degree)



(f) Average Resolution Time, min-max confidence (increasing degree)

Figure 5.22: Simulation results for Bernoulli graph with 1000 nodes: comparison with gossip and optimized flooding

the case in which the flooding approach is the only viable approach to achieve high recall in a very sparse network (average degree is lower than 3).

Regarding resolution time (Fig. 5.22c), the overhead of the probabilistic forwarding is negligible with respect to the other approaches.

Figures 5.22d, 5.22e and 5.22f are related to the second variation of the main simulation scenario: the average node degree of the Bernoulli graph is progressively increased from 14 to 30. This scenario simulates the situation for P2P networks in which new links appear because of neighbor discovery or network decongestion. These simulations aim at verifying the capability of the probabilistic forwarding to maintain its reduced message overhead (Fig. 5.22e) and high recall levels (Fig. 5.22d) even on denser network, where flooding produces a significant growth in the number of exchanged messages (which is proportional to the number of edges in the graph), while gossip strategies with fixed fanout maintain a constant message overhead.

In Fig. 5.22e, the message overhead percentage difference between our probabilistic strategy and optimized flooding increases (in absolute value) as the node average degree increases. With respect to gossip forwarding, it tends to be stable on values

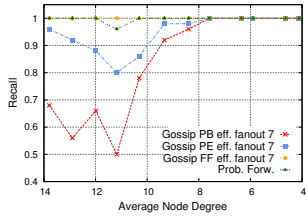
comprised in the range 10-20%, up to very high levels of average degree (i.e. $\approx$ 28). This is due to the asymptotical behavior of the $\tau_{density}$ function, while message overhead for gossip strategies becomes constant after the average node degree becomes larger than the effectual fanout.

As in the previous scenario, resolution time (Fig. 5.22f) is almost unaffected by the choice of the forwarding strategy.

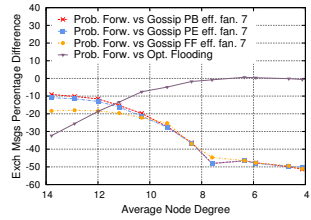
Simulations over Random Geometric Graph

The same scenarios considered in the case of Bernoulli graphs have been evaluated over the Random Geometric topologies (Fig. 5.23).

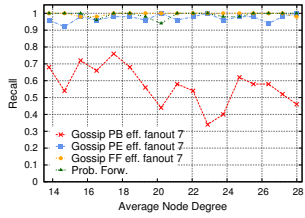
Figures 5.23a and 5.23b are related to the scenario of decreasing average node degree, while Figures 5.23c and 5.23d are related to the one of increasing average. The probabilistic forwarding strategy, as in the case of the Bernoulli topology, is effectively able to autonomously adapt to network density variations by reducing message overhead with respect to the gossip strategies (Figures 5.23b and 5.23d) and preserving high levels of recall (greater than 94% in all the cases, see Figures 5.23a



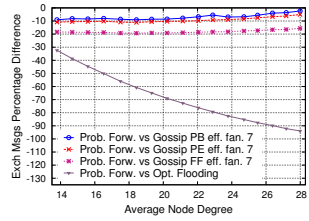
(a) Recall (decreasing degree)



(b) Percentage difference between average exchanged messages (decreasing degree)



(c) Recall (increasing degree)



(d) Percentage difference between average exchanged messages (increasing degree)

Figure 5.23: Simulation results for Random Geometric graph with 1000 nodes: comparison with gossip and optimized flooding

and 5.23c).

An important aspect to point out is related to the selective choice of neighbors, described in Algorithm 5 and characterizing both the probabilistic forwarding and the optimized flooding: the received query is never sent back to the neighbors that have been forwarded at the previous step of propagation. Differently from the Bernoulli graph, the Random Geometric topology provides high-density areas in the network, where groups of neighbors are strongly connected to one another (see Fig. 5.20). Because of this property, by keeping track of the already forwarded neighbors in the query, as in our approach, it is possible to significantly reduce the message overhead both in the case of probabilistic forwarding and when using optimized flooding, with respect to the three different gossip strategies, which are not optimized in this sense.

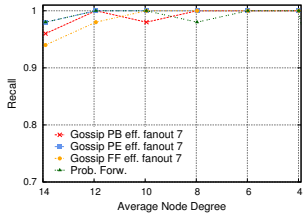
However, as shown by the curves related to the comparison between probabilistic forwarding and optimized flooding, reported in Figures 5.23b and 5.23d, the effect of this optimization is mainly evident in the case of lower average degree. In fact, when the number of neighbors per node is relatively small, a larger overlap exists between previously forwarded nodes and new neighbors to be forwarded.

Resolution times are comparable for the different forwarding strategies and their values increase in the range $[10s, 40s]$, in the case of decreasing degree, being more stable in the range $[10s, 20s]$, in the case of increasing degree.

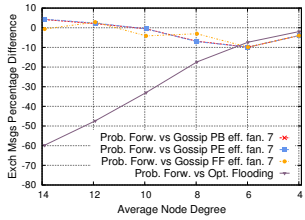
Simulations over Scale-Free Graph

In the case of a Scale-Free graph, using an effectual fanout equal to 7 makes gossip strategies less effective, with respect to the previous topologies, especially if the average node degree is higher than 8 (Figures 5.24a and 5.24c). This arises from the peculiarities of the scale-free topology: very few nodes (*hubs* - red vertices in Fig. 5.20c) have the largest number of links in the network and most nodes have very few connections (*peripheries*). By cutting the number of forwarded neighbors at the network hubs, it is possible that entire zones of the networks are never reached by the goal query, while, if the message reduction is applied at a peripheral site, it may happen that hubs are not forwarded and the query never spreads outside the neighborhood where it has been issued. Therefore, it may easily happen that solutions are not discovered. This problem is even more relevant in the case of resource aggregation (i.e. service composition), where multiple queries are issued by progressively discovering partial solutions.

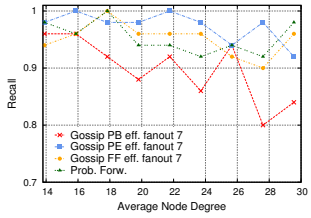
In the worst cases, our strategy is at least as effective as GossipFF forwarding in both the evaluation scenarios; also, in many cases it exhibits higher recall levels than



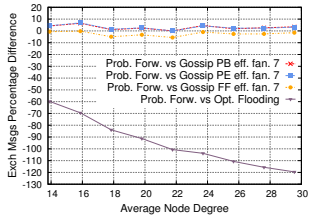
(a) Recall (decreasing degree)



(b) Percentage difference between average exchanged messages (decreasing degree)



(c) Recall (increasing degree)



(d) Percentage difference between average exchanged messages (increasing degree)

Figure 5.24: Simulation results for Scale-Free graph with 1000 nodes: comparison with gossip and optimized flooding

all the other strategies, as can be seen in Figures 5.24a and 5.24c.

This is due to the ability of our forwarding mechanism to recognize the need for more messages to be injected in the network, if compared with the Bernoulli or Random Geometric topologies, because many nodes have very low degrees and the anti-entropy gossip protocol executed to compute the average network density slowly converges. This is reflected in the lower (in absolute values) negative percentage differences of Figure 5.24b (regarding average degrees lower than 10) and in some positive values in Figures 5.24b and 5.24d (regarding average degrees greater than 10). The positive values indicates that, in some cases, our forwarding strategy inject more messages than the gossip ones, which are limited by their fixed fanout, allowing for higher levels of recall (i.e. more solutions found).

Resolution times increase with a low exponential trend in the range $[8s, 11s]$, in the case of decreasing degree (from 14 to 4). Our probabilistic forwarding strategy is slower than the gossip ones according to an offset of about $+0.5s$. When average increases from 14 to 30, resolution times are stable in the same range with the same offset of $+0.5s$ for the probabilistic forwarding.

Chapter 6

Conclusion and Perspectives

This thesis has presented several research contributions to the problem of service composition, both in centralized and decentralized environments. In the following, we summarize the major scientific achievements and the related limitations, paving the way for future research work.

6.1 Contributions

In Chapter 3, we have described our tool for automatic composition of OWL-S semantically annotated Web services. Executable compositions are automatically generated by the tool in either WS-BPEL or XPDL languages, referencing concrete WSDL services retrieved from the OWL-S groundings.

OWL-S descriptions of both services and problem are analyzed, converted into PDDL and fed as input to the Graphplan planner PDDL4J. Solution plans are translated into WS-BPEL or XPDL and can be executed by any common business process execution engine (like RiftSaw). An application scenario about the partial handling of hydrogeological disasters has been defined and used for testing the potential of the presented tool. The performance analysis has confirmed the efficiency of the tool, showing execution times in the order of few seconds, in the case of small/medium-size domains.

Instead of optimizing the tool for planning complete workflows, we have chosen to take advantage of it for re-planning (parts of) already defined (autonomic) workflows, when some activities of the original plan are not available and equivalent sub-processes are required to replace them.

Chapter 4 has presented a model to design context-aware services that can be exploited as a flexible domain to automatically generate context-aware compositions by means of a specific tool. It exploits an extension of the problem definition, which includes the context representation, in order to adapt context-aware services to the specific context.

The model has been presented with reference to an example and the final discussion highlights the advantages of the approach in improving the precision of automatic compositions.

In Chapter 5, we have presented and discussed our approach to cooperative and fully distributed service composition, aimed at increasing the scalability of typical centralized composers. An efficient technique for discovery and composition in unstructured P2P service networks has been detailed. It is based on a probabilistic forwarding algorithm (a sort of dynamic gossiping) driven by network knowledge, such as network density, and traces of already discovered service compositions (CONs). The technique aims at reducing the composition time and the messages exchanged during composition, relying on two considerations: if the network is dense, forwarding can be limited to a small number of neighbors; if the network is semi-structured in CONs, forwarding can be directed to the superpeers that may own the desired information.

The conceptual and analytical comparison of our solution to other P2P-based approaches to service discovery and composition, such as DHTs and SONs, demonstrates its benefits in terms of flexibility and performance.

The validation of our approach exploits a simulator to observe resolution time, recall and message overhead on small and large-size P2P networks. The results of our experiments show that, when using a traditional backward search approach, both the resolution time for service composition and the number of messages exchanged are significantly reduced, while keeping almost unchanged the recall, especially when the network is dense. In this sense, our strategy proves to support scalability from four different perspectives: memory (i.e. service repository), computation, network (i.e. messages and topology) and composition.

We have also improved the discovery and composition process by using distributed bidirectional search. The benefit is twofold: first, it is possible to have concurrent searches in a P2P service network in both goal directions (from pre- to post- and from post- to pre-conditions), thus reducing the response time when solutions are present; second, when no complete solution for a goal is present, gaps in partial found solutions can be identified. This way, it is possible to have feedbacks about users' most required unavailable business operations, allowing providers to discover new business

opportunities. The results confirm lower resolution times when using the proposed bidirectional approach and the possibility of inspecting the network for finding gaps in partial found solutions.

Besides the comparison to optimized flooding, we have compared our probabilistic forwarding algorithm to three well-known gossip-based strategies over unstructured P2P networks. Gossip-based techniques reduce message exchange for query propagation by exploiting graph redundancy through a parameter, called fanout. Our (informed) technique for service composition adopts network knowledge to guide routing and reduce query diffusion: when no or limited domain-specific knowledge is present in the network, the knowledge about the topology of the connectivity graph is exploited (e.g. average density) to avoid or limit redundant paths.

In the comparison, two different scenarios have been considered regarding average node degree changes. Bernoulli, Random Geometric and Scale-Free graphs have been adopted to model different kinds of random networks. The experimental results show that our approach can adapt to network changes and preserve high levels of recall, without manual setup of any fanout parameter. In particular, it reduces message overhead, with respect to both optimized flooding and the analyzed gossip-based strategies, or improves the recall, while resolution time remains almost unchanged in all the cases.

6.2 Limitations and Open Research

Despite of its robustness and usefulness, the tool proposed in Chapter 3 for service composition in a centralized repository is still prototypal and further improvements are possible. In particular, we aim at improving the data and control flow generation, by introducing support for finer-grained synchronization mechanisms, like WS-BPEL <*link*>, <*source*> and <*target*> in <*flow*> activities. However, new semantic constructs to semantically specify such synchronization requirements in the OWL-S descriptions need to be investigated.

Moreover, during XPDL or WS-BPEL generation, concrete services may ground same ontological concepts to different concrete data types, generating data mismatches when they are included within the same solution plan and present data dependencies over the differently grounded concepts. A possible solution consists of providing a set of Web service data adapters to the composer, to perform data conversions between different representations: they can be automatically selected and interposed between the mismatching services. Finally, when no semantically exact

solution is available in the service domain, semantically relaxed plans for partial goal satisfaction could be proposed and ranked.

The analysis conducted during the definition of our context model, reported in Section 4.1, has highlighted a relevant problem to address: by exploiting context for problem expansion and rules evaluation before planning, it is possible that valid composite solutions are indirectly excluded during domain construction. In particular, if multiple context adaptations of the same service fit the current context, it should be avoided that selected adaptations prevent valid composite solutions to be retrieved. As an example, let us consider a scenario in which there is enough disk space for downloading a High Definition (HD) media content, the user prefers HD quality, but the service for playing media does not support HD input. In such situation, service domain contextualization should not exclude regular non-HD content retrieval from the service domain. In fact, the planner would be able to find a composite solution for playing regular media files, downloaded on the local media server, even though the user HD preference would not be satisfied. This requires either a relaxed management of context preferences (i.e. service adaptations enacted by user preferences should be added to the service domain together with those that do not take into account preferences) or to perform the contextualized expansion of services during planning (i.e. real-time), thus taking into account the state dependencies directly when context rules are applied.

In distributed bidirectional search, the message overhead of our strategy can be controlled by adequately choosing the algorithm configurations since, in this case, it tends to grow due to query proliferation. In addition, further improvements could be achieved by dynamically exploiting knowledge about service networks to properly change the coefficients used and tune the behavior of our algorithm at run-time. Heuristics can be used to stop the partial goal search in the network before a complete solution is found or the query search timeout expires. The overall strategy may be further optimized with the heuristics proposed in [99, 107, 108].

Concerning the planning approach, we aim at investigating the Open World Assumption (OWA) to increase the flexibility and the recall of the P2P composition system. This would probably require a real-time planner, based on the incremental execution of services to check actual service effects and pre-conditions. We believe that results like those reported in [69] are very promising in relation to planning in the context of Description Logics (DLs) and demonstrate the need for more research to be carried in this field.

We are currently working on improving our forwarding strategy, by taking into

account the topological properties of the peer's neighborhood: if the forwarding peer has a reduced set of neighbors, with respect to the average degree, the number of neighbors to forward should be higher; in case of high density nodes, message reduction should take into account information about past propagation of the query in the neighborhood in order to not cut messages over unexplored paths. We also plan to evaluate our strategy over other topologies (i.e. tree-like structures, rings) and to consider different evaluation scenarios (i.e. changes in network size, multiple overlay networks).

Finally, we plan to investigate graph-based models for the representation of social networks and their processing in order to extract useful information to support proactive search and composition of services (i.e. service recommendations) in modern Web environments. This could be extremely promising for the application of service computing technologies to several domains, especially eGovernment.

Bibliography

[1] Gregory D. Abowd et al. "Towards a Better Understanding of Context and Context-Awareness". In: *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. HUC '99. Karlsruhe, Germany: Springer-Verlag, 1999, pp. 304–307. ISBN: 3-540-66550-1.

[2] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Reviews of modern physics* 74.1 (2002), p. 47.

[3] *Annual International Contest S3 on Semantic Service Selection*. Last accessed on July 2014. URL: http://www-ags.dfki.uni-sb.de/~klusch/s3.

[4] Dionysis Athanasopoulos et al. "CoWSAMI: Interface-aware context gathering in ambient intelligence environments". In: *Pervasive and Mobile Computing* 4.3 (2008), pp. 360 –389. ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2007.12.004.

[5] Dionysis Athanasopoulos et al. "Mining service abstractions: NIER track". In: *Software Engineering, International Conference on* (2011), pp. 944–947. DOI: http://doi.ieeecomputersociety.org/10.1145/1985793.1985954.

[6] Marco Autili, Paola Inverardi, and Massimo Tivoli. "CHOREOS: Large scale choreographies for the future internet". In: *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE. 2014, pp. 391–394.

[7] Roberto Baldoni et al. "TERA: topic-based event routing for peer-to-peer architectures". In: *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*. ACM. 2007, pp. 2–13.

[8] Luciano Baresi et al. "A framework for the deployment of adaptable web service compositions". In: *Service Oriented Computing and Applications* 1.1 (2007), pp. 75–91.

[9] George Baryannis and Dimitris Plexousakis. "Automated Web Service Composition: State of the Art and Research Challenges". In: *ICS-FORTH, Tech. Rep* 409 (2010).

[10] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: An Open Source Software for Exploring and Manipulating Networks". In: *International AAAI Conference on Weblogs and Social Media*. 2009.

[11] Luca Bevilacqua et al. "A tool for automatic generation of ws-bpel compositions from owl-s described services". In: *Software, Knowledge Information, Industrial Management and Applications (SKIMA), 2011 5th International Conference on*. IEEE. 2011, pp. 1–8.

[12] Mario Bisignano, Giuseppe Di Modica, and Orazio Tomarchio. "Jaxson: A semantic P2P overlay network for web service discovery". In: *Services-I, 2009 World Conference on*. IEEE. 2009, pp. 438–445.

[13] Avrim L Blum and Merrick L Furst. "Fast planning through planning graph analysis". In: *Artificial intelligence* 90.1 (1997), pp. 281–300.

[14] Bastian Blywis et al. "Gossip routing in wireless mesh networks". In: *Personal Indoor and Mobile Radio Communications (PIMRC), 2010 IEEE 21st International Symposium on*. IEEE. 2010, pp. 1572–1577.

[15] C. Bolchini et al. "And what can context do for data?" In: *Commun. ACM* 52.11 (Nov. 2009), pp. 136–140. ISSN: 0001-0782. DOI: 10.1145/1592761.1592793. URL: http://doi.acm.org/10.1145/1592761.1592793.

[16] Nicolo M Calcavecchia et al. "DEPAS: a decentralized probabilistic algorithm for auto-scaling". In: *Computing* 94.8-10 (2012), pp. 701–730.

[17] Mark Carman, Luciano Serafini, and Paolo Traverso. "Web service composition as planning". In: *ICAPS 2003 workshop on planning for web services*. 2003, pp. 1636–1642.

[18] Yatin Chawathe et al. "Making gnutella-like p2p systems scalable". In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM. 2003, pp. 407–418.

[19] I.Y.L. Chen, S.J.H. Yang, and Jia Zhang. "Ubiquitous Provision of Context Aware Web Services". In: *Services Computing, 2006. SCC '06. IEEE International Conference on*. 2006, pp. 60 –68. DOI: 10.1109/SCC.2006.110.

[20] Edith Cohen and Scott Shenker. "Replication strategies in unstructured peer-to-peer networks". In: *ACM SIGCOMM Computer Communication Review*. Vol. 32. 4. ACM. 2002, pp. 177–190.

[21] Massimiliano Colombo, Elisabetta Di Nitto, and Marco Mauri. "Scene: A service composition execution environment supporting dynamic changes disciplined through rules". In: *Service-Oriented Computing–ICSOC 2006*. Springer, 2006, pp. 191–202.

[22] Arturo Crespo and Hector Garcia-Molina. "Routing indices for peer-to-peer systems". In: *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*. IEEE. 2002, pp. 23–32.

[23] Arturo Crespo and Hector Garcia-Molina. "Semantic overlay networks for p2p systems". In: *Agents and Peer-to-Peer Computing*. Springer, 2005, pp. 1–13.

[24] Giuseppe Damiano, Ester Giallonardo, and Eugenio Zimeo. "onQoS-ql: A query language for QoS-based service selection and ranking". In: *Service-Oriented Computing-ICSOC 2007 Workshops*. Springer. 2009, pp. 115–127.

[25] Reza Dorrigiv, Alejandro Lopez-Ortiz, and Pawel Pralat. "Search algorithms for unstructured peer-to-peer networks". In: *Local Computer Networks, 2007. LCN 2007. 32nd IEEE Conference on*. IEEE. 2007, pp. 343–352.

[26] Schahram Dustdar and Wolfgang Schreiner. "A survey on web services composition". In: *International journal of web and grid services* 1.1 (2005), pp. 1–30.

[27] Paul Erdos and Alfréd Rényi. "On the evolution of random graphs". In: *Bull. Inst. Internat. Statist* 38.4 (1961), pp. 343–347.

[28] P Th Eugster et al. "Lightweight probabilistic broadcast". In: *ACM Transactions on Computer Systems (TOCS)* 21.4 (2003), pp. 341–374.

[29] Stefano Ferretti and Gabriele D'Angelo. "Multiplayer online games over scale-free networks: a viable solution?" In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST. 2010, p. 5.

[30] George HL Fletcher, Hardik A Sheth, and Katy Börner. "Unstructured peer-to-peer networks: Topological properties and search performance". In: *Agents and Peer-to-Peer Computing*. Springer, 2005, pp. 14–27.

[31] Agostino Forestiero et al. "A scalable architecture for discovery and planning in P2P service networks". In: *Grid Computing*. Springer. 2008, pp. 97–108.

[32] P. Fragopoulou et al. "Self-* and Adaptive Mechanisms for Large Scale Distributed Systems". In: *Grids, P2P and Services Computing*. Springer US, 2010, pp. 147–156. DOI: 10.1007/978-1-4419-6794-7_12.

[33] Davide Frey et al. "Heterogeneous gossip". In: *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag New York, Inc. 2009, p. 3.

[34] Roy Friedman et al. "Gossiping on MANETs: the Beauty and the Beast". In: *ACM SIGOPS Operating Systems Review* 41.5 (2007), pp. 67–74.

[35] Rossano Gaeta and Matteo Sereno. "Generalized probabilistic flooding in unstructured peer-to-peer networks". In: *Parallel and Distributed Systems, IEEE Transactions on* 22.12 (2011), pp. 2055–2062.

[36] Benoit Garbinato, Denis Rochat, and Marco Tomassini. "Impact of scale-free topologies on gossiping in ad hoc networks". In: *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*. IEEE. 2007, pp. 269–272.

[37] Mohamed Gharzouli and Mahmoud Boufaida. "PM4SWS: A P2P Model for Semantic Web Services Discovery and Composition." In: *Journal of Advances in Information Technology* 2.1 (2011).

[38] Ester Giallonardo and Eugenio Zimeo. "More semantics in qos matching". In: *Service-Oriented Computing and Applications, 2007. SOCA'07. IEEE International Conference on*. IEEE. 2007, pp. 163–171.

[39] *Gnutella Protocol Specification v. 0.4*. Last accessed on July 2014. "The Gnutella Developer Forum". URL: http://rfc-gnutella.sourceforge.net/developer/stable/index.html.

[40] Ya Dong Gong et al. "An Efficient Search Scheme in Unstructured Peer-to-Peer Networks". In: *Advanced Materials Research* 211 (2011), pp. 295–299.

[41] Zygmunt J Haas, Joseph Y Halpern, and Li Li. "Gossip-based ad hoc routing". In: *IEEE/ACM Transactions on Networking (ToN)* 14.3 (2006), pp. 479–491.

[42] Hatim Hafiddi et al. "An Aspect Based Pattern for Context-Awareness of Services". In: *International Journal of Computer Science and Network Security* 12.1 (2012), pp. 71–78.

[43] Ourania Hatzi et al. "Semantic awareness in automated web service composition through planning". In: *Artificial Intelligence: Theories, Models and Applications*. Springer, 2010, pp. 123–132.

[44] Feng Hong et al. "ISI: Integration of Search and Incentive Strategy in P2P Systems". In: *Internet Technology and Applications (iTAP), 2011 International Conference on*. IEEE. 2011, pp. 1–4.

[45] Paul Horn. *Autonomic Computing: IBM's Perspective on the State of Information Technology*. Tech. rep. IBM Research, 2001.

[46] Hung-Chang Hsiao and Hong-Wei Su. "On optimizing overlay topologies for search in unstructured peer-to-peer networks". In: *Parallel and Distributed Systems, IEEE Transactions on* 23.5 (2012), pp. 924–935.

[47] Ruijing Hu et al. "Fair Comparison of Gossip Algorithms over Large-Scale Random Topologies". In: *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*. IEEE. 2012, pp. 331–340.

[48] IBM. *An Architectural Blueprint for Autonomic Computing*. Tech. rep. IBM Research, June 2005.

[49] Imad Jawhar and Jie Wu. "A two-level random walk search protocol for peer-to-peer networks". In: *Proc. of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*. 2004, pp. 1–5.

[50] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. "Gossip-based aggregation in large dynamic networks". In: *ACM Transactions on Computer Systems (TOCS)* 23.3 (2005), pp. 219–252.

[51] Song Jiang et al. "Lightflood: Minimizing redundant messages and maximizing scope of peer-to-peer search". In: *Parallel and Distributed Systems, IEEE Transactions on* 19.5 (2008), pp. 601–614.

[52] Yang Kai-Hsiang and Wu Chi-Jen. "AntSearch: An ant search algorithm in unstructured peer-to-peer networks". In: *IEICE Transactions on Communications* 89.9 (2006), pp. 2300–2308.

[53] Vana Kalogeraki, Dimitrios Gunopulos, and Demetrios Zeinalipour-Yazti. "A local search mechanism for peer-to-peer networks". In: *Proceedings of the eleventh international conference on Information and knowledge management*. ACM. 2002, pp. 300–307.

[54] A-M Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. "Probabilistic reliable dissemination in large-scale systems". In: *Parallel and Distributed Systems, IEEE Transactions on* 14.3 (2003), pp. 248–258.

[55] Matthias Klusch, Andreas Gerber, and Marcus Schmidt. "Semantic web service composition planning with OWLS-Xplan". In: *AAAI Fall Symposium on Semantic Web and Agents, USA*. 2005.

[56] Matthias Klusch and Patrick Kapahnke. "iSeM: Approximated reasoning for adaptive hybrid selection of semantic services". In: *The semantic web: Research and applications*. Springer, 2010, pp. 30–44.

[57] Matthias Klusch and Kai-Uwe Renner. "Fast dynamic re-planning of composite OWL-S services". In: *Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*. IEEE Computer Society. 2006, pp. 134–137.

[58] Freddy Lécué, Alain Léger, and Alexandre Delteil. "DL reasoning and AI planning for Web service composition". In: *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*. Vol. 1. IEEE. 2008, pp. 445–453.

[59] Li Li, Dongxi Liu, and Athman Bouguettaya. "Semantic based aspect-oriented programming for context-aware Web service composition". In: *Information Systems* 36.3 (2011), pp. 551 –564. ISSN: 0306-4379. DOI: 10.1016/j.is.2010.06.003.

[60] Zheng Li et al. "Effort-oriented classification matrix of web service composition". In: *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*. IEEE. 2010, pp. 357–362.

[61] Fotis Loukos, Helen D Karatza, and Constandinos X Mavromoustakis. "AntP2PR: An ant intelligence inspired routing scheme for Peer-to-Peer networks". In: *Simulation Modelling Practice and Theory* 19.2 (2011), pp. 649–661.

[62] Eng Keong Lua et al. "A survey and comparison of peer-to-peer overlay network schemes." In: *IEEE Communications Surveys and Tutorials* 7.1-4 (2005), pp. 72–93.

[63] Qin Lv et al. "Search and replication in unstructured peer-to-peer networks". In: *Proceedings of the 16th international conference on Supercomputing*. ACM. 2002, pp. 84–95.

[64] Zakaria Maamar, Djamal Benslimane, and Nanjangud C. Narendra. "What can context do for web services?" In: *Commun. ACM* 49.12 (Dec. 2006), pp. 98–103. ISSN: 0001-0782. DOI: 10.1145/1183236.1183238.

[65] R Manfredi and T Klingberg. *Gnutella 0.6 Specification*. Last accessed on July 2014. URL: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html.

[66] Drew Mcdermott et al. *PDDL - The Planning Domain Definition Language*. Tech. rep. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212.

[67] Sheila McIlraith and Tran Cao Son. "Adapting golog for composition of semantic web services". In: *KR* 2 (2002), pp. 482–493.

[68] Brahim Medjahed, Athman Bouguettaya, and Ahmed K Elmagarmid. "Composing web services on the semantic web". In: *The VLDB Journal* 12.4 (2003), pp. 333–351.

[69] Maja Miličić. "Complexity of planning in action formalisms based on description logics". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Springer. 2007, pp. 408–422.

[70] Sonia Ben Mokhtar et al. "EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support". In: *Journal of Systems and Software* 81.5 (2008). Software Process and Product Measurement, pp. 785 –808. ISSN: 0164-1212. DOI: http://dx.doi.org/10.1016/j.jss.2007.07.030.

[71] Alberto Montresor and Márk Jelasity. "PeerSim: A scalable P2P simulator". In: *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*. IEEE. 2009, pp. 99–100.

[72] Dana S Nau et al. "SHOP2: An HTN planning system". In: *J. Artif. Intell. Res.(JAIR)* 20 (2003), pp. 379–404.

[73] J. Pascoe. "Adding generic contextual capabilities to wearable computers". In: *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*. 1998, pp. 92 –99. DOI: 10.1109/ISWC.1998.729534.

[74] D Pellier. *PDDL4J*. Last accessed on July 2014. 2011. URL: http://sourceforge.net/projects/pdd4j.

[75] Mathew Penrose. *Random geometric graphs*. Vol. 5. Oxford University Press Oxford, 2003.

[76] Minh Phan and Fumio Hattori. "Automatic web service composition using congolog". In: *Distributed Computing Systems Workshops, 2006. ICDCS Workshops 2006. 26th IEEE International Conference on*. IEEE. 2006, pp. 17–17.

[77] Giuseppe Pirrò, Domenico Talia, and Paolo Trunfio. "A DHT-based semantic overlay network for service discovery". In: *Future Generation Computer Systems* 28.4 (2012), pp. 689–707.

[78] Marina Polese, Giancarlo Tretola, and Eugenio Zimeo. "Self-adaptive management of Web processes". In: *Web Systems Evolution (WSE), 2010 12th IEEE International Symposium on*. IEEE. 2010, pp. 33–42.

[79] Jinghai Rao and Xiaomeng Su. "A survey of automated web service composition methods". In: *Semantic Web Services and Web Process Composition*. Springer, 2005, pp. 43–54.

[80] Katharina Rasch et al. "Context-driven personalized service discovery in pervasive environments". In: *World Wide Web* 14 (4 2011), pp. 295–319. ISSN: 1386-145X.

[81] Sylvia Ratnasamy et al. *A scalable content-addressable network*. Vol. 31. 4. ACM, 2001.

[82] Domenico Redavid, Stefano Ferilli, and Floriana Esposito. "P2P support for OWL-S discovery". In: *Proceedings of The Workshop on Mining Complex Patterns*. 2011, pp. 54–65.

[83] John Risson and Tim Moors. "Survey of research towards robust peer-to-peer networks: Search methods". In: *Computer networks* 50.17 (2006), pp. 3485–3521.

[84] Keivan Ronasi et al. "An enhanced random-walk method for content locating in P2P networks". In: *Distributed Computing Systems Workshops, 2007. ICDCSW'07. 27th International Conference on*. IEEE. 2007, pp. 21–21.

[85] Antony Rowstron and Peter Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems". In: *Middleware 2001*. Springer. 2001, pp. 329–350.

[86] Ozgur D Sahin et al. "Spider: P2P-based web service discovery". In: *Service-Oriented Computing-ICSOC 2005*. Springer, 2005, pp. 157–169.

[87] B. Schilit, N. Adams, and R. Want. "Context-Aware Computing Applications". In: *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*. WMCSA '94. Washington, DC, USA: IEEE Computer Society, 1994, pp. 85–90. ISBN: 978-0-7695-3451-0. DOI: 10.1109/WMCSA.1994.16.

[88] Fatemeh Sharifkhani and Mohammad Reza Pakravan. "A review of new advances in resource discovery approaches in unstructured P2P networks". In: *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*. IEEE. 2013, pp. 828–833.

[89] Chien-Chung Shen, Zhuochuan Huang, and Chaiporn Jaikaeo. "Directional broadcast for mobile ad hoc networks with percolation theory". In: *Mobile Computing, IEEE Transactions on* 5.4 (2006), pp. 317–332.

[90] Lenie Sint and Dennis de Champeaux. "An improved bidirectional heuristic search algorithm". In: *Journal of the ACM (JACM)* 24.2 (1977), pp. 177–191.

[91] Evren Sirin et al. "HTN planning for web service composition using SHOP2". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1.4 (2004), pp. 377–396.

[92] Ion Stoica et al. "Chord: A scalable peer-to-peer lookup service for internet applications". In: *ACM SIGCOMM Computer Communication Review*. Vol. 31. 4. ACM. 2001, pp. 149–160.

[93] Clemens Szyperski. "Component Software and the Way Ahead". In: *Foundations of Component-based Systems*. Ed. by Gary T. Leavens and Murali Sitaraman. New York, NY, USA: Cambridge University Press, 2000, pp. 1–20. ISBN: 0-521-77164-1. URL: http://dl.acm.org/citation.cfm?id=336431.336434.

[94] Sabu M Thampi and Chandra K Sekaran. "Survey of search and replication schemes in unstructured P2P Networks." In: *Network Protocols & Algorithms* 2.1 (2010).

[95] Giancarlo Tretola and Eugenio Zimeo. "Autonomic internet-scale workflows". In: *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*. ACM. 2010, pp. 48–56.

[96] Giancarlo Tretola and Eugenio Zimeo. "Structure matching for enhancing UDDI queries results". In: *Service-Oriented Computing and Applications, 2007. SOCA'07. IEEE International Conference on*. IEEE. 2007, pp. 21–28.

[97] Hong-Linh Truong and Schahram Dustdar. "A survey on context-aware web service systems". In: *International Journal of Web Information Systems* 5.1 (2009), pp. 5–31. DOI: 10.1108/17440080910947295.

[98] Dimitrios Tsoumakos and Nick Roussopoulos. "Adaptive probabilistic search for peer-to-peer networks". In: *Peer-to-Peer Computing, 2003.(P2P 2003). Proceedings. Third International Conference on*. IEEE. 2003, pp. 102–109.

[99] Nilesh Ukey et al. "A bidirectional heuristic search technique for web service composition". In: *Computational Science and Its Applications–ICCSA 2010*. Springer, 2010, pp. 309–320.

[100] Deniz Ustebay, Rui Castro, and Michael Rabbat. "Efficient decentralized approximation via selective gossip". In: *Selected Topics in Signal Processing, IEEE Journal of* 5.4 (2011), pp. 805–816.

[101] Hugues Vincent et al. "CHOReOS: scaling choreographies for the internet of the future". In: *Middleware'10 Posters and Demos Track*. ACM. 2010, p. 8.

[102] Chen Wang and Li Xiao. "An effective P2P search scheme to exploit file sharing heterogeneity". In: *Parallel and Distributed Systems, IEEE Transactions on* 18.2 (2007), pp. 145–157.

[103] Stefan Wesner, Theo Dimitrakos, and Keith Jeffrey. "Akogrimo: The Grid goes Mobile". In: *ERCIM News,(59)* (2004), pp. 32–33.

[104] Hua Xiao et al. "An Approach for Context-Aware Service Discovery and Recommendation". In: *Web Services (ICWS), 2010 IEEE International Conference on*. 2010, pp. 163 –170. DOI: 10.1109/ICWS.2010.95.

[105] Beverly Yang and Hector Garcia-Molina. "Improving search in peer-to-peer networks". In: *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*. IEEE. 2002, pp. 5–14.

[106] Demetrios Zeinalipour-Yazti, Vana Kalogeraki, and Dimitrios Gunopulos. "Information retrieval techniques for peer-to-peer networks". In: *Computing in science & engineering* 6.4 (2004), pp. 20–26.

[107] Bo Zhang. "A heuristic bidirectional search algorithm for automatic Web service composition". In: (2010).

[108] Bo Zhang. "A Web service composition method based on Sub Web Service". In: *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*. IEEE. 2011, pp. 438–442.

[109] Haibo Zhao and P. Doshi. "Haley: A Hierarchical Framework for Logical Composition ofWeb Services". In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. 2007, pp. 312–319.

[110] Haibo Zhao and Prashant Doshi. "A hierarchical framework for logical composition of web services". English. In: *Service Oriented Computing and Applications* 3.4 (2009), pp. 285–306. ISSN: 1863-2386. DOI: 10.1007/s11761-009-0052-9.

[111] Yuping Zhao and Chunlin He. "Survey of resource-searching algorithms in structured P2P network". In: *Information Science and Engineering (ICISE), 2010 2nd International Conference on*. IEEE. 2010, pp. 4738–4741.

[112] Zhu Zhengdong et al. "A P2P-based Semantic Web Services Composition Architecture". In: *e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*. IEEE. 2009, pp. 403–408.

[113] Jiehan Zhou et al. "Context-aware pervasive service composition and its implementation". In: *Personal Ubiquitous Comput.* 15.3 (Mar. 2011), pp. 291–303. ISSN: 1617-4909. DOI: 10.1007/s00779-010-0333-5.

[114] Eugenio Zimeo et al. "Cooperative self-composition and discovery of grid services in P2P networks". In: *Parallel Processing Letters* 18.03 (2008), pp. 329–346.

List of Figures

List of Tables