Résumé

Récemment, l'architecture Orientée Services (SOA) devient un aspect important pour l'agilité du systéme et le développement rapide des nouvelles entreprises. SOA permet l'intégration des différentes plateformes et technologies proposées par les différentes entreprises, et apporte un nouveau niveau de modularité qui permet de garantir la qualité des services de bout-en-bout. Les services Web, définies comme composants indépendants de la plate-forme et qui peuvent décrire des applications pour répondre à une seule tâche, sont l'une des approches les plus prometteuses pour la mise en oeuvre du SOA et qui ont récemment reçu beaucoup d'intérêt. L'un des principaux atouts de l'orientation service est la composition, qui consiste à développer des services de niveau supérieur en ré-utilisant des fonctionnalités bien connues fournies par des autres services avec un faible coût et développement rapide du processus. Automatisation de ce processus devient aujourd'hui un des défis les plus intéressants face à SOA. Par conséquent, la composition des services Web est considéré un sujet de recherche majeur dans les derniéres années. Un grand nombre d'approches de composition des services Web ont été proposés dans la littérature. Malgré le grand nombre des efforts de recherche et développement rapide des modéles de composition et des approches qui ont été proposés au cours des derniéres années, deux problémes majeurs concernant le processus actuel de modélisation de la composition nécessitent d'être résolus. Le premier probléme est lié au niveau expert requis pour parvenir à une telle composition. En général, le style procédural typique de la modélisation, inspiré par des paradigmes/processus de workflow ne fournissent pas les abstractions nécessaires, et donc ne parviennent pas à supporter les compositions dynamiques et auto-gérées qui sont capables de s'adapter aux changements continus qui peuvent survenir de façon imprévisible, et inévitablement conduire à des échecs. Un langage comme le Business Process Execution Language pour les services Web (WSBPEL) est clairement un langage d'experts, et l'utiliser pour programmer et spécifiez une composition est un processus long, coûteux et à haut risque. Le deuxiéme probléme de la composition des services actuels concerne leur cycle de vie et leur gestion, également appelé "leur gouvernance". Le défi est de savoir comment parvenir à une gouvernance compléte de la composition permettant son amélioration continue

et dynamique. Les approches traditionnelles se concentrent uniquement sur certaines étapes du processus de cycle de vie et un peu de travail été réalisé pour intégrer ces dimensions en utilisant un formalisme unifié. Selon Gartner, de nombreuses initiatives dédiées à la gestion des processus ne parviennent pas à se libérer du désignés proposés par les architectes des systémes reconnaissants l'ampleur du travail d'intégration afin de rassembler les différents éléments de la fonctionnalité mise en jeu.

Pour réaliser les défis décrits ci-dessus, l'objectif de cette thése peut être résumé en deux points. Tout d'abord, nous visons à fournir un langage de spécification de service, conçu avec une approche déclarative, basée sur la logique et alimentée par des mécanismes de raisonnement pour répondre aux exigences fonctionnelles et non fonctionnelles des utilisateurs et de fournir des modéles trés expressives et qui ne nécessitent pas les spécifier. Ensuite, en utilisant ce langage de spécification déclarative, nous prévoyons de développer un cadre global et bien intégré pour permettre la maîtrise de la complexité et la fiabilité des compositions de services en réalisant une gouvernance compléte de la composition.

Pour réaliser ces objectifs, nous avons proposé un framework de composition basé sur les capacités fonctionnelles et qui prend en charge le cycle de vie complet du processus de composition d'une maniére unifiée et déclarative, réduisant ainsi le temps de développement et les efforts d'intégration et permettant d'auto-recouvrement de la composition des services Web. Basé sur trois étapes qui consistent de l'abstraction, la composition (instanciation), et la surveillance, notre solution offre un moyen facile de préciser les exigences fonctionnelles et non fonctionnelles des services composés d'une maniére précise et déclarative, et guide l'utilisateur à travers le processus de composition tout en permettant la détection et la récupération des violations à la fois, qui peuvent se reproduisent durant la phase de la conception ainsi que la phase de l'exécution à l'aide du concept de "preuve" et de la planification. Les études expérimentales effectuées démontrent l'efficacité et la performance de notre systéme de composition proposé à la fois au moment du désign ainsi qu'au moment d'exécution.

Abstract

[**Context**] Service-Oriented Architecture (SOA) is becoming a key aspect for system agility and quickly developing new businesses. SOA fosters the integration of different technologies and platforms coming from various enterprises, and brings a new level of flexible modularity that is able to guarantee end-to-end quality of service. Web services, defined as platform-independent and self-describing applications to satisfy a single task, are one of the most promising approaches for implementing SOA and have recently received significant interest. One of the main assets of service-orientation is composition, which consists in developing higher-level services by re-using well-known functionality provided by other services in a low-cost and rapid development process. Automation of this process is emerging as one of the most interesting challenges facing SOA today. Consequently, composition has been a major research topic in the past years. A vast number of service composition approaches have been proposed in literature. [**Motivations**] Despite the huge number of research efforts and fast development of composition models and approaches over the last years, two major bottlenecks in the current process of modeling compositions still remain. The first bottleneck is related to the expert level needed in order to achieve such a composition. Typical procedural style of modeling, inspired by workflow/business process paradigms do not provide the required abstractions, and therefore fail to support dynamic, self-managed compositions able to adapt to changes that may happen continuously, unpredictably, and inevitably lead to failures. A language such as the Business Process Execution Language for Web Services (WS-BPEL) is clearly an expert language, and specifying and programming a composition using WSBPEL is a lengthy, costly, and high-risk process. A second bottleneck in current services compositions concerns their life-cycle and their management, also called their governance. The challenge is how to achieve a full governance of the composition allowing its continuous and dynamic improvement. Traditional approaches focus only on some stages of process life-cycle and little work however has been done in integrating these related dimensions using a unified formalism. According to Gartner, many process management initiatives fail to get off the drawing board once systems architects recognize the scale of integration work to bring the different elements of functionality into

play. [**Objectives**] To address the above challenges, the objective of this dissertation is twofold. First, we aim to provide a service specification language, designed with a declarative and logic-based approach and powered by reasoning techniques to handle both functional and non-functional requirements and highly expressive interaction models without over-specifying them. Second, using this declarative specification language, we plan to develop a comprehensive and well-integrated framework to enable the mastery of complexity and dependability of service compositions by achieving a full governance of the composition. [**Contributions**] To realize these objectives, we have proposed a capability driven composition framework that supports the full roundtrip composition life-cycle in a unified and declarative way, thereby reducing development time and integration efforts and allowing for self-healing Web services compositions. Based on the three stages of abstraction, composition, and monitoring, our solution provides an easy way to specify functional and non-functional requirements of composite services in a precise and declarative manner, and guides the user through the composition process while allowing detection and recovery of violations at both design and run time using proofs and planning.

Dedication

This thesis is dedicated especially to the soul of my brother Houssem, to my Parents, to my brother Helmi, to my fiancy Atef and to all persons who made all of this possible, for their endless encouragement and patience. To my Professors who have been my friends, guides and philosophers. To my best friends who have always assisted me and believed that I could do it. To my family who have always stood by me and dealt with all of my absence from many family occasions with a smile.

♡

Acknowledgments

First of all, I thank Allah for giving me strength and ability to complete my PhD Thesis.

I would like to express my gratitude to my co-supervisor Mohsen ROUACHED for his outstanding supervision, advice, support, inspiration, patience, exceptional seriousness and for always being available during the course of this work. Without his kindness and invaluable support, the work presented in this thesis would not have been possible.

My sincere gratitude goes to my supervisor and Professor Mohamed ABID, for his interest in my work and guidance through the development process of this Thesis. I specially like to thank Prof. Ahmed HADJ KACEM for giving me the honor of being the president of my PhD jury.

Furthermore, I deeply Mr. Lotfi BOUZGUENDA, Prof. Erik MANNENS and Mr. Mounir BEN AYED, for accepting the reviewing and the judgment of my PhD thesis.

During this thesis, I had the pleasure of working with several distinguished researchers all over the world. In particular, I would like to thank those whom I have collaborated with as a part of the work described in this thesis. I am very grateful to Ruben VERBORGH for inviting me as a visiting researcher to Semantic Web Unit at Multimedia Lab research group in iMinds Research group of Ghent University, Belgium. I am very grateful for the interest he showed toward this work, his valuable suggestions, discussions and fruitful cooperations. I would like to thank all the friends I have met during my internship in Multimedia Lab.
Thanks to Wassim DERGUECH, who also provided valuable suggestions on my thesis during my internship in Green and Sustainable IT research unit at Insight Center for Data Analytics (DERI), Galway, Ireland. A special thanks to all the people I had the pleasure to meet in Galway.

Finally, I deeply thank all my friends and the people in the CES Research Unit at the National School of Engineers of Sfax for their support and enthusiasm that makes working in CES very stimulating and challenging.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

The demand for software to live in an open world and to evolve continuously as the world evolves is now reaching unprecedented levels of dynamism. Over the past years, a major step of evolution toward this direction has been made possible by the birth of the concepts of services and Service Oriented Computing (SOC) and by the development of technologies and proposed standards to support this emergent paradigm.

SOC is a programming paradigm that relies on services to facilitate the development of dynamic, interoperable, inexpensive and widely distributed applications. An important and distinguishing feature of services [2] is the fact that they are loosely-coupled, allowing to create dynamic business processes that can flexibly adapt to a continuously changing and unpredictable environment [3, 4]. Modules are reusable entities with specific objectives and can be used for orchestrating composite applications based on a service infrastructure [5]. SOC spans a variety of concepts, protocols, and technologies from different disciplines like distributed computing systems, grid computing, computer architectures, software engineering, database systems, programming languages, knowledge representation and security [6].

The key to realize the SOC vision is Software Oriented Architecture (SOA) [6, 7]. SOA is gaining acceptance among academia and industry as a computing paradigm for business and systems integration. The powerful concept of SOA consists on its decoupling between service provision and consumption, which leads to a much more flexibility and cost-effective integration, within and/or across organizational boundaries, than existing workflows or middle-ware systems do. Thus, SOA popularity has been dramatically increased, and its adoption has expanded across various industries, geographies and organization sizes. It is used in more than 90 percent of new mission-critical operational systems and business processes. It has received significant attention of major computer and software companies such as IBM, HP, Microsoft, Intel and SAP, as well as government agencies such as DoD (US Department of Defense) and NASA.

The best known enabler supporting SOA is Web services technologies. Web Services are emerging as the lead implementation of SOA upon the Web. They have added a new level of functionality for service description, publication, discovery, composition and coordination extending the role of the Web from a support of information interaction to a middleware for application integration. They represent a systematic and extensible method of communication for application-to-application interactions, built on top of existing Web protocols and open XML standards.

One of the most interesting and relevant properties of services is the possibility to combine a number of existing services to create a more general composite service. Composing services allows for the definition of increasingly complex applications by progressively combining services at increasing levels of abstraction [8]. It has received much interest to support business-to-business or enterprise application integration. This process can be performed either manually or automatically (or semi automatically in some cases). However, it can occur either in design-time by producing a static composition model, or at run-time, when a particular service will be executed, providing a dynamic composition schema. Furthermore, composition models can be categorized in four complementary categories: An orchestration model describes both the communication actions and the internal actions in which a service engages. A choreography model describes a collaboration between a collection of services in order to achieve a common goal. A Behavioral interface model captures the behavioral aspects of the interactions in which a particular service can engage to achieve a goal. Finally, coordination models involve temporarily grouping a set of service instances following a coordination protocol.

This pattern is not new, however, it poses some new challenges, which have yet to be addressed by current technologies and platforms. Indeed, while the technology for developing basic services and interconnecting them on a point-to-point basis has reached

a certain level of maturity, it still suffer from some shortcomings when it comes to engineering services that engage in complex interactions, which go beyond simple sequences of requests and responses or involve large numbers of candidates. The process of service composition requires an effective development environment to facilitate quick and simple composition of Web services, and is considered as a key challenge to realize the true potential of web services. Such development environment is of utmost importance for Web service composition languages to keep their promises. [8] discussed the importance of a middleware to support composition in terms of abstractions and runtime infrastructure. The core elements of such a middleware, they identify, are a composition model and language to specify the services involved in the composition, a development environment with a graphic user interface to browse components, and a run-time environment to execute the business logic.

Composition of Web services has been active area of research recently [9], [10]. Several efforts have led to the development of platforms and languages to support composition and deployment of services. However, despite this considerable progress, the composition process still poses limitations and challenges which have yet to be addressed by current technologies and tools for Web service composition. What is clearly required is an Integrated Development Environment to ease the process of composition, thus, reducing development time as well as integration efforts [11]. In this context, we propose, in this thesis, to handle the governance of the composition process in an integrated and declarative way to offer a good support for SOA technologies and achieve their promise to provide flexible business models, guarantee end-to-end quality of service and ensure a seamless service management. Declarative languages have a true potential to represent highly expressive interaction models without over-specifying them, allowing vendors to define architectures to support modular, pluggable service-oriented components, each infinitely configurable to match a small organization's business processes and strategy. The reminder of this chapter discusses motivations and problem statement of our work, exposes the main objectives to consider, and introduces our contributions to realize these objectives.

1.1 Motivations and Problem statement

Having a technology that supports the full round-trip process life-cycle allows enterprises to not only model and automate activities and processes, but also to monitor critical aspects of the process, analyze changes, and apply continuous process improvements across the organization. Therefore, a major requirement is to provide an integrated

solution for dealing jointly with adaptation at modeling, deployment and runtime levels at a time. Below, we discuss motivations and difficulties to achieve such solution.

1.1.1 Composition modeling

To realize the promises and values of SOA, it is important to enable rapid and easy service discovery and compositions inexpensively. Unfortunately, mainstream SOC languages, such as BPMN and BPEL, make it quite hard to fulfill such requirement. Specifying a composition needs good knowledge of many specifications such as WSDL, SOAP, UDDI, and WSBPEL, and requires to use languages and concepts built with classical procedural constructors. The drawbacks of procedural approaches are that these approaches do not provide the required abstractions, and in case of complex relationships or protocols, all the possible interactions have to be explicitly enumerated. Thus, they failed to support dynamic and self-managed compositions and could not adapt to changes that can happen continuously, unpredictably, and inevitably lead to failures.

Indeed, a deep analysis of the composition state-of-the-art approaches clearly categorize them in two classes [12]. A first class requires all services to be semantically described with enough details to allow the engine to choose and combine them in the right way to satisfy the users' goals. A second class is more restrictive, as it relies on the service architect to provide an abstract yet detailed enough model of the orchestration, often using languages like BPEL and BPMN, whose structure is considered the main source of problems.

For both classes, the expert level needed in order to achieve a given composition is considered as one of the major shortfalls in the current approaches of modeling compositions. Typical procedural style of modeling, inspired by workflow/business process paradigms imposes service architects to detail every aspect in the control flow among services candidates: from the most general to the most specific ones. Compositions must explicitly define the different routes starting from the initial state and arriving to the final state, and they have to explicitly manage in advance all possible faults and exceptions that can happen at run-time. Specifications are not guaranteed to reflect the actual nature of what is being specified.

A traditional language like BPEL, is completely an expert language, and composition using such language is a lengthy, costly, and high-risk process. This means that only experts can specify and develop BPEL composite services for instance. Even for small scenarios, writing BPEL code is definitely not a trivial task because it has to take into account both external and internal non-determinism, and also all the success/failure

terminal states. Moreover, the messages (of the protocol) have to manage application-related data, security-related data and reliability-related data. As soon as you try to augment the flexibility or the agility, the over specification increases dramatically.

The challenge at this level concerns the languages and tools for specifying services and their interactions with other services: how to express in simple and understandable way the interaction modalities with services of other organizations? How do I express the *what* versus the *how*? How to consider, enforce, validate, and realize governance concerns?

1.1.2 Composition verification

The focus at this level is on the actions to take once composition specifications are given: how can I verify that specifications satisfy some properties which I deem to be important? For example: if a service represents the delivery of a product, do the specifications that I have produced guarantee compensation in the event of failure to deliver? Moreover, what tools are available to facilitate interoperation with other organizations through services? Assuming that it is possible to discover a set of services that I could use to achieve a goal of mine, how can I verify which specific services meet my requirements? Are there tools available to support composition and cooperation with other services? To which extent can I judge how secure and reliable is an entreprise of which I do not know the background? This means that the challenge here relies on the availability of tools that are able to specify requirements, to verify these requirements, to monitor if the execution is compliant to the specification, and to analyze what really happens during the execution of the composition.

Flexible composition of services and processes with non-functional concerns entails the danger that important rules or constraints of the service or process models get violated or overlooked. Today's state-of-the-art in verifying and validating instances of service or process models, however, can hardly cope with the complexity and dynamics of an end-to-end business compliance framework – both at design time and runtime. Furthermore, they are hard to use, especially for non-programmers. In addition, existing formal verification methods are not integrated with the existing service or process models, and hence a semi-automated verification is hard to achieve in an end-to-end business compliance framework.

1.1.3 Composition monitoring

Monitoring deals with the actual execution of the composite service and is responsible for monitoring the execution and recording violation of any requirement of the goal service at runtime. This requires to define novel principles and techniques for cross-layer monitoring of composition processes, which is a challenging task due to the versatility and the dynamicity of a service composition. It is important to monitor and to analyze properties of a dynamic composition, allowing dynamic refactoring of the composition or of the selected services if needed. These properties can be functional constraints (service A must not be executed before end of service B) or non-functional (the time dedicated to the execution of service A and service B is 10 seconds). Monitoring frameworks should also deal with the scalability of the monitoring/analysis process, because it is crucial that the solution will be able to handle a large number of services, interactions, and events. Another challenge is dedicated to the feedback control and analysis of the composition, i.e. the conformance monitoring and analysis. It is therefore mandatory to be able to express constraints and properties to be monitored and analyzed. Example of these constraints are data to be delivered, timing factors, security enforcement,etc. Then, the last challenge is to incorporate the monitoring /analysis process within the execution framework in order to have a more efficient approach when compared to solution based on external components.

When studying traditional approaches for the composition monitoring [13], [14], [15], [16],[17], [18], [19], [20] , [21], [22], we can observe that the run-time monitoring of the composition process which is tightly coupled with the composition process was not well integrated to these approaches, and very few proposals such as [23] and [24] handle it by adding a new layer for the composition monitoring and thus do not provide the important execution time violations feedback to the composition process. One other commun pattern of traditional approaches is that they are highly procedural, which make the possibility to learn from run-time violations and to change the process instance/model at execution time very difficult. Once again, we believe that these proposals are more oriented to low-level analysis (services/components), while we are more interested by business activity monitoring.

1.1.4 Lack of integration

Another problem with current services compositions concerns their life-cycle and their management, also called their governance. The different stages are respectively the initial modeling of the composition (the specification); its deployment into the overall

architecture, and finally analytics feeding back to the designer. According to Gartner, many SOA management initiatives fail to get off the drawing board once systems architects recognize the scale of integration work to bring the different elements of functionality into play. The inflexibility of current proposals is antinomic to the round-trip model, which implies a philosophy of agility and ongoing improvement as a result of the analysis performed in different stages.

Traditional approaches used to focus only on some stages of the composition process life-cycle, which results in a complex model that can be able to handle the above stages. Also, it is not always possible to have a complete transformation between modeling approaches mainly if we consider non-functional requirements into account. Lack of integration presents a major barrier to learn from run-time failures and offer recovery mechanisms.

Integration concerns also the different entities that may interact with the composition process. In this context, an efficient Web service composition approach needs to support different views, each of which may be applicable to a given kind of role. The Service Requester may be interested in specifying his/her requirements that he/she wants to get fulfilled from the new service. The Service Developer, is interested in ways of fulfilling those requirements through service discovery and selection or if needed, service composition. The Deployment Engineer is more concerned about receiving the service in a shape that is deployable on a runtime engine. Administrator, on the other hand, needs to have an overall view to perform functions such as managing registries, managing user roles, setting appropriate access controls and performing rollbacks in case of errors. Each of the above mentioned user roles have different technological and user interface requirements.

To conclude, providing an integrated declarative solution that covers the entire spectrum is considered the core motivation of our work. This framework will offer tools that enable to specify requirements, to verify how far the solution answers these requirements, to monitor if the execution is compliant to the specification, and to analyze what really happens during the execution of the composition.

1.2 Objectives and Contributions

1.2.1 Objectives

To address the above challenges, we aim, in this dissertation, to design and develop an integrated declarative solution to bridge the gap between the process modeling, verification and monitoring and thus allowing for self-healing Web services compositions. More specifically, the objective of our work is twofold.

1. *Declarative composition specification.* We aim to provide a service specification language, designed with a declarative and logic-based approach and powered by reasoning mechanisms to handle functional and non-functional user requirements of the composition process. This language will enable highly expressive interaction models without over-specifying them. Consequently, designers and engineers will be able to define architectures to support modular, pluggable service-oriented components, each infinitely configurable to match a small organization's business processes and strategy.

2. *Full governance of the composition process.* Using this declarative specification language, we plan to develop a comprehensive and well-integrated framework to enable the mastery of complexity and dependability of service compositions by offering tools for specifying, composing, verifying, and monitoring service usage. This framework will achieve a full governance of the composition allowing the continuous and dynamic improvement of the composition to support and encourage the adoption of SOA technologies.

A more general objective is to help in speeding up and increasing technological transfer and innovation and therefore enhancing the competitiveness of small and medium entreprises in the field of service engineering, SOA, and related technologies.

1.2.2 Contributions

To fulfill the above objectives, we propose, in this work, an integrated declarative framework for Web Services Composition Modeling and Engineering as depicted in figure 1.1. Based on the three stages of abstraction, composition, and monitoring, our solution provides an easy way to specify functional and non-functional requirements of composite services in a precise and declarative manner, and guides the user through the composition process while allowing detection of violations at both design and run time. The staged approach is designed keeping in mind the best knowledge engineering practices of

modularity, conciseness, and scalability, while providing a fair amount of control to the composition process. It focuses on the specification of "what" level without having to state the "how", which enables to preserve the autonomous nature of interacting services and represent expressive interaction models without over-specifying them.

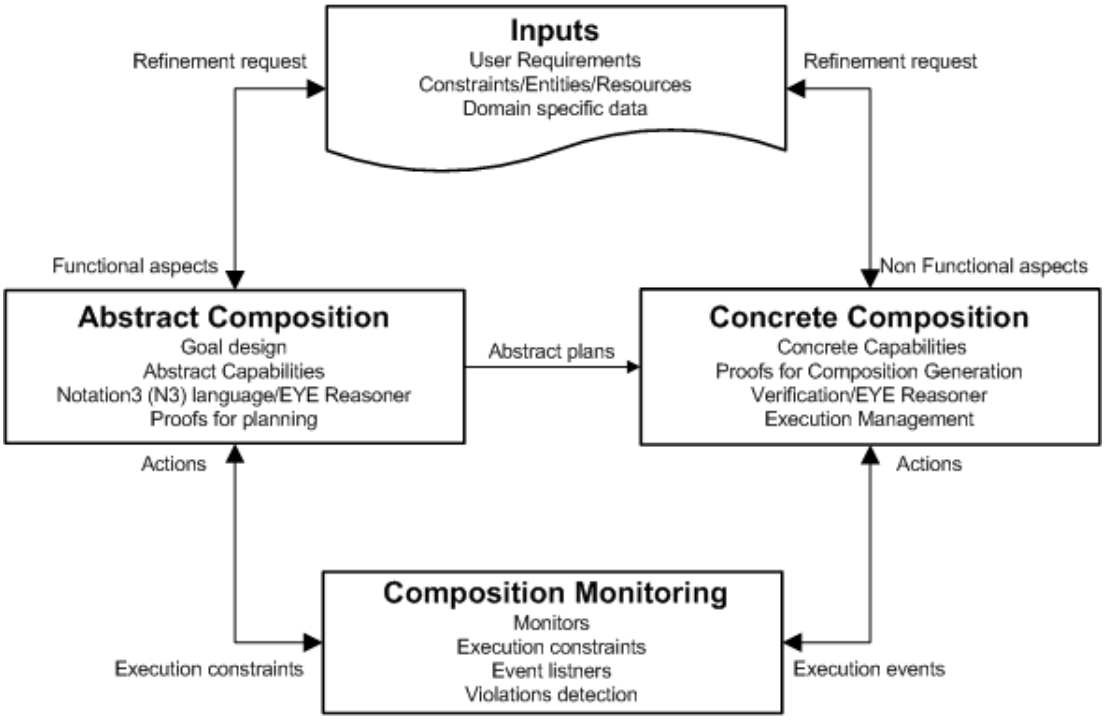


FIGURE 1.1: Framework Architectural Diagram

As shown in the architectural diagram, the proposed framework strives to automate the core phases of the composition process, while leaving scope for valuable user feedback between the phases. The key features of this framework are:

- Users/Requestors provide the high-level description of the service desired (goal) using a user-friendly interface that enables end users to specify and express their requirements and preferences that mark the boundary of the solution (requested service) by following some instructions to build the query. A query parser is integrated to parse and validate the query by checking syntactic correctness and decoupling the functional requirements from the non-functional parameters (such as QoS properties). Because users provide high-level specification of the composite service which may not be realizable using the published component services, our framework guides the users for iterative refinement of the goal service specification.

- A declarative capabilities driven specification that uses Notations3 rule language [25] to specify the components of the composition process and define patterns for specifying the functional and non-functional aspects for process specification. We use capabilities [26, 27] to express compositions, modeling both their functional

and non-functional requirements. A Web service is described as a structured entity featured via a set of capabilities, non-functional features and workflow properties (in case of a composite service). Such description considers a service as an access mechanism to a capability, which is, in its turn, a structured entity that describes what a service can do via an action verb, a set of domain-specific attributes, a set of preconditions and a set of effects. Different services can be interconnected at different levels of abstraction/concreteness by establishing links between them.

- A proof based approach using EYE reasoner [28] for the process design-time verification. The capabilities driven description is used to provide a two staged Web service composition approach, which is purely declarative and support flexible self-managed compositions. First, an abstraction stage consists in constructing a composition of available services that provide the desired functionality by semantically generating a composition plan of abstract capabilities. Second, a concrete stage concretizes the abstract composition into an executable composition by selecting the appropriate concrete capabilities instances based on non functional aspects. Verification will be applied both to the single services and to the whole, dynamically evolving, composition. Proofs obtained through formal verification could be provided and advertised. In this way, other services can reason upon the declarative specifications, and possibly check the advertised proofs of properties, leaving no doubts on the dependability of the services.

- An event-based monitoring framework that allows to reason about the events and does not require defining and extracting events from process specification, as the events are first class objects of both design and monitoring framework. As the proposed monitoring approach builds upon capabilities and N3 based composition design, it allows for the specification of monitoring properties that are based on both functional and non-functional (such as temporal, security or their combinations) requirements. These properties are expressed as N3 formulas and can be added to the process specification both during process design and during the process execution. A continuous monitoring of the on-line behavior of services will provide reliable trust levels and a dependable quality of service.

- Implementation of the above key features in a composition management system that realizes all algorithms and models proposed in the dissertation. This management system presents all the functionalities from design phase to monitoring and provides the corresponding tools for each phase. It was tested and evaluated using realistic scenarios.

To summarize, our work targets dynamic environments where a composition is built through service composition of basic building services. Formal verification, both a-priori

and at run-time, will be an invaluable tool to guarantee dependability of the client on the available services. The stability of this complex system is ensured through a continuous and pro-active monitoring of the interaction, and on-the-fly verification. This full round trip - modeling, verifying, monitoring, and analyzing - will be a very important added value for enterprises using SOAs.

1.3 Thesis outline

The reminder of the thesis is structured as follows.

In **Chapter 2**, we draw a deep and comprehensive review of the state of the art of the current approaches that deal with composition process design, verification and monitoring. Before starting this review, we introduce the notions of SOA and Web services and their related technologies. We discuss the different existing compositions models, and make a systematic study of existing composition approaches depicting their limitations in terms of our objectives.

In **Chapter 3**, we discuss the proposed capabilities based composition design by first presenting the capabilities meta models for different components that form the composition design. Then, based on this specification, we detail the components of the proposed framework.

In **Chapter 4**, we tackle issues related to formalizing our services models using Notation3 and generating compositions using proofs and EYE semantic reasoner.

In **Chapter 5**, we describe the implementation of our approach of composing Web services in an integrated and declarative way and show its applicability in a real use case. We conduct an extensive performance study for our developed approach.

Finally, **Chapter 6** is dedicated to concluding remarks and directions for future research.

Chapter 2

State of the art

Contents

Service-oriented architecture and the supporting Web services technology are becoming more and more involved in everyday sensitive, mission-critical operational applications and business processes. Today, business processes are increasingly implemented by dynamically composing Web services seen as the main contribution the SOAs bring to enterprise business process automation, thus enabling to create complex systems that are interoperable, composable, extensible, and dynamically reconfigurable. Complex dependencies can be created between Web services offered by different organizations using compositional models such as choreography, orchestration, and coordination to facilitate the integration of enterprise applications between businesses and organizations.

In this chapter, we start by briefly discussing notions of SOA, Web services and their related technologies. Then, we concentrate on the Web services composition process as an important feature to realize the objectives of our thesis. The main goal is to understand the key considerations that underlay the specification, the verification, the execution, and the monitoring of Web services compositions. We discuss the different existing compositions models (Orchestration, Choreography, Coordination, and Semantic Web Services) and their related languages. Finally, a systematic study of existing composition approaches depicting their limitations in terms of being procedural, lack of expressivity and lack of integration is made. We consider the following two main approaches: (i) procedural composition approaches, and (ii) declarative composition approaches, and compare them according to whether or not they address the composition global life-cycle.

2.1 SOA and Web services

Most of people often think of SOA and Web services in combination, however, these two concepts are totally distinct in an important manner. SOA is considered as an abstract architectural concept to build software systems based on loosely coupled modules (services) that can be easily discovered and composed. Thereby, the concept of Web service represents one important approach to realizing SOA, built on top of open standards such as HTTP and XML and can be described, published, located and invoked by other programs over the Web. The main feature that distinguishes the Web service from the Websites or applications is that it is designed to be used/consumed by an actual application rather than directly targeting a browser.

2.1.1 Service Oriented Architecture

The demand for software to live in an open world and to evolve continuously as the world evolves, however, is now reaching unprecedented levels of dynamism. Over the past years a major step of evolution toward this direction has been made possible by the birth of the concepts of services and service-oriented architectures and by the development of technologies and proposed standards to support them. SOA is a design style that leads to the all aspects of creating and using services throughout their life-cycle. SOA aims to bring about component reuse, irrespective of implementation language or host platform, and as such it can be thought of as simply an extrapolation of good software engineering practices, taking us from the *class reuse* concept to *service reuse* concept. Thus, SOA typically encompasses the following features:

- *Component architecture*: SOA is based on reusable software components enabling to build scalable heterogeneous (i.e. platform- and language-independent) service architecture.
- *Loose coupling*: The principle of Service Loose Coupling promotes the independent design and evolution of a service's logic and implementation while still guaranteeing baseline interoperability with consumers that have come to rely on the service's capabilities.
- *Platform independence*: This feature has been achieved by the adoption of standards, which have been the key mechanism enabling previously incompatible technologies to work cooperatively across a wide range of different platforms. Single services can interoperate with each other without depending on specific platforms or programming languages.
- *Transparency*: It is ensured by decoupling service functionalities from their actual implementation.
- *Flexibility*: SOA must ensure flexibility so as a system would be able to deal with dynamic changes of its configuration and behavior according to varying requirements.

It results that SOA enables an IT infrastructure to allow different applications to participate in several business processes and exchange data, regardless of the details of the applications, such as the operating systems or the implementation or the programming languages used to implement them.

Service is considered the main concept in SOA. It is the mechanism through which components that provide capabilities (service providers) and components with specific needs

(service requester/consumer) can interact. The interaction with services is regulated by a set of basic methods that allow to provide, discover, use and interact with services in a seamless way. In general, a service is accessed by its interface, which comprises the specifications of how to access the underlying capabilities. Figure 2.1 illustrates the most basic interactions required to interact with a service.

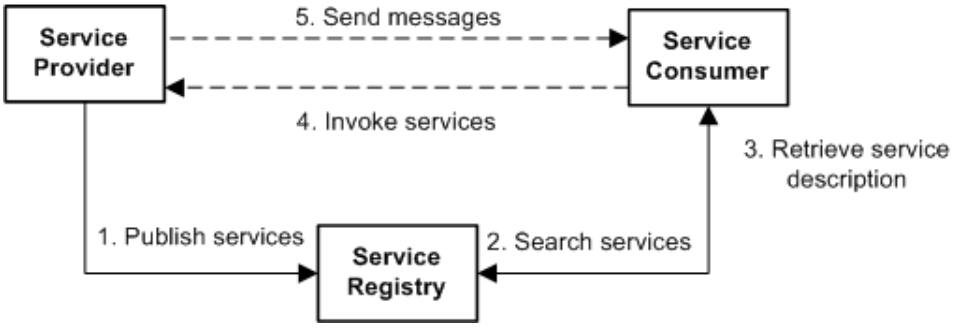


FIGURE 2.1: Service Interaction basic Model

In an ideal scenario, a service provider hosts a network-accessible software module, which defines an implementation of a given service, and offers a service description that leads the service to be published and discoverable. A user discovers the required service and retrieves its related description that will be then used to bind to the provider and invoke the service.

Services are created from scratching or by breaking down or refashioning older applications and existing information. Roughly speaking, a service is a software entity that implements some well defined functionality that can be consumed by clients (e.g. other services), regardless of the application or the business model. Services communicate with each other by means of message exchanges. Notice that the definition of a service provides a high level conceptual understanding of what a service is, not going into implementation and specification details, which are of no importance at this level. In particular it does not make any assumption about the underlying communication framework nor the specifications used for describing services. It only states that a service has some capability and communicates using messages, thus aiming at giving a clear conceptualization that can be achieved in several ways. The main advantages provided by the use of services revolve around the interoperability, loose coupling, isolation and composability.

SOA is a way of reorganizing software applications and infrastructure into a set of interacting services. However, the basic SOA does not address overarching concerns such as management, service orchestration, service transaction management and coordination, security, and other concerns that apply to all components in services architecture. Many

initiatives have emerged to meet this need, which includes SOC, Enterprise Service Bus (ESB) and Component Architecture (SCA).

2.1.2 Web services and Architectural Styles

Web services have been the focus of several standardization contributions by global consortia such as W3C and OASIS and have been the most successful implementation of services, used by software corporations such as Microsoft and IBM, which let them be popular with traditional enterprise. World wide Consortium has also been a driving force behind Web service research.

The World Wide Web Consortium (W3C) [29] defines Web Services as *a software system designed to support interoperable machine-to machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Service Description Language WSDL[1]). Other systems interact with Web services through their descriptions and using SOAP[2] messages and HTTP with XML serialization in conjunction with other Web-based standards.* As it is mentioned in [30], this definition does not restrict the Web service to particular technologies or standards such as SOAP and WSDL, it just assumes that the higher levels of the Web services protocol stack are built on the foundation of SOAP and WSDL. There are, and will be in the future, other technologies and protocols to define the Web services.

2.1.2.1 Reference architecture

In general, the basic Web Service protocol stack comprises four protocols as depicted in figure 2.2.

The main characteristic of a Web service is the use of the World Wide Web and the Internet as a communication medium for services to communicate with each other and with service consumers. By using WWW, Web services use the existing URI infrastructure so as to be located by anyone having access to the Web. The URI scheme gives a name to each Web service which uniquely identifies it and allows one to use all existing operations on URIs in order to access it.

Web services extensively use the XML language. From the definition of the messages exchanged between services to the service description, everything is based on XML, which is quite advantageous. Indeed, Web services represent an SOA that relies on the following three XML-standards technologies:

[1] http://www.w3.org/TR/wsdl20/
[2] http://www.w3.org/TR/soap/

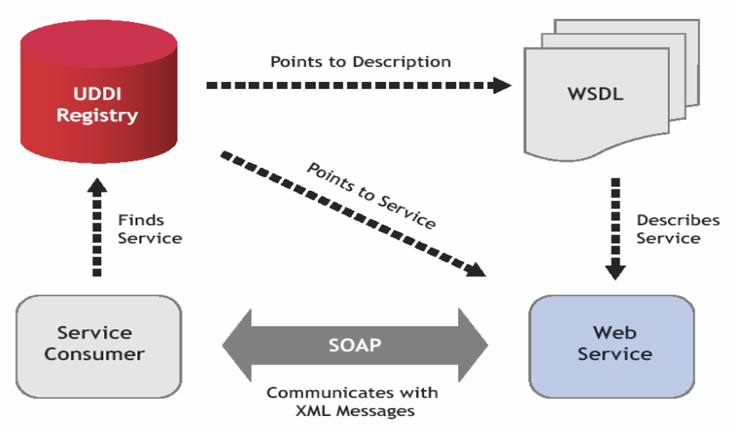


FIGURE 2.2: SOAP, WSDL and UDDI Interaction

- **Simple Object Access Protocol (SOAP)** is used for passing messages and invoking operations that Web services offer. SOAP represents a general pattern for Web service messages and defines how to encode an XML information in such messages and how these messages can be transmitted over the Internet using existing protocols such as HTTP and SMTP. Although one of the main purposes of SOAP is to support Remote Procedure Call (RPC), this protocol can support asynchronous or message based communications as well. In SOAP, the way a remote operation is invoked is specified through an XML document, while HTTP represents the transport protocol. By using HTTP, SOAP solves the problem encountered by typical invocation protocols such as COM+, Java RMI, and CORBA, where firewalls often block interaction messages. In SOAP, the basic item of transmission is a SOAP *message*, which consists of a mandatory SOAP *envelope*, an optional SOAP *header*, and a mandatory SOAP *body*. The SOAP *body* contains the main part of the SOAP message, that is, the part intended for the final recipient of the SOAP message. The SOAP *header* can be used to indicate some additional processing at an intermediate node, which is, processing independent of the processing done at the final destination. Typically, the SOAP *header* is used to convey security-related information to be processed by runtime components. The *envelope* specifies the XML name-space and the encoding style that identifies the data types recognized by the SOAP message.

- **Web Service Description Language (WSDL)** used for describing Web services and their relevant properties. WSDL represents a contract between the service requester and the service provider. It allows formalizing the service features according to a schema very similar to a typical Applications Programming Interface (API) definition. WSDL is a XML-based language able to specify the service feature described in text form. The first element comprising a WSDL specification is *service*, which identifies a set of services, each specified by a *port*. It

should be noted that a *port* only represents the physical address where the service operates and the protocols the user should adopt to communicate with it, with no description of the provided functionalities. This aspect is defined by the *portType*, directly associated with the *port*, which is responsible for defining the available operations. Hence, *portType* defines what the service does, whereas *port* defines where the service is. The *binding* element is responsible for defining this specialization by mapping the operations specified by the *portType* to a *port*, according to a particular protocol such as SOAP, HTTP, or SMTP. In more details, a *portType* is composed of a set of operations which reflect the functionalities characterizing the service available to the user.

- **Universal Description, Discovery and Integration (UDDI)** protocol used for creating Web Service directories and searching for adequate services. A UDDI directory entry consists on an XML file that describes a given business as well as the services it offers. The services are defined in a UDDI document called a *Type Model* (or *tModel*). In most of cases, the *tModel* includes a WSDL file that defines a SOAP interface to an XML-based Web service. However, the *tModel* is extensible enough to describe any other kind of service. Moreover, the UDDI directory includes various ways to search for the services, through which different applications can build applications. For example, a user can search for service providers in a given location or for a business of a specified nature. The UDDI directory will then deliver the required information such as links, contacts and technical data that enable user to identify and evaluate which services can meet his/her requirements. Furthermore, UDDI allows finding businesses that user might want to obtain from Web services.

Web services have added a new level of functionality for service specification, publication, discovery, composition and monitoring, which extends the role of the Web from a support of messages interaction to a middleware for application integration.

2.1.2.2 Representational State Transfer (REST)

REpresentational State Transfer (REST) is an architectural style that has been introduced for distributed hypermedia systems [31–34]. Its core goal was to apply the powerful features of the Web to the SOA in an efficient way by making resources accessible through Uniform Resource Identifiers (URIs). Using these URIs gives a powerful concept to the RESTful architecture by making it possible to have different representations for the same resources e.g. the server can provide XML or JSON format for machines understanding and HTML content for human interpretation. This resource

can be accessed and mapped without restricting communication to a particular protocol. However, it is often used in conjunction with the Hypertext Transfer Protocol (HTTP) and its CRUD-operations like POST for Creating, GET for Retrieving, PUT for Updating and DELETE for Deleting. Two main features distinguish REST from the existing network-based styles: i) REST decouples the functionality and the constraints of the component from the details of its implementation. ii) It is based on applying a uniform interface between different components to simplify the system architecture and improve the visibility of interactions between its components.

Moreover, REST architectural style consists of a set of constraints that restricts the behavior of its architectural components. Among the main constraints, there are the *uniform interface constraints*, which are based on four main principles [31]:

1. *Resource identification.* One of the key features of REST is its abstraction of any information, that can be named, to a resource. Such abstraction leads to: i) encompass a set of information regardless of the details of their implementation and; ii) facilitate the changes of any concept without the need to change all the concepts that are linked to.

   REST identifies each resource by a Uniform Resource Identifier (URI) that best suits the nature of the identified concept. The quality of an identifier is evaluated by its validity over time. The less the quality is, the more the links will be possibly broken.

2. *Manipulation of resources through representations.* Resources are decoupled from their representation, which leads to access to their content in a variety of formats (e.g., HTML, XML, plain text, PDF, JPEG, etc.). Resource meta-data are available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control.

3. *Self-descriptive Messages.* For each user request, REST constraints the server messages to be self-contained (called also state-less), i.e., each message contains all information required to answer the task. HTTP is considered one of the main protocols widely used to interact with messages through its methods: GET, HEAD, OPTIONS, PUT, POST, and DELETE. The first three are used for read-only messages, while the last three are used for updating messages.

4. *Hypermedia as the engine of application state.* Sharing representations by sending self-descriptive messages to identified resources changes the state of the application. For example, successfully POSTing or requesting (via GET) a given concept are

accomplished via hypermedia links such as anchor tags that have an attribute ("href") or form tag that contains a resource URI. In such a way, a RESTful application enables the server to inform the client of the possible paths to change the state of the application via hypermedia.

In [33], authors used architectural principles and decisions as a comparison method to illustrate the conceptual and technological differences between RESTful Web services and WSDL/SOAP based Web services. Authors concluded that: On the principle level, both two approaches have similar quantitative characteristics. On the conceptual level, less architectural decisions must be made when deciding for Web services, but more alternatives are available. On the technology level, the same number of decisions must be made, but less alternatives have to be considered when building RESTful Web services. More details are exposed in [33].

2.1.2.3 WS* specifications

The Web services community has done significant work to address the interoperability issue, and since the introduction of the first Web services, various organizations have introduced other Web services–related specifications. Figure 2.3 illustrates a population of the overall SOA stack with basic standards and extended Web services specifications that IBM, Microsoft, and other significant IT companies have developed.

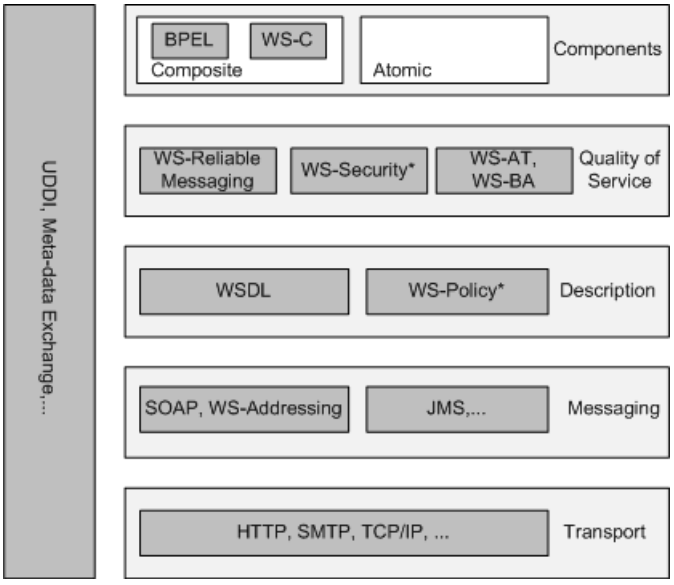


FIGURE 2.3: Web services technologies

WS-Addressing provides an interoperable, transport-independent way of identifying message senders and receivers that are associated with message exchange. WS-Addressing decouples address information from the specific transport used by providing a mechanism

to place the target, source, and other important address information directly within the Web service message. This specification defines XML elements to identify Web services endpoints and to secure end-to-end endpoint identification in messages.

WS-Policy proposes a framework that extends the service description features that WSDL provides. Having more refined service descriptions, qualified by specific WS-policies, supports much more accurate discovery of services that are compatible with the business application that is to be deployed. In a service registry (such as a UDDI registry), queries of WS-Policy-decorated services enable the retrieval of services that support appropriate policies in addition to the required business interface. Other Web services specifications will be detailed in the rest of this chapter.

2.2 Web Service Composition

Web service composition involves combining and coordinating a set of services with the purpose of achieving functionality that cannot be realized through existing services. This process can be performed either manually or automatically (or semi automatically in some cases), while it can occur when designing a composite service, hence producing a static composition schema or at run-time, when that particular service is being executed, leading to dynamic composition schemas. With static composition the concrete services are determined and integrated into the specification at design-time, and therefore every change of an already integrated services has to be taken into account in the specification. The dynamic composition of services requires the location of services based on their capabilities and the recognition of those services that can be matched together to create a composition. With dynamic composition, at design-time there is only a specification of the type of a given service. The concrete service is then integrated at run-time. Thereby it is possible that the concrete service has to be discovered first or that it is already known at run-time. To enable automated Web service composition, several requirements need to be addressed. These requirements include automation, dynamicity, need for semantics, support of non functional properties (such as time, security, QoS, privacy...), correctness, scalability, and adaptation [10].

Complex dependencies can be cerated between Web services offered by different organizations using compositional models such as choreography, orchestration, and coordination. These models are used by most of the Web services composition approaches.We note that these models are not used exclusively: one approach can implement more than one model at the same time.

2.2.1 Composition Models

2.2.1.1 Service Orchestration

The orchestration model consists in invoking and combining the services candidates via a central coordinator, called orchestrator (figure 2.4). In general, this model is static, so that, it is not able to adapt requirements and environment changes during run-time. The orchestration describes the dependencies between services operations; even that are not appearing in the service's behavioral interface for security concerns. A lot of orchestration researches, such as [35] and [36], differentiate between orchestration and composition synthesis. Indeed, composition synthesis specifies the coordination process that unrolls between services to meet the user requirements and generates a plan that is considered as a workflow to realize the required behavior by combining the capabilities of two-to-many services. While the orchestration consists in executing and monitoring the workflow generated by the composition synthesis. Orchestrations are also called executable processes since they will be executed by an orchestration engine.

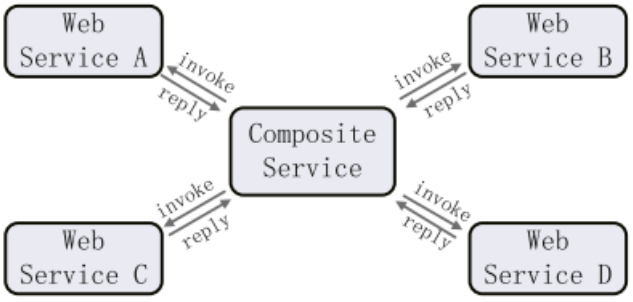


FIGURE 2.4: Service Orchestration

Various workflow languages have been proposed for service orchestration. Below we briefly introduce the most known languages.

- **WSBPEL**. WSBPEL [37] is an XML-based language for specifying workflow-based compositions and providing interoperability between different applications. It is based on specifying the detailed activities in the composition process that range from an abstract model to executable composition. It handles privacy by replacing private information by opaque activities. Among its limitations are its support of the static compositions and the total absence of the semantic representation of the WS. In addition, BPEL supports only implicit data flow, which requires mechanisms for binding service instances to the workflow activities. Moreover, BPEL compositions are mainly based on the interfaces of the composed services, so that it considers services capabilities as invocation interfaces and not as

functionalities. Besides, it supports only interaction with WSDL Web services, so tha supporting other service types requires some language extensions.

- **Web Service Flow Language (WSFL)**. WSFL [38] is a graph-based language that explicitly presents the control flow, which includes alternative execution paths, fault handling and compensation and event handling, as well as the data flow that defines the exchanged data between atomic activities and between the workflow and its candidates.

- **JOpera Visual Composition Language (JVCL)**. JVCL [39] is used by JOpera tool to provide a visual orchestration of processes. It defines the interactions between different services candidates through two separate graphs to specify the control flow and the data flow. It is independent of the type of the services to be orchestrated.

- **Declarative Service Orchestration Language (DSOL)**. DSOL [12] is a declarative, implementation-independent language that focuses on modeling different aspects of service orchestration using the notion of abstract actions and concrete actions. It focuses on what should we reach (the required goal) instead of how can we reach it. Moreover, it supports the dynamic feature of the service world by compensation and fault handling.

2.2.1.2 Service Choreography

The choreography model consists in capturing the interactions and the dependencies between a set of services to achieve a common goal. Differently to the coordination, service choreography describes the interactions between multiple services without the control of a coordinator agent. Orchestration is considered a more detailed, execution-driven mechanism, whereas choreography is more abstract and does not describe any internal action that occurs within a participating service. It encompasses all the interactions between the candidate services that are relevant with respect to the choreography's goal. Below, we review principal existing languages for defining choreographies.

- **Web Service Choreography Interface (WSCI)**. WSCI [40] is an interface choreography modeling language that aims to provide a standard for specifying the overall collaboration between Web services providers and services users. It is an implementation-specific language based on WSDL services description and supports bi-lateral interactions, contexts, correlations, exception handling and transactions.

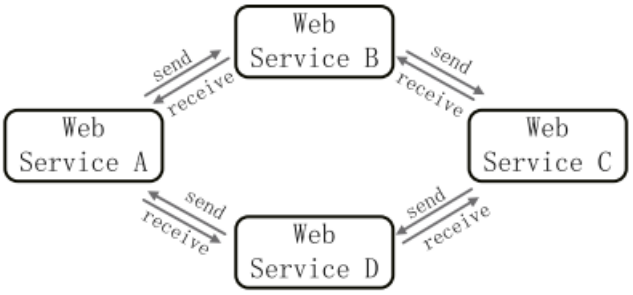


FIGURE 2.5: Service Choreography

- **Web Services Choreography Description Language (WS-CDL)**. WS-CDL [41] is an XML-based specification targeted for ensuring an interoperable, long-running and peer-to-peer interactions between services candidates involved in the abstract business process. It defines the process collaborations by bi-lateral interactions building blocks. Using a set of control flow constructs such as sequence, choice and parallel, this language enables to compose interactions into actions. It is an implementation-specific language based on WSDL or Java.

- **BPEL4Chor**. BPEL4Chor[42] is an implementation-specific choreography modeling language that adds a new layer in the top of the abstract BPEL to seamlessly support choreography and orchestration at the same time. BPEL4Chor consists of three perspectives: participant topology, participant behavior description, and participant grounding.

- **Business Process Modeling Notation (BPMN)**. BPMN [43] is an implementation independent graphical modeling language used to describe both inter and intra-organizational business processes. Moreover, it supports modeling complex control flow scenarios as well as private and abstract processes. Such a language still suffers from some shortfalls since it is not executable and lacks of formal execution semantics.

Although orchestration and choreography are two different models that can be used separately to generate a service composition, they can also be combined to provide a complete representation. Orchestration can be utilized to describe services candidates at lower abstraction level while choreography can provide a higher level description of interactions between these orchestrations to fulfill users goal. In this context, [44] proposed a framework that combines interface based and functionality based rules.

2.2.1.3 Coordination

Service coordination is based on temporarily combining a set of service instances through a coordination protocol. This protocol describes the possible interactions between services candidates as well as the outcome of these interactions, whether they were successful or not. All the communications unrolling during the coordination process are controlled by a third party, called coordinator. Below, we introduce some coordination specifications.

- **WS-AtomicTransaction (WS-AT)**. WS-AT [45] is a specification for atomic Two-Phase Commit (2PC) transactions coordination used as an extension to the WS-Coordination framework. Such specification is used by applications that need consistent agreement on the distributed-activities outcome with all-or-nothing property.
- **WS-BusinessActivity (WS-BA)**. WS-BA [46] is a specification for long-running distributed business transactions. In general, it defines two specific agreement coordination protocols for the business activity coordination type, which are *Business-Agreement-With-Participant-Completion* and *Business-Agreement-With-Coordinator-Completion*. Its Atomic Outcome coordination type requires that all candidates must either confirm or ignore their work.
- **WS-Coordination (WS-C)**. WS-C [47] is a specification for defining the context of a short and long running coordination used to create, share and register activities information carried across multiple services.

2.2.1.4 Semantic Web services

Web Services technology based on WSDL, SOAP and UDDI, define common standards that ensure interoperability between heterogeneous platforms. However, although low level interoperability is essential, SOA challenges go beyond data formats and communication protocols interoperability. The purely syntactic focus of WS technologies makes service description non interpretable by the machine which hampers the automation of operations, inherent to SOA, such as service discovery, composition and invocation. Semantic Web services initiatives have emerged with the objective of complementing the interoperability ensured by Web services to deal with data and behavioral heterogeneity along with automation support for capability-based service discovery, and dynamic service composition and invocation. The basic and common principle of these initiatives is extending syntactic service descriptions with a semantic layer the machine can interpret and reason over it. Ontologies play a central role for defining this semantic extension.

Ontologies define a common vocabulary and formal semantics by providing concepts, and relationships between them. Using a common vocabulary for describing services capability and behaviors ensures interoperability at data level. Formal semantics enable the application of powerful and well proven reasoning based techniques in order to enable capability-based service discovery and automatic service composition.

[48] presented and compared several SWS initiatives. Below, we just resume some of these initiatives.

- **OWL-S**. OWL-S [49] is an upper ontology for service description based on Web Ontology Language (OWL) [50]. As depicted in figure 2.6, an OWL-S service description consists in three interrelated parts: the service profile, the process model and the grounding. The service profile is used to describe what the service does; the process model is used to describe how the service is used; and the grounding is used to describe how to interact with the service. The service profile and process model are abstract descriptions of a service, whereas the grounding specifies how to interact with it by providing the concrete details related to message formats, and communication protocols.

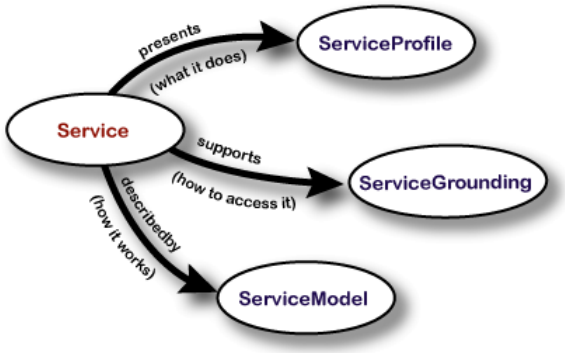


FIGURE 2.6: Upper Service Ontology for OWL-S

- **Web Service Modeling Ontology (WSMO)**. WSMO [51, 52] is an ontological conceptual model for describing various aspects related to SWS. WSMO refines and extends the Web Service Modeling Framework (WSMF), by developing a set of formal ontology languages. WSMF is based on two complementary principles that WSMO inherits: strong decoupling between the various resources and a strong mediation to ensure the interoperation between these loosely coupled components. While WSMO provides the conceptual model for describing core elements of SWS, WSML provides a formal language for writing, storing and communicating such descriptions. Following the main concepts identified in the WSMF, WSMO identifies four top level elements as the main concepts for describing several aspects of SWS, namely ontologies, Web services, goals and mediators [53].

- **Internet Reasoning Service (IRS-III)**. IRS-III [54] is a framework for creating and executing SWS. It acts as a semantic broker between a client application and deployed Web services by supporting capability-based invocation. A client sends a request encapsulating the desired goal and, by exploiting the semantic description of Web services, IRS-III framework: (a) discovers potentially relevant Web services; (b) selects the set of Web services which best fit the incoming request; (c) mediates any mismatches at the conceptual level; and (d) invokes the selected Web services whilst adhering to any invocation constraints. IRS-III service ontology defines the conceptual model of IRS-III framework. It extends the core epistemological framework of its previous IRS-II framework by incorporating WSMO conceptual model. Different from WSMO, IRS-III service ontology uses its own ontology language, OCML [55]. While there are some differences between IRS-III and WSMO conceptual models, IRS-III service ontology defines the same concepts for describing SWS namely goals, Web service capability and interface (choreography and orchestration), and mediators.

- **METEOR-S**. METEOR-S project [56, 57] addresses the usage of semantics to support the complete lifecycle of Semantic Web processes using four kinds of semantics - data, functional, non-functional and execution semantics. The data semantics describe the data (inputs/outputs) of the Web services. The functional semantics describe the functionality of a Web services (what it does). The non-functional semantics describe the non-functional aspects like Quality of Service and business rules. The execution semantics model the behavior of Web services and processes. Unlike above initiatives, METEOR-S does not define a fully fledged conceptual model for SWS description. It rather follows a light-weight approach by extending WSDL files with semantic annotation. The semantic annotation is achieved by mapping WSDL elements to ontological concepts. WSDL-S citewsdls, METEOR-S specification for WSDL annotation, was one of the main works that influenced SAWSDL [58] the W3C standard for WSDL and XML schema semantic annotation. SAWSDL consists in adding semantics to the WSDL files using semantic annotations associated to ontological concepts. Such annotations introduce new extensions, such as model Reference and Schema Mapping attributes, that may be applied to service interface as well as service operations [59].

The first three initiatives separate explicitly between the semantic and syntactic descriptions of a Web service and link them using the concept of grounding that maps abstract concepts and data types of the semantic description to concrete data formats and communication protocols at the syntactic level. METOER-S, however, semantically annotate WSDL files by linking their elements to ontology concepts and relations.

2.2.2 Non-functional requirements

Modeling, managing and performing service related tasks such as discovery, composition, negotiation and agreement based on non functional properties become fundamental challenges in SOA especially in real business settings.

How Web services are described is crucial for the successful realization of service related tasks like discovery, selection, and composition. Three different layers can be considered when talking about services : (1) behavioral, semantic (2), and (3) non-functional. The behavioral description is about how the functionality of the service can be achieved in terms of interactions with the service as well as in terms of functionality required from the other Web services. The semantic descriptions make easy for programmers to combine data from different resources and services without losing the meaning. Finally, non-functional descriptions capture constraints over the behavioral one. For example, in case of a train booking service, invoking its functionality (booking a train ticket) might be constrained by using a secure connection (security as non-functional property) or by actually performing the invocation of the services in a certain point in time (temporal availability as non-functional property). Among the three aspects of a service description, the behavioral and semantic aspects are the most investigated aspects. Although the third aspect, non-functional properties, did not capture a very broad attention from the Web service research community, as behavioral and semantic descriptions did, one has to recognize the big importance of describing them. This is due to their high relevance for all service related tasks. Non-functional properties might play an important role in all service related tasks, especially in discovery, selection and substitution of services. It is simple to imagine a scenario in which services which can fulfill a user request and which provide basically the same functionality are selected based on some non-functional properties like price or performance. The lack of real support (languages, methodologies, tools) for non-functional properties might be due to various factors. These factors include:

- Non-functional properties are usually too abstract and most of the time they are stated informally.
- There is no clear delimitation between the functional and non-functional aspects of a service.
- Non-functional properties are complex to model and difficult to formalize.

Different formalisms have been proposed to compose services satisfying QoS parameters. These efforts can be classified into three main categories as depicted below.

- *Local optimization.* In this type of optimization, the composition process consists in selecting the best instance for each functionality existing in the composition plan. For example, authors in [60] propose an heuristic composition approach named Local Optimization and Enumeration Method (LOEM). This method is mainly based on filtering the candidate services of each sub-goal through a local selection and then enumerate the composed solutions to reach a close-to-optimal ones. [61] proposes an extension to the meta-framework for service composition presented in [62] by developing formal operations that satisfy non-functional user requirements. Calculating the preferences over non-functional attributes is based on denoting the non-functional valuation of each service that satisfied the functional requirements. Then, the most dominant service will be taken. However, the main limitation of the local optimization-based approaches consists in identifying the optimal service at design-time phase, which can be inappropriate due to the changes that can be appeared at the run-time level of the composition process.

- *Global optimization.* This category focuses on selecting the composed service that best meets the user requirements among the other composite services. For instance, researchers in [63] aim to provide an optimized selection of the most closed composition to the user preferences among the feasible compositions through a fuzzy system. After generating a set of execution plans, each feasible plan's quality will be calculated and the optimized composed services are specified in a quality vectors, from which the optimal plan will be extracted. In [64], researchers tried to support dynamic Web service composition by a multi-objective selection method based on multi-objective ant colony optimization (MOACO). It consists of converting the QOS global optimization into a multi-objective optimization problem with user constraints. Furthermore, based on the MOACO, a set of optimal solutions, known as Pareto set, is calculated through a multi-objective genetic algorithm (MOGA).

- *Hybrid approach for optimal service composition.* Recently, much efforts deviated toward combining global and local selection techniques to benefit from their powerful concepts. For example, researchers in [65] proposed a mixed approach based on i) finding the most optimal decomposition of the QoS properties into local constraints through a mixed integer programming (MIP), and ii) identifying the optimal composite-services that meet the QoS properties through a distributed local selection.

An efficient approach to handle non-functional properties associated with services may consist in considering them as constraints exhibited over the functionality of the service. Indeed, the ability to describe non-functional properties of services in a rich way has

applicability in the tasks of service discovery, selection, monitoring and substitution. An increased level of service property information would also facilitate more through decision-making by a service requestor. Non-functional properties must be considered as early as possible in the development cycle in order to avoid costly failures.

2.3 Web services composition approaches

The past decade has witnessed significant innovative approaches to enable automatic Web services compositions. A lot of platforms, languages, tools and techniques, have been developed [9], [10]. Due to the diversity of the existing efforts and in order to make the presentation of the approaches easier, we propose the following classification to the approaches to be examined:

- Procedural Composition Approaches
- Declarative Composition Approaches

Then, for each class, we review the proposed approaches according to what composition stage they belong: modeling, verification, and monitoring.

2.3.1 Procedural composition approaches

2.3.1.1 Modeling approaches

Various initiatives have been raised to design and model Web service compositions. Graph-based and model-based techniques can be considered as important efforts among these initiatives. Graph-based planning techniques consist of multi-level structures. The first level of the graph includes the inputs and the initial state of the composition process. Starting from the first level, a planning algorithm is applied to generate the applicable actions within the precedent level and a new level that consists of the derived effects of the actions will be created. This process will be iterated until reaching the required outputs in the last level and the solution plan, if exists, is extracted. Various approaches were developed in this context. For instance, the graph-based composition framework, proposed in [66], consists of three main steps. The first step analyzes the given inputs, while the second step provides the desired outputs through a third step that is based on a graph planning algorithm to select and compose applicable services. The algorithm used applies a backward search method for each given input of the services existing in the final actions layer. A similar graph-based contribution was presented in [67], through

which the existing services are represented as nodes and their possible connections are represented as edges. To solve the composition problem, the proposed approach is based on four main stages: preliminary stage, search stage, simulated execution stage, and selection and execution stage, where only the first two are related to the composition task. The preliminary stage is concerned with the inclusion of special nodes into the (initially preprocessed) graph for representing the request parameters. The second stage aims at searching for all sub-graphs satisfying the user request. The algorithm performs a breadth-first search in an AND/OR tree, where an AND node is a service cell for which all its inputs have been provided, whereas an OR node is one input which has to be provided by at most one output of another service cell. The interesting feature that differentiates this work from the other composition works is its independence from the inputs given in advance. Starting from the required outputs, it checks and identifies the missing inputs.

Other contributions use model-based composition techniques that range from Petri-Net to workflow-based techniques. For example, a method based on Petri Net cover-ability was proposed in [68] to provide an automatic Web service composition. Each input and output of an atomic service are mapped into places of a Petri net, whereas each service is considered as a transition of a Petri net. The existing of a token in a given place means that the corresponding input was given by the user. According to the given inputs and the desired outputs, the initial and target markings of the Petri net can be obtained and the coverability graph can be generated and used to extract all shortest paths covering the final marking and then identifying a sequence of service executions from the transitions. Brogi and Corfini [69] presented Service Aggregation Matchmaking (SAM), a system that is able to discover service compositions, or create them when no suitable composition for a given request is found. The model used to represent Web services is Consume-Produce-Read (CPR) Nets, a simple variant of the standard Condition/Event Petri Nets defined by the authors to properly model the control flow and the data flow of a service. The authors provide a translator from OWL-S descriptions to CPR Nets. The SAM framework can handle both functional and behavioral requests (i.e. requests declaring the desired functional properties of the service or its desired behavior). The functional analysis returns a set of participating services based on a functional request, while the behavioral analysis can generate a composite service based on a set of participating services and a behavioral request. [70] proposed a framework that automatically generates a Web service composition schema from a high-level goal. Taking the target (objective) as input, an abstract workflow generator creates a BPEL-based workflow that meets the objective, either by exploiting already generated workflows or parts of them or by coordinating a set of services to meet the goal. The generated workflow is then concretized either by finding existing

services for each candidate activity of the resulting workflow or by recursively calling the workflow generator in the case that no service can be found for a candidate activity. In a more recent work, [71] propose an extension to the Partial Order Planning technique to support actions with multiple conditional effects. The preconditions and effects are expressed by a conjunction of predicates, whereas the conditional effects are represented as pairs of conditions and effects. SC-APOP planner is used to generate Partial order plans, which will be then translated to a directed acyclic graph (DAG). Each vertex of the graph corresponds to an action and each edge corresponds to an ordering constraint. After translation, a Transitive Reduction [72] strategy is performed on the DAG in order to reduce the complexity of the workflow diagram to be generated. In the post-processing phase, the final workflow diagram is derived from the reduced DAG by adding new vertices to represent the recognized workflow patterns.

Although their efforts to provide an automatic service composition, none of these approaches considers the non-functional properties in the modeling of the composition process. To address this issue, different approaches have been proposed, such as PAWS [18], which is a framework focusing on the adaptation and the flexibility of the service compositions modeled as business processes. This framework consists in creating a BPEL process to be annotated with global and local QoS-constraints. In particular, it includes an advanced service retrieval module that is used to discover service implementation suitable for each task in the business process. The selected service instance has to suit the required interface and do not violate any constraint by meeting SLA negotiation. If no existing service matches the required task interface, a mediator is utilized to reconcile the interface inconsistencies. For each activity, more than one candidate service is selected. When the process is executed by a BPEL engine, one service is invoked for each task. Fujii and Suda [73, 74] propose an architecture for context-aware service composition called CoSMoS. The framework is based on a semantic abstract model for representing service components as well as service users. After receiving a user request, the framework converts the query to a CoSMoS model instance through pre-existing natural language analysis mechanisms. The converted query will be used by the workflow synthesis module to create an executable workflow by retrieving and interconnecting components according to that of the request and the functional descriptions of the existing components. This module is limited to sequential and parallel composition patterns. Then, a semantic matching module checks the functionality equivalence between the required task and the offered service. If more than one functionally-equivalent service are available, a context-aware discovery will be performed to select the most suitable component. The resulted workflow will be then executed and monitored to support the dynamicity of the context and the user requirements. Berardi et al. [75] presented

e-Service Composition (ESC), a prototype tool that implements a model-based technique for automated service composition using Finite State Machines (FSMs). They argue that the behavior of a service can be modeled using two schemata, an external schema that specifies its exported (externally-visible) behavior and an internal schema that contains information on the service instance of each action in the composition process. Both schemata are expressed using FSM models. When attempting to synthesize a composition, the external FSM models of the available services and the target service are transformed to model logic formulas in Deterministic Propositional Dynamic Logic (DPDL). If the resulting set of formulas is satisfiable, it means that there exists an FSM for the target service. The automatically synthesized FSM can then be converted to a BPEL process and executed in a BPEL engine. In a more recent work [76], Berardi et al. replace FSMs with Transition Systems and use the notion of simulation to virtually compute all possible compositions at once.

One commun pattern of the above modeling approaches is that they are mostly procedural and they over-constrain the process making it rigid and assuming the design choices that may not be present in the requirements but only added to specify the process flow.

2.3.1.2 Verification approaches

Verification of service composition proceses, both a-priori and at run-time, is an invaluable tool to guarantee dependability of the client on the available services. Various verification methods have been introduced in the literature. [15] proposed ASTRO methodology to check the correctness of the services operations using a verification mechanism that may be used in both on-line and off-line mode [77, 78]. [79] introduces a formal verification approach based on Colored Petri Net (CPN) [80], which detects the inconsistencies at design time by performing a structural and behavioral analysis independently from the concrete-flow language. The verification is ensured by creating a CPN model for the service composition using boundedness and liveness properties of CPN. The resulted occurrence graph, named O-Graph, consists of a set of nodes that represent the reachable marking, and a set of arcs that represent occurring binding elements. Event-B mechanism is recently used by several verification approaches, such as [81], which is based on transforming BPEL process into Event-B [82] models to verify the relevant properties of the composition process. Similar approach is proposed in [83] to check the transactional consistency of service composition at design-time using refinement and proof mechanisms. Work presented in [84] propose to verify BPEL processes using BPEL2SEM component [85], which consists of an automatic verifier to perform a semantic analysis of the flow constructs used in the definition of BPEL4WS

processes. Taking the BPEL process as input, BPEL2SEM translates it into Prolog-based language and starts by checking i) if there exists at least one activity in the BPEL definition able to start the process; ii) if the elements links are described in the flow activity; and iii) if every link declared in the flow activity has exactly one source activity and one target activity. These constraints are checked using inferencing prolog rules of the Semantics Rules knowledge base and this analyze finishes by providing information about process execution (traces) in both cases of correctness and incorrectness of the process definition. SU Huan et al.[86] propose to verify the service composition by converting the BPEL4WS-based composed services into Interface automata using Promela language [87]. The generated automata will be then verified by SPIN tool [88] to check its correctness. This approach is applicable for deterministic systems rather than non-deterministic systems. In the same scope, other type of automata, called Timed Automata, was proposed in [89] for verification concerns. It is based on converting WSCDL-based composed services into Timed Automata and verifying it using UPPAAL tool [90, 91]. Such method is useful in non-deterministic systems, however, it is applicable just in the design time stage, without taking into consideration the non-functional properties such as security and temporal requirements. To address these issues, a recent work proposed in [92] consists of a combination of Muller Automata (MA) and Push Down Automata (PDA), named Enhanced Stacked Automata Model (ESAM). The proposed method is based on converting the BPEL4WS-based composite services into ESAM using Promela language. Taking the generated model and a set of required verification properties (such as dead transition and deadlock) as inputs, SPIN tool is used to verify whether the system design meets the requirement or not and prove the correctness of process interactions in both deterministic and non-deterministic systems.

One common pattern of the above verification approaches is that they require to map the process to some formal logic and then verify the process. However, this lack of integration results in a complex model and it may not always be possible to have a complete transformation from one modeling approach to other. Further, with the addition of non-functional requirements the transformation becomes even more complex and challenging.

2.3.1.3 Monitoring approaches

To detect anomalous situations and maintain high level quality of the generated service during execution phase, many efforts tried to provide an efficient runtime monitoring of service composition process. These efforts addressed mainly the difficulty of locating the appropriate monitoring instances that should be quickly updated or executed. To

overcome this problem, [13] proposes an on-line Web service monitoring system that dynamically analyzes multiple data-centric properties of composition processes. This system is based on a Parametric Behavior Constraint Language (Par-BCL) and aim to optimize the monitoring process by statically generating the monitored properties to a parameter state machine behavior. Thus, the violation of a specified property can be verified. [14] proposes an adaptive service composition (ASC) framework to adapt user requirements changes and environment dynamicity. The proposed framework tries to link between the required services and the existing ones depending on the type of change. Each ASC process is based on goal-driven, environment-triggered or service-association adaptation strategies. Astro [15] includes an execution and monitoring component based on the ActiveBPEL engine. Robinson [16] proposes a framework and a tool to automatically derive Web service monitors from high-level requirements descriptions. Bostrom et al. [17] present a rule-based engine for monitoring service invocations and evaluating if the service fulfills the obligations in its SLA. PAWS framework [18] supports self-healing that allows faulty services to be substituted by other candidates and enables recovery actions to cancel the results of the faulty services. [19, 20] aim at monitoring the behavior compliance of Web service compositions represented as WSBPEL processes with a set of requirements. Other approaches concentrate on capturing and monitoring negotiations, which incorporate security policies and policy models to facilitate service life-cycle management [21]. [22] focuses on monitoring and capturing private data use based on Web services agreements.

Other approaches believe that monitoring QoS attributes of Web services is one of the main aspects to guarantee the quality of the generated services and enforce Service Level Agreements (SLAs) that are associated with business partners. For that, they focus on monitoring QoS properties during composition process. For example, [93] introduces a Web Service Level Agreement (WSLA) framework to provide the required specification and monitoring of QoS-aware Web services by applying an electronic contract. Monitoring of the compliance with an associated contract is implemented using the IBM Web Services Toolkit. Researchers in [94] propose a new mechanism for QoS attributes (such as response time and availability) monitoring by combining the client-and server-side with a Web service runtime, called VRESCo [95], to detect the possible SLA violations for Web services and enable subscribers to react appropriately to such violations. However, the proposed monitoring method does not go beyond the procedural methods that need client intervention to react to SLA violations. Other efforts adapt a cross-layer monitoring methodology for services-based systems. Among these efforts, there is the S-Cube European Network of Excellence on Software Services and Systems [96], which extend the well-known Monitor-Analyze-Plan-Execute loop by proposing a multi-layered version. The loop was then applied in a preliminary prototype that integrates various

single-layer approaches introduced in [97]. However, this integration is based mainly on BPEL processes, and suffers from the lack of well-devised abstractions that could guide the integration in a more general way. In the same scope, Monere system, introduced in [98], is a multi-layer based approach that manipulates the system across its different layers and exploits the BPEL-based workflows that compose the services to generate structural cross-domain dependency graphs. The system components will be then monitored and their parameters will be correlated to introduce the system manager with an easy to use multi-layer diagnosis. However, this approach focused mainly on generating dependency graphs from BPEL specifications, without detailing the multi-layer monitoring task.

Two common patterns of the above approaches can be observed. First, the run-time monitoring of the composition process which is tightly coupled with the composition process was not well integrated to these approaches, and therefore do not provide the important execution time violations feedback to the composition process. Second, they are highly procedural, which make the possibility to learn from run-time violations and to change the process instance/model at execution time very difficult. Once again, we believe that these proposals are more oriented to low-level analysis (services/components), while we are more interested by business activity monitoring.

2.3.2 Declarative composition approaches

2.3.2.1 Modeling approaches

In this context, some approaches were proposed. [99] defines bucket rewriting algorithms to reformulate the user-query into a set of functionalities that directly refer to the existing services. Possible executable compositions are ranked according to user preferences related mainly to the service level. The highest composition will be selected and returned to the user. [100] proposes an approach based on modeling the available services into RDF views over a mediated ontology. The generated views are then integrated as annotations into WSDL files. User queries are transformed into a mediated ontologies using SPARQL query languages. According to the reformulated query, the Web Service Management System (WSMS) selects the candidate services through the RDF views defined in the WSDL files. Finally, the selected services are executed and composed through an RDF rewriting algorithm. This approach addresses only data privacy issues when invoking the candidate services. [101] proposes an approach that complements the above proposal by taking into consideration non-functional requirements using the fuzzy sets theory [102]. A fuzzification of Pareto dominance is used to compute the rank of the resulted composed services. [103] proposes a composition approach based on

HTN planner that decomposes the desired task into sub-tasks and apply recursively the decomposition process until satisfying the resulting sub-tasks. In case of a state change, the whole composite process will fail, and a new re-planning will be activated. In [104], the authors combine Constraint Satisfaction with HTN to convert user requests to valid inputs for the HTN planner (such as SHOP2). The HTN planner generates a workflow and a set of constraints that are then fed to a Constraint Satisfaction solver. This latter attempts to find a solution that satisfies all constraints by invoking candidate services with respect to a given task flow. The authors noted, however, that providing a solvable Constraint Satisfaction set can be so difficult for some domains and may require an expert intervention in order to create a broader set.

[105] and [106] propose to extend the HTN technique and provide an HTN-DL planner, which consists in cascading description logic (DL) [107] representations and reasoning over them. Various ways of the decomposition can be identified and in case of an operator matching, the planner will invoke the action and apply effects to the current state. In case of a method matching, the decomposition process will be continued until reaching an atomic action. An extension of the DLs, called Dynamic Description Logics (DDLs), has been discussed in [108, 109]. For instance, G.Shen et al. proposed, in [110], a DDL-based formalism for modeling and reasoning about Web services and their behaviors. This formalism combines semantic and formal reasoning at a highly abstract level and reduces the service realizability and executability by checking formulas satisfiability and consistency. Another DDL-based composition approach is proposed in [111]. It consists of two phases: the first phase generates a partial order diagram, and the second phase runs the composition process using a fast algorithm according to the generated diagram. In [112], the authors proposed an extension of the Arithmetic and Logic Class (ALC), called Service Dynamic Description Logic (SDDL), to support the discovery and composition of Web services. [113] proposes a new DDL-based composition model, which supports context-based services using a bottom-up filtering approach. This contribution increases the DDL-reasoning efficiency by providing context-aware support and reduces the reasoning space. It is based on an abstract reasoning without detailing the orchestration process and the action-based reasoning.

Zeng et al. [114] propose the formulation of service composition as a goal-directed planning problem that takes three inputs: service composition rules (which are manually defined), goal specification and a set of business assumptions (organizational rules and structures). To that extend, they divide an ontology to represent these concepts and use a three-step composition schema generation process. In the first step, called Backward-Chaining, the composition rules are exploited trying to provide a chain starting from the goal and moving backwards, until no more rules exist, or the initial state is reached. The second step, Forward-Chaining, attempts to complete the first phase

composition schema by adding services that are required to fulfill the tasks following an opposite direction. The final phase, Data-Flow-Inference, adds data flow to the composition schema, since the previous steps only contribute to the control flow aspects of the composition. [115] creates service compositions using planning as theorem proving in the event calculus [116]. The planning problem in the event calculus is formulated by the domain knowledge, the event calculus axioms and a goal state. The theorem-prover generates the plan, which is a sequence of events and temporal ordering predicates.

A two-staged composition process was proposed in [117]. The first phase is a logical phase that handles the functional requirements and specifies the business process by generating control flows through a logical system. The second phase is a concrete phase that takes the control flow as a composition template and generates executable data flows by selecting the optimal service instances according to the non-functional requirements. The provided data flows will be then parsed into a BPEL flow that can be deployed and executed. Researchers in [118] proposed a constraint-based composition approach that consists of two stages, each of which includes a set of constraints targeted to handle the composition. In the abstract stage, the user specifies the abstract composition workflow and identifies local, choreography and non-functional constraints. The second stage instantiates services candidates according to the first-stage workflow and a set of execution constraints. The proposed framework tried to refine the Web service composition process using a two-stage technique and a set of constraints for modeling and monitoring the composition. Otherwise, this framework is semi-automatic and needs user intervention especially in the first stage. However, there is no implementation details nor evaluation study to highlight the effectiveness of the proposed framework. Other two-stage based approaches were proposed in [119] and [120]. In the first phase, the framework classifies candidate services based on their QoS according to the user's preferences through an associative classification algorithm. The second phase uses this classification to rank the filtered QoS-optimal services focusing on their semantic functional matching.

2.3.2.2 Verification approaches

To validate the service composition, two components are proposed in [99]. The first component is a re-writing checker used to eliminate the non-useful rewritings, which have no potential solutions. The second component is a configurator to detect any possible inconsistency by enacting two types of constraints: i) business-level constraints including order and dependency relationships in data and control flows, and ii) service-level constraints relevant to a specific service, such as its preconditions and postconditions. Sohrabi and McIlraith [121] introduce a modified HTN planning algorithm, named HTNWSC, which takes into account both preferences and regulations in the plan generation procedure.

These regulations include verification-geared constraints, such as safety, that should be followed by any generated plan. [122] represents an automatic verification approach for OWL-S service composition using the Spin model-checker [88]. Such verification necessitates mapping OWL-S service description to SPIN's Promela language [87]. To verify the satisfiability of a specification over a given claims, SPIN requires Promela-based design specification and Linear Temporal Language LTL-based correctness claims, which consist in formulae to represent the execution paths of a composition process. In case of non-satisfiability, the statements, under which the error occurs, are identified. [123] addresses the task of automatically verifying a service composition at design time by checking if the system exhibits some particular behavior. The proposed methodology is based on a bottom-up projecting style, where the application is built by putting together the available components and identifying the possible fruitful interactions that can exist between them. These components consist of a given set of services specified by a declarative language, called ConDec [124], augmented by a set of roles and participants. The specified services candidates are then mapped to SCIFF rules [125] to automatically perform the verification tasks. [126] introduces a declarative approach to specify and verify service choreographies by mapping a set of graphical constructs of DecSerFlow framework [127] onto logic-based languages (Linear Temporal Language and onto Abductive Logic Programming) that enable to reason over the developed models. Zahoor proposed in [128] and [129] to use a bounded model-checking method based on satisfiability solving (SAT) to verify the Web services composition processes in a declarative manner. The proposed approach is based on a modeling formalism that consists of a combination of Event Calculus (EC) [130] and SAT encoding process [131]. EC is used to specify both composition and verification models, while filtering criteria are used to remove any anomalies and conflicts (such as deadlocks) that can be derived from SAT solver.

2.3.2.3 Monitoring approaches

The run-time monitoring of Web service compositions has been widely acknowledged as a significant and challenging problem. There have been several proposals that deal with different aspects of the monitoring of Web services and distributed business processes. [23] introduces a constraint-based declarative approach that uses the same constraints in the modeling phase as in the monitoring phase of the composition process. The proposed framework includes execution monitors, each of which contains a set of activation constraints (such as pattern detection) and its associated actions as well. An event listener is used by the monitor to listen to the (sent or received) messages attached to the composition process. In case of run-time violations, such as failure of instance execution or

delay of response-time, the composition workflow will be refined. In a more recent work, the authors defined in [24] a self-healing system that consists of three phases for composition monitoring, which are i) detection phase that detects the constraints violations or temporal ordering mismatch by keeping track of the exchanged messages between the composition system and the candidate services during execution process; ii) side-effects calculation phase that extracts the side-effects of the detected violation by returning the violated part of the plan and re-invoke the reasoner; and iii) recovery phase that recovers the composition from violation by finding alternatives using user recovery constraints. [132] proposes a declarative approach for monitoring functional correctness and quality of service (QoS) metrics during the composition run-time. The Monitor is mainly based on integrating and extending the Web Service Offerings Infrastructure (WSOI) [133] for monitoring functional and QoS properties of the services. The monitor uses the workflow itself to provide an execution trace, by adding additional information, such as the values of the service attributes at run-time and whether functional/non-functional constraints are verified or not, to each executed service. However, adaptation mechanisms to overcome the detected anomalies are still underway. In the same context, [134] proposes a policy-based monitoring framework, named Manageable and Adaptive Service Compositions (MASC), to monitor functional and QoS policies of a composite service. Service monitoring policies are specified using WS-Policy4MASC language and can be associated to the SOAP messaging layer as well as to the orchestration engine layer. A cross-layer monitoring of complex services is proposed in [135]. This approach uses a new declarative language, called Multi-layer Collection and Constraint Language (mlCCL), to ensure an on-line and off-line analyzes of the collected data in a multi-layered system. The proposed monitoring technique was applied to SocialTools, which consists of a set of services that are based on Service Component Architecture (SCA) components and hosted on Amazon EC2 to collect up-to-date information about the Social networks clients. [136] addresses the complexity of the SLA monitoring activity by introducing an event-based monitoring framework designed for complex Service Based Systems (SBSs). This framework is based on a hierarchical design and consists of a cross-layer monitoring mechanism of different properties at different layers of SBSs such as the workflow execution environment of a service composition, services invocation, and services execution. Moreover, this framework takes into consideration additional features such as events coordination property that is based on applying a recursive monitoring method of all the services candidates of the workflow.

2.3.3 Synthesis

Table 2.1 resumes the main WSC approaches that are proposed to satisfy one-to-many phases of the composition life-cycle as well as its requirements. (✓) denotes that the corresponding approach satisfies that particular phase/requirement. (∼) denotes that the corresponding approach only partially satisfies that particular phase/requirement. No symbol denotes that the corresponding approach does not satisfy that particular phase/requirement.

From a technical point of view, the composition approaches described above can be categorized into two classes. A first class aims to provide a fully automatic composition from a large set of semantically-rich service descriptions, to be interpreted, selected, combined, and executed by the composition engine without the intervention of the service architect. A second class which is less ambitious leaves to service architects the goal of defining an abstract model of the composition, which the engine interprets and makes concrete by selecting and invoking the actual services to accomplish each task.

With relation to our objective to compose services in a declarative and integrated way, none of the two approaches completely solve the addressed problem. Although the former works in too specific and restricted domains, it can hardly be applied in more general settings, since it requires all services to be semantically described with enough details to allow the engine to choose and combine them in the right way to satisfy the users' goals. The latter is too restrictive and needs from the service architect to provide an abstract yet detailed enough model of the composition, often using languages like BPMN and BPEL, whose structure is the main source of the problems [12].

Also, from the above literature review study, it was gathered that:

- One common pattern of the studied approaches is that they tackle the composition problem by focusing on the control flow of the composition process. The problem is that flexibility and control on the composition process are conflicting requirements, and therefore we should find a compromise between flexibility and control flow.
- The majority of these approaches are procedural (using imperative languages such as BPEL and WS-CDL), which makes the composition process rigid and enable to handle dynamically changes.
- Some important aspects for the composition such as data, temporal constraints, security and other non functional requirements were not considered in the approaches examined.

	Approaches	Modeling	Verification	Monitoring	Recovery	NFP
Procedural Composition	[70]	(✓)			(∼)	
	[18]	(✓)		(∼)	(✓)	(✓)
	[73] [74]	(✓)		(∼)		(✓)
	[75] [76]	(✓)	(∼)			
	[66][67][68] [71][67][137]	(✓)				
	[138]	(✓)				(✓)
	[69]	(✓)				(∼)
	[15]		(✓)	(✓)		
	[79] [81] [84] [83] [86] [89]		(✓)			(✓)
	[92]		(✓)			
	[13]			(✓)		
	[14]			(✓)	(✓)	
	[16] [17] [96] [97] [98]			(✓)		
	[19] [20] [22]			(✓)		(∼)
	[93] [94]			(✓)		(✓)
Declarative Composition	[99]	(✓)	(✓)			(✓)
	[100]	(✓)				(∼)
	[103] [104] [139] [105] [106]	(✓)				
	[110]	(✓)	(✓)			
	[111]	(✓)				
	[112]	(✓)				
	[113]	(✓)				(✓)
	[114] [115] [140]	(✓)				
	[117]	(✓)				(✓)
	[101] [118] [119] [120]	(✓)				(✓)
	[121] [122] [123] [126] [128] [129]		(✓)			
	[141]			(✓)	(✓)	
	[23] [135] [136]			(✓)		
	[24]			(✓)	(✓)	
	[132]		(∼)	(✓)		(✓)
	[134]			(✓)		(✓)
	[12]			(✓)	(✓)	

TABLE 2.1: Comparative Study of Automated Web Service Composition Approaches

- Non-functional properties did not capture a very broad attention from the Web service research community despite their importance, and there is a lack of real support (languages, methodologies, tools) for such properties
- The run-time monitoring task which is tightly coupled with the composition process was not well integrated to the traditional composition approaches, and very few proposals handles it by adding a new layer for the composition monitoring and thus do not provide the important execution time violations feedback to the composition process.
- Verification and validation of service instances or process models can hardly cope with the complexity and dynamics of an end-to-end business compliance framework both at design time and runtime.
- Existing formal verification methods are not integrated with the existing service or process models, and hence a semi-automated verification is hard to achieve in an end-to-end business compliance framework.

To provide an efficient solution that addresses the above limitations, composition approaches should respond to the following requirements:

1. Specifying composition requirements using declarative languages to help end users for specifying the functional and nonfunctional requirements of a composite service.
2. Enabling to handle the continuous and unpredictable change of the composition process.
3. Explicitly managing runtime faults and exceptions.
4. Offering tools that are able to specify requirements, to verify these requirements, to monitor if the execution is compliant to the specification, and to analyze what really happens during the execution of the composition.
5. Providing most commonly used controls and menu options and enabling all the complex functionality through these. This is to help create a familiar environment with most tasks organized in an intuitive fashion.
6. Enabling to incorporate the monitoring process within the composition framework, rather than building it as an external component.
7. Supporting rapid re-design and re-deployment of services, through appropriate reuse of parts of previously assembled services or incorporation of new components.
8. Providing support for bridging semantic gaps.

9. Provide support to deal with the increasing number of Web services, which demands for an accurate and scalable solution to select appropriate services.

10. Dealing with a set of disparate technologies while hiding all heterogeneity from the user. This is important to unify the solutions developed by various communities for different but related problems. The integration involves providing support for different kinds of editors, viewers and registries.

In our work, we progress beyond the composition approaches described above by:

- providing more reusable composition strategies through abstract and well defined languages while the majority of the approaches focus on specific technologies or composition techniques.
- introducing expressive languages for verification and compliance concerns, tackling the whole lifecycle (including runtime elements and governance), and integrating formal model for behavior modeling.
- monitoring and management concepts will be provided to validate the compliance concerns that can only be validated at runtime and to provide governance of the composition.

2.4 Conclusion

In this chapter, we have first briefly discussed different concepts and background knowledge about SOA, Web services, and their related technologies. We have presented SOA principles, Web services reference architecture(WSDL, UDDI, SOAP), Web service extended architecture (WS*), and REST architectural style. Then, the focus was on on the Web services composition process as an important feature to realize the objectives of our work. The main goal was to understand the key considerations that underlay the specification, the verification, the execution, and the monitoring of Web services compositions. We have discussed the different existing compositions models (Orchestration, Choreography, Coordination, and Semantic Web Services), and outlined the main requirements of the composition process in terms of non functional concerns such as time, security, QoS, privacy, etc. Following this step, a systematic study of existing composition approaches depicting their limitations in terms of being procedural, lack of integration and lack of expressiveness is made. We have classified the existing composition approaches into two main categories, which are procedural and declarative composition approaches; and we compared them according to whether or not they address the composition global

life-cycle (modeling, verification, and monitoring). Finally, we have presented lessons learned from the composition literature review and discussed our work progress beyond existing efforts. Next chapter exposes our approach to compose Web services in a declarative and integrated way.

Chapter 3

Capabilities driven Web Services Description and Composition

Contents

Providing a composition framework that supports the full round-trip process life-cycle allows organizations to not only model and automate activities and processes, but also to monitor critical aspects of the process, analyze change, and apply continuous process improvements across the organization. The process modeling can be considered as the most important stage of the composition process life cycle. It aims to provide a high-level specification which is independent from the implementation of the process. It should be easily understandable by modelers who create the process, developers responsible for implementing the process, and business managers who monitor and manage the process. Adaptation should be considered at three levels: modeling, deployment and runtime. Any change in the execution environment of a composition process may imply modifying the modeling of the process, or redeploying some part of it, or adapting the runtime infrastructure to encompass the management of the new change. Yet no solution exists for that right now and adaptation is always considered at one unique level at a time. To address this issue, we propose, in this chapter, an integrated declarative composition framework that serves as a unified framework to bridge the gap between the process design, verification and monitoring and thus allowing for self-healing Web services composition. The rest of this chapter details the ingredients and the components of the proposed approach.

3.1 Capabilities driven Web Services Description

The concept of capability is considered as one of the most important component in the enterprise information systems and the SOAs. Many initiatives tried to provide a semantic model for service description, however, they did not give the notion of capability the importance that deserves. They either confuse between service capability and invocation interface or fail in modeling service functionality at several abstraction/concreteness levels. Other efforts describe service capabilities in terms of Inputs, Outputs, Preconditions and Effects, which makes it hard to be understandable by non domain-expert users. In this scope, various efforts have been done to overcome these shortcomings and provide a new-fashioned model that enables service discovery and composition in a semantic way. For instance, [26] discussed two types of capabilities: i) an abstract capability that consists of a set of states to describe what the service can provide without going into the details; and ii) a contract capability that represents a model of the abstract capability. It includes concrete services and their related constraints that should be fulfilled to ensure a successful execution. [142] proposed a three-layer capability meta-model, which is based on a profile layer that consists of an action verb and a set of attributes. In the same context, the authors introduced in [27, 140, 143, 144], a new conceptual model that

describes functional capabilities as an action verb and a set of domain specific attributes using RDF-schema. Such description enables to represent capabilities at several levels of abstraction, from the most abstract capability, named capability category, to the most concrete one, named capability offer, by explicitly extracting the relation between capabilities (i.e., specify and extend) and linking between the different levels. However, this description is inadequate when it comes to supporting composition and planning processes. To overcome this limit, we propose an extension to the capability meta-model introduced in [27] by adding two main additional attributes, which are preconditions and effects, to capture the state/change of the world before/after executing the corresponding action. So that a service capability consists of: i) action-verb; ii) a set of domain specific attributes; iii) a set of preconditions and v) a set of effects. Formally speaking, we consider a service capability as a quadruplet *Cap*={*AV, DA, Pre, Effect*}, where:

- action Verb (AV): a concept used to define the action offered by the service;
- set of attributes (DA): a finite set of domain specific attributes, each of which is defined as a pair (attribute name, attribute value);
- Pre: a finite set of predicates that should be satisfied to execute the action;
- Effect: a finite set of facts that can be reached after executing the action and which result in a new world state.

Through such description, the semantics of the capability can be captured at two levels, which are:

1. Coarse-grain level: to handle the discovery process through a combination of an action verb, domain specific attributes and semantic links between capabilities.
2. Fine-grain level: to handle the composition process via a set of preconditions and effects.

As shown in figure 3.1, inspired from the model described in [27], the new model considers service capability as an attribute-featured entity, over which a declarative access to the offered functionality can be handled, discovered and composed at different abstraction levels. Various attribute values can be specified, such as dynamic, conditional and constrained values. Moreover, a capability can be as abstract as a category that denotes a class of actions (capability type), as it can be very concrete and corresponds to a specific consumer request (capability instance).

Navigation between different abstraction levels is established through semantic links between the existing capabilities based mainly on *Extend* and *Specifies* relation types.

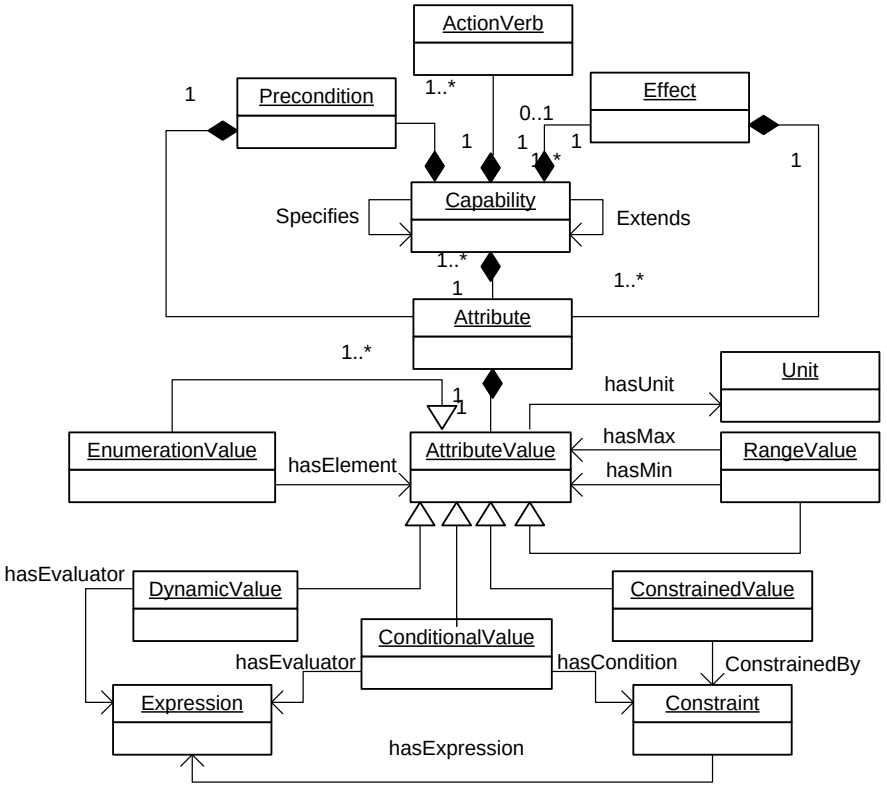


FIGURE 3.1: Capability Meta-Model

More specifically, the main relationship that can be handled between capabilities is the link established between the capability type and its concrete implementation. Indeed, it is considered as a kind of linking between a class and its instances. A capability type captures the core functionality to which the individual instances belong. We assume that the capability type remains valid when any of its instances is selected, which leads to facilitate the compensation in case of planning failure or unavailability of the selected instance at run-time. As depicted in figure 3.2, such instanciation is based on three main factors: a) ensuring that the instance specifies the service type capability; b) the preconditions of a the capability type are more specific than those of some or all of their instances and c) the postconditions are more general than those of their instances.

3.1.1 Service description

The class Service defines a software system designed to support interoperable machine-to-machine interaction over a network. Based on their atomicity, we distinguish two types of services: *atomic* and *composite* services.

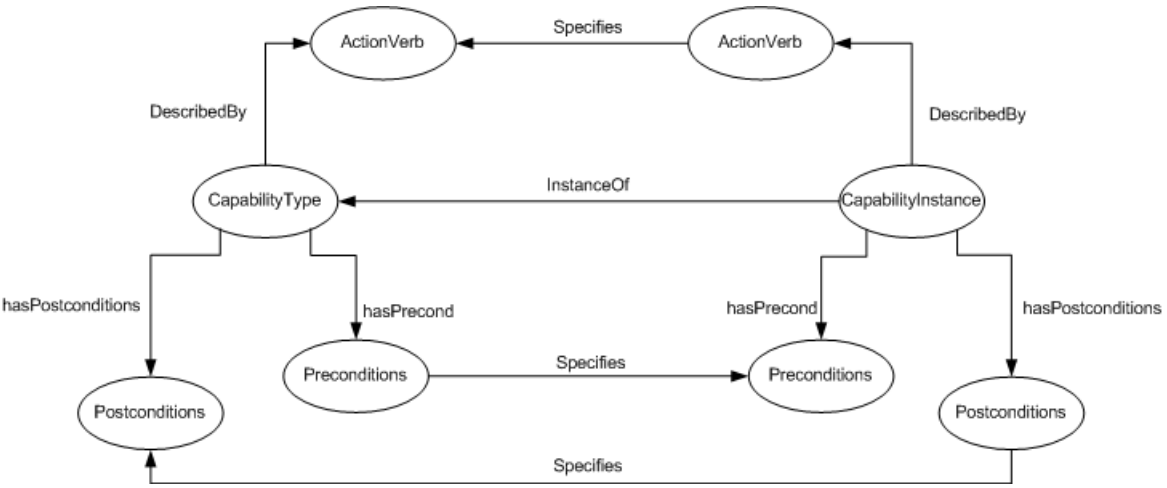


FIGURE 3.2: Relationship between Capability Type and Instance

- Atomic service: is a basic unit, which cannot be decomposed further and does not rely on any other service. In the case that the features provided by an atomic service cannot satisfy the full requirements given by the users, it needs to be composed.
- Composite service: is considered as a collaboration of existing services that can be atomic or composite to provide complex operations.

As depicted in figure 3.3, we describe a service via a set of capabilities, non-functional properties and workflow model (in case of composed service). These components are defined in detail in the following subsections.

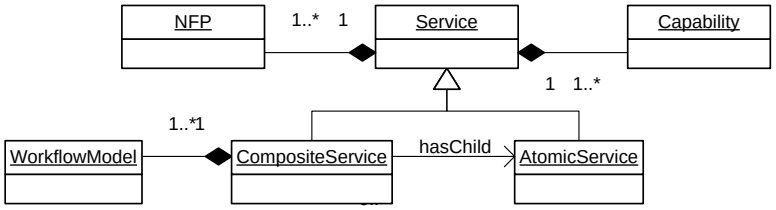


FIGURE 3.3: Service Meta-Model

3.1.2 Non-Functional Properties meta-model

As there will be many functionally-equivalent capabilities for a specific user request, we propose to consider the non-functional aspect as part of the service description in addition to its capability. Such extension leads mainly to filter and select the best offer among the existing capability offers to better fulfill customer requirements and

ensure his/her satisfaction. Separating the capability from the non functional constraints enables to reuse the capability under different constraints. Moreover, in their turn, these constraints can be re-used by different capabilities.

As shown in figure 3.4, the NFP concept contains four types of NFPs: (i) *user-generated* properties, e.g, reputation, (ii) *policy-related* properties e.g, cost and availability, (iii) *QoS* properties e.g., security and trust, response time, latency and reliability, and (iv) *contextual specifications* e.g., location or business requirements. The NFPs are described by a combination of non-functional terms (attribute, metrics) and their constraints (attribute values). Different to the functional perspective, the user should define the NFPs as well as their desired constraints to sort search results according to his preferences, which can reduce the capability offer search space.

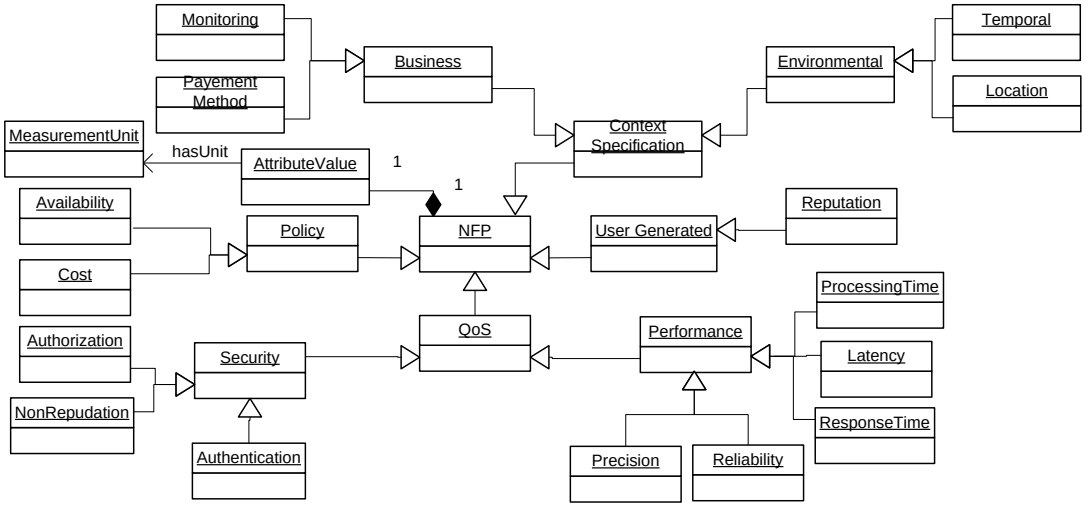


FIGURE 3.4: Non-Functional Properties Meta-Model

3.1.3 Workflow meta-model

The workflow model includes the control-flow patterns used to represent the structural knowledge of the composite service and the connections between its different components. In general, we use workflow patterns to describe a service composition either as a block structure or as a graph model including a set of nodes interconnected with arcs. Due to its ability to describe different representation of the same composite service at different granularity levels, only block model concept was considered in our conceptual model. As illustrated in figure 3.5, the block-patterns can be classified into five main patterns:

- Sequence: is a list of ordered elements that have an order and contain a WorkflowModel.
- ParallelSynchronize: combines a synchronizing pattern and parallel split; and consists of a set of branches. Each branch contains a WorkflowModel.
- ExclusiveChoice: is an exclusive (choice, merge workflow patterns) pair that consists of a set of ConditionalBranches constrained by a Condition. At runtime only one ConditionalBranch of an ExclusiveChoice can be active.
- MultipleChoice: is a (multi-choice, synchronizing merge workflow patterns) pair, which consists of a set of ConditionalBranches. At runtime there can be many active ConditionalBranches of a MultipleChoice.
- Loop: is a block structure that executes until a stop condition becomes true.

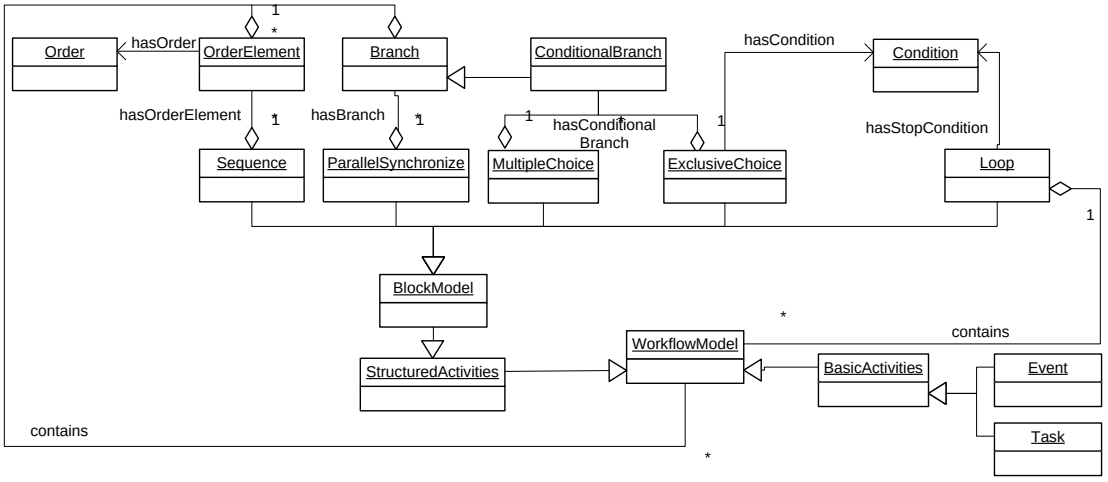


FIGURE 3.5: Workflow Meta-Model

3.1.4 Illustrative example

To illustrate the notion of capability-based service description, we use the *Programmable Dinner (PD)* scenario proposed in [145]. The idea is to organize an event by choosing a movie title, a theater that plays the movie and inviting friends to select a restaurant using a map view of some proposed restaurants that exist in a predefined location. Given a movie title, a location, a date and a list of friends, the system has to provide the weather forecast during the given date, identify the possible restaurants existing in the given location and suggest a list of movies that are similar to the movie title proposed. The selected movie will be taken by the system to provide a list of theaters that play this movie. By choosing the theater, an invitation will be sent to a given list

of friends. This invitation includes all the details of the event, with a list of restaurants that are presented as markers in a Map showing their positions. Therefore, the following requirements should be considered:

- *Rq0:* The goal is to send an invitation to a list of friends with a list of restaurants in a given location marked in maps. The invitation includes all the details of the event such as date, location, movie title and theater name.
- *Rq1:* The user should provide his actual location, the list of friends he/she wants to invite, a date and a movie title.
- *Rq2:* Based on the given location and date, the system has to provide a weather forecast.
- *Rq3:* W.r.t the location and the weather, the system has to provide a list of restaurants that exist in the given location.
- *Rq4:* The identified restaurants have to be represented as markers in a map.
- *Rq5:* Based on the given movie title, the system has to provide a list of similar movies.
- *Rq6:* Taking the list of similar movies as inputs, the system provides a list of theaters that play these movies at the required date.
- *Rq7:* A mail that includes the whole details of the event accompanied with a map of suggested restaurants has to be sent to the given list of friends.

Rq1 represents the initial state of the system, while *Rq2-Rq7* represent the abstract capabilities that are required to reach the goal described in *Rq0*. Listing 3.1 illustrates the required abstract capabilities, serialized in RDF.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix cap: <http://.../ontology/capability#> .
@prefix md: <http://.../DoodleMap-Domain#> .
md:FindPlacesCap a cap:Capability;
 cap:hasActionVerb md:FindPlaces;
 md:hasLocation md:Location;
 md:hasCriteria md:Criteria;
 cap:hasPracondition md:FindPlacesPrec;
 cap:hasEffect md:FindPlacesEffect.
md:FindPlacesPrec a cap:Expression;
  cap:expValue "and GeoLocation ?location
                        Property ?criteria."
md:FindPlacesEffect a cap:Expression;
        cap:expValue "and Location ?place
                              hasCriteria ?place ?criteria."
md:hasLocation a cap:attribute.
md:hasCriteria a cap:attribute.
md:CreatePollCap a cap:Capability;
 cap:hasActionVerb md:CreatePoll;
 md:hasPoll md:Poll;
 cap:hasPracondition md:CreatePollPrec;
 cap:hasEffect md:CreatePollEffect.
md:hasPoll a cap:attribute.
md:CreatePollPrec a cap:Expression;
cap:expValue "and hasName md:Poll ?Name
 hasTitle md:Poll ?Title
 hasOption md:Poll ?Op
 EqualTo ?op ?Location."
md:CreatePollEffect a cap:Expression;
cap:expValue "and hasData md:Poll ?ps
          PollStream ?ps."
md:CreateMarkersCap a cap:Capability;
 cap:hasActionVerb md:CreateMarkers;
 md:hasPlaceList md:PlaceList;
 cap:hasPracondition md:CreateMarkersPrec;
 cap:hasEffect md:CreateMarkersEffect.
md:CreateMarkersPrec a cap:Expression;
  cap:expValue "PlaceList ?pl".
md:CreateMarkersEffect a cap:Expression;
  cap:expValue " MarkerList ?ml".
md:hasPlaceList a cap:attribute.
md:CreateMapWithMarkersCap a cap:Capability;
   cap:hasActionVerb md:CreateMapWithMarkers;
   md:hasMarkersList md:MarkerList;
   cap:hasPracondition md:CreateMapWithMarkerPrec;
   cap:hasEffect md:CreateMapWithMarkerEffect.
md:CreateMapWithMarkerPrec a cap:Expression;
   cap:expValue "MarkerList ?ml".
md:CreateMapWithMarkerEffect a cap:Expression;
  cap:expValue "and Map ?mp
                   Includes ?mp ?ml".
md:hasMarkersList a cap:attribute.
```

Listing 3.1: Abstract Capabilities of Programmable Dinner Scenario

Each abstract capability can be implemented by one-to-many executable capability instances. Listing 3.2 represents two alternative instances for the abstract capability *FindPlacesCap*.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix cap: <http://.../ontology/capability#> .
@prefix md: <http://.../DoodleMap-Domain#> .
@prefix dbpedia: <http://dbpedia.org/ontology#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
md:FindPlacesByLocationCap a cap:Capability;
cap:hasActionVerb md:FindPlacesByLocation;
md:hasLocation dbpedia:Belgium;
md:hasCriteria md:FamousRestaurant;
cap:hasPracondition md:FindPlacesPrec;
cap:hasEffect md:FindPlacesEffect.
md:FindPlacesPrec a cap:Expression;
cap:expValue "and GeoLocation ?location
Property ?criteria."
md:FindPlacesEffect a cap:Expression;
cap:expValue "and Location ?location
hasPlace ?location ?place
hasCriteria ?place ?criteria."
dbpedia:Belgium a md:Location.
md:FamousRestaurant a md:Criteria.
md:FindPlacesByCoordinatesCap a cap:Capability;
cap:hasActionVerb md:FindPlacesByCoordinates;
md:hasLocation dbpedia:Brussels;
geo:lat md:Latitude;
geo:long md:Longitude;
md:hasCriteria md:FamousRestaurant;
cap:hasPracondition md:FindPlacesByCoordinatePrec;
cap:hasEffect md:FindPlacesByCoordinateEffect.
md:FindPlacesByCoordinatePrec a cap:Expression;
cap:expValue "and GeoLocation ?location
                              SearchCoordinate ?cord
                              Property ?criteria."
md:FindPlacesByCoordinateEffect a cap:Expression;
cap:expValue "and Location ?cord ?place
                               hasCriteria ?place ?criteria."
dbpedia:Brussels a md:Location.
md:FamousRestaurant a md:Criteria.
md:FindPlacesByLocation rdfs:subClassOf md:FindPlaces.
md:FindPlacesByCoordinates rdfs:subClassOf md:FindPlaces.
md:hasLocation a cap:attribute.
md:hasCriteria a cap:attribute.
md:hasLocation a cap:attribute.
md:hasCoordinate a cap:attribute.
```

Listing 3.2: Alternative instances for FindPlace Capability serialized in RDF

3.2 Integrated Framework for Web Services Composition

Using the capabilities based model discussed above, we propose a declarative framework for Web services composition based on the three stages of abstraction, concretization and monitoring as depicted in figure 3.6.

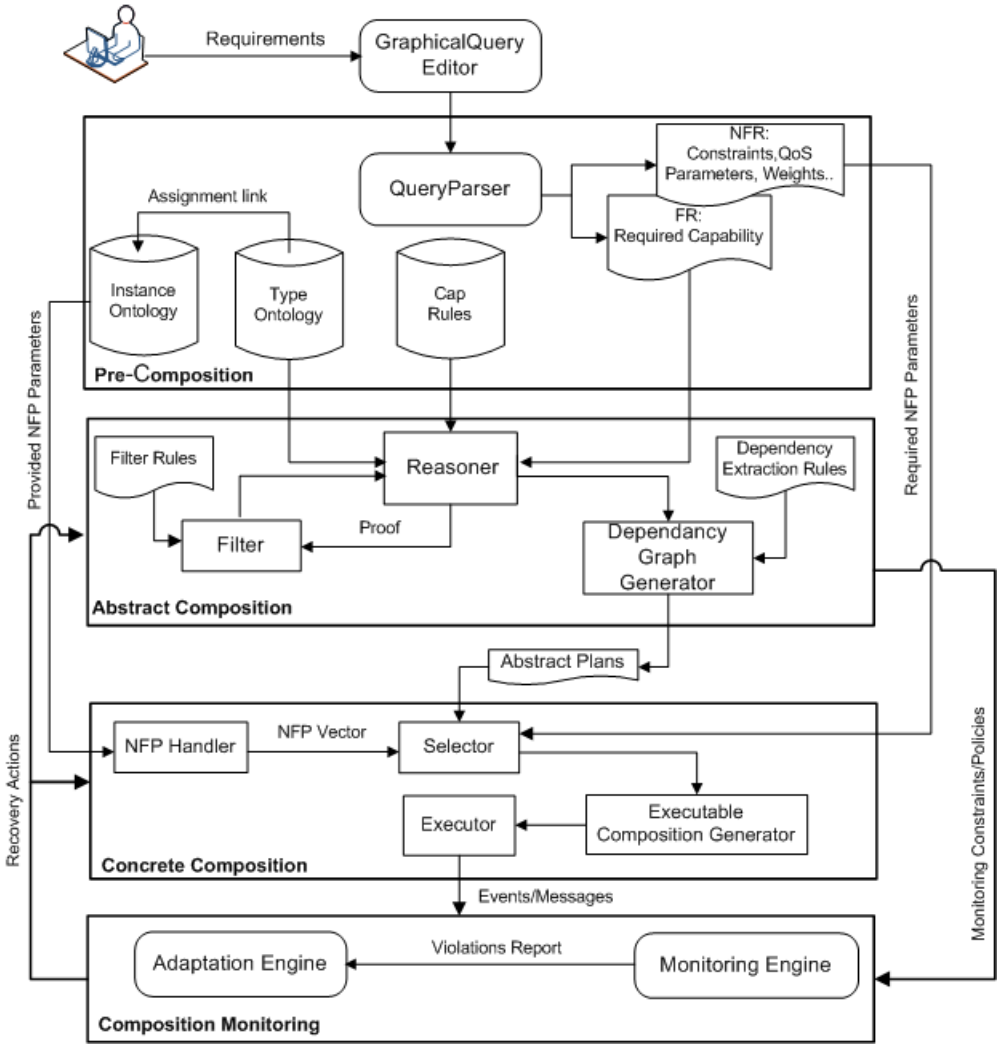


FIGURE 3.6: Modeling Framework for Web Services Composition

Abstraction refers to the high-level description of the desired service (goal), which drives the identification of an appropriate composition strategy (abstract plan). The concretization stage aims to find a solution (executable plan) to the composition process respecting the constraints and requirements specified in the model resulting from the design phase. Monitoring deals with the actual execution of the composite service and is responsible for monitoring the execution and recording violation of any requirement of the goal service at runtime. In case of detecting one-to-many constraints violation, recovery actions can take place on both abstract and concrete stages. This framework provides an easy way to specify functional and non-functional requirements of composite

services in a precise and declarative manner, and guides the user through the composition process. The staged approach is designed keeping in mind the best knowledge engineering practices of modularity, conciseness, and scalability, while providing a fair amount of control to the composition process. It focuses on the specification of *what* level without having to state the *how*, which enables to preserve the autonomous nature of interacting services and represent highly expressive interaction models without over-specifying them. In what follows, we detail and expose the key features of the proposed architecture.

3.2.1 Pre-Composition phase

3.2.1.1 Goal Specification

Users/Requesters provide the high-level description of the desired service (goal) using a user-friendly interface that enables end users to specify and express their requirements and preferences by following some instructions to build the query. A query parser is integrated to parse and validate the query by checking syntactic correctness and decoupling the functional requirements from the non-functional parameters (such as QoS properties). Due to the high-level specification of the desired composite-service, our framework guides the users for iterative refinement of the goal specification.

Definition 3.1. Query Specification: Given an ontology domain O, a user query Q consists of an action verb $Q.av \subseteq O$, a set of attribute pairs (name, value) $Q.att \subseteq O$, a set of preconditions $Q.pre \subseteq O$, a set of effects $Q.eff \subseteq O$, and a set of quality of service constraints $Q.qos = \{(q_1, v_1, w_1), (q_2, v_2, w_2), ..., (q_k, v_k, w_k)\}$, where $q_i (i = 1, 2, ..., k)$ is a quality criterion, v_i is the required value for criterion q_i, w_i is the weight assigned to this criterion such that $\sum_{i=1}^{k} w_i = 1$, and k the number of quality criteria involved in the query. Note that a quality criterion can be either negative, i.e. the higher the value the lower the quality, or positive, i.e. the higher the value the higher the quality.

3.2.1.2 Service Specification

Given a Service Repository D of all the available services, the representation of each service $S \in D$ is given by definition 3.2.

Definition 3.2. Service Specification: A service $s \in D$ is composed of an action verb $S.av \subseteq O$, a set of attributes $S.att \subseteq O$, a set of preconditions $S.pre \subseteq O$, a set of effects $S.eff \subseteq O$ and a set of provided quality criteria $S.qos = \{(q_1, v_1), (q_2, v_2), ..., (q_k, v_k)\}$, where $q_i (i = 1, 2, ..., k)$ is a quality criterion, v_i is the value associated to criterion q_i provided by the service, and k is the number of the involved criteria.

In order to enable working with large collections of Web services, we distinguish between Web services types and instances. A Web Service Type is considered as an abstract capability, which includes the main functionality offered by the service. While Web services instances are the to-be-invoked services, which consist of one-to-many capabilities and a set of non-functional properties. This separation may lead to a significant reduction of the search space, and therefore provides scalability to the composition process. For example, as mentioned in [146], suppose that we have α service types in the service repository and each service type has β service instances, if we consider the composition approach that does not use the distinction between types and instances, then the required search space to find an executable plan is $O((\alpha \times \beta)^n)$. However, if we deal with a separation strategy, this space research is significantly reduced to $O(\alpha^n) + O(\beta^n)$. For example, For $\alpha = 10$, $\beta = 5$, $n = 9$, we have $(\alpha * \beta)^n = (10 * 5)^9$,while $\alpha^n + \beta^n = 10^9 + 5^9$.

However, it is very important to identify the relationship that a Web service type has with its various instances. The service type represents the core functionality of the service, while the instances represent this functionality with different attribute values and non-functional parameters. Following this separation between services types and instances, we have introduced two types of ontologies, which are *Type Ontology* and *Instance Ontology*. The Type ontology describes services domain at a high level of abstraction. It contains fine grained definitions of different types of domain-specific services; the possible abstract capabilities and their related action verbs, domain-specific attributes, preconditions and effects; and their non-functional attributes. The Instance Ontology is a concrete ontology that gives a fine grained view of the existing service instances, their capabilities, their preconditions and effects; and the possible values each functional and non-functional attribute can have. The instance ontology is generated using the Instance Ontology Builder (IOB). As shown in figure 3.7, IOB consists of a Jena-Based parser and an ontology builder. The Jena API [1] takes a set of semantic Web services as inputs and stores them in a single data structure. Given this generated data structure and the domain ontology, the ontology builder provides a corresponding Instance ontology. The Instance ontology is a concrete representation of the available instances organized in a hierarchical structure reflecting the relationships between service capabilities according to the domain ontology.

3.2.1.3 Constraints

We distinguish between behavioral constraints (orchestration and choreography aspects), non-functional constraints, and execution constraints. Behavioral constraints are used for plan nodes instantiation and process execution. They describe how multiple services

[1] http://jena.apache.org/

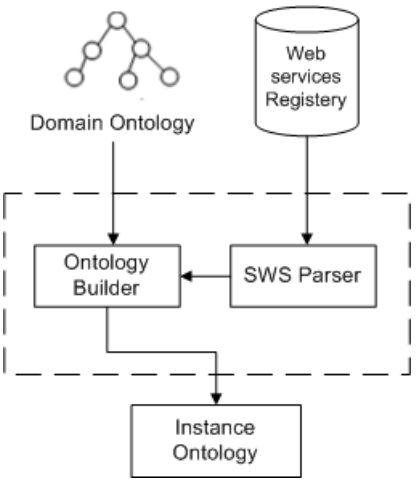


FIGURE 3.7: Instance Ontology Builder

can interact by exchanging messages including the business logic and execution order of the interactions (sequence, parallel, loops, etc.) and the dependencies between services (before, after, if-then-else, choice,..). Non-functional constraints are constraints exhibited over the functionality of the services. They include temporal and spatial availability, service quality, security, trust and ownership. Execution constraints specify the constraints to be validated at run-time. These constraints are implemented as monitors, which have an associated monitoring event/condition and actions to perform if the condition to be monitored is encountered. All these constraints are specified in terms of capabilities and attributes, and can be added to the process specification both at design time and run-time.

3.2.2 Abstract Composition

The abstract composition is responsible for automatically generating an abstract capability-based plans that meet the functional requirements of the user using a novel *Proof-based* mechanism [147, 148]. Over such mechanism, providing a composition that satisfies the user goal is reduced to generating a proof that supports the goal. The provided proof includes a set of capabilities required to transform the world from the initial state to another state that satisfies the required functionality. Unlike classical planners, proof-based composer does not require new reasoning and composition tools. A generic semantic Web reasoner that supports rule language is good enough to generate the appropriate proof. Moreover, the generation of proof indicates that a solution exists for a given composition problem. To meet the goal G, the generated plan is a sequence of actions $\{A_p\} = \{A_0, ...A_i, ...A_n\}$, so that, $\forall j; 0 \leq j \leq n, preconditions(A_i, S_i) \implies$

$effects(A_i, S_{i+1})$ where S_j denotes the j^{th} state in which A_j will be called, S_{j+1} is the successor state in which A_{j+1} will be called, and $S_0 \cup effects(A_p, S_p) \models G$.

Algorithm 1 describes the composition strategy used by proof-based reasoner to generate the abstract plans.

Algorithm 1: Abstract Composition Algorithm

Input: A set of abstract capabilities AS_T, a goal G, a set of abstract capabilities description formulas $\{CR_i\}$ true in initial state S_0.
Output: Composition plan **or** nil as failure.

```
begin
    procedure FindGoal(S_c, S_f, G)
    begin
        if G is already true in the target state S_f, where S_c ≺ S_f then
            return S_f
        ;
        else
            T ← set of abstract capabilities ∈ AS_T to be called in the current state S_c
            such that for each AS_t ∈ T, S_c ⊨ AS_t.prec and G unifies with AS_t.eff ;
            while T is non-empty do
                foreach (AS_t ∈ T) do
                    Unify the attributes, preconditions, effects of AS_t with G;
                    If successful with AS_t, Mark the effects as being true in state S_f and
                    the preconditions as being true in state S_c, where S_c < S_f , and AS_t
                    was called in S_c.;
                    Return S_c
                End While
        End If
    End procedure
P ← the set of preconditions of AS_T;
while P is non-empty do
    foreach (AS_t ∈ P) do
        P ← P ∪ AS_t.att;
        Select the first precondition in P ;
        S_r = FindGoal(S_0, S_i, AS_t.pre) ;
        while S_r results in new states and unifications do
            propagate new instantiations of variables in preconditions and effects of AS_t
            in states S_i and S_f ;
            S_r = FindGoal(S_0, S_i, AS_t.pre) ;
        if the last call to FindGoal failed then
            return fail ;
        P ← P − {AS_t} ;
    Return S_f and all marked states ;
```

This algorithm has been proven to be sound and complete. Indeed, every plan generated by this algorithm is a service that meets all the query requirements. However, since the constraints enable to specify only the boundaries to the composition, the reasoner may

return multiple plans, and therefore the composition system is given the possibility to pick and choose among them based on some optimization criteria (e.g., number of service candidates) and also provide better failure resiliency (e.g., if a plan fails to get concretized, the system can try a different plan without going through the logical composition stage again).

The main ingredients of this stage are described as follow:

3.2.2.1 Reasoner

As shown in figure 3.8, the reasoner component behaves as a checker as well as a planner that takes as inputs the knowledge base, the initial state, a set of rules and the goal, and generates the proof of the goal.

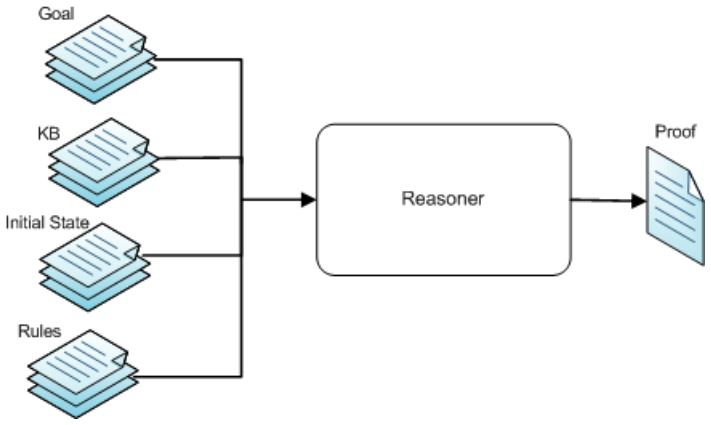


FIGURE 3.8: Proof based Reasoner

3.2.2.2 Filter

Given the generated proof and a set of rules, the filter component re-invokes the reasoner to provide a new proof by considering only abstract capabilities that constitute the required abstract plan to meet the user goal. This component is responsible for avoiding redundancy from the plan by identifying service types with potential relevance to the goal. Such services can either be invoked using the initial state that satisfies their preconditions, or can potentially contribute to the goal using their effects.

3.2.2.3 Dependency Graph Generator (DGG)

The core functionality of this component is to extract the dependencies among services and represent them as a directed acyclic graph, where the candidate services represent

its vertexes and the dependency relationships between each two services represent the edges between these vertexes. Algorithm 2 is used to deliver the composition plan with the execution order of its service candidates.

Algorithm 2: Ordered Plan Generation Algorithm

Input: DADG: Directed Acyclic Dependency Graph
Output: L: list of ordered plans
begin
  Queue Q=$\emptyset$;
  List L= {} ;
  foreach *($V_i \in DADG.Vertices$)* **do**
    $in_i \leftarrow V_i.incomingsEdges$;
    $out_i \leftarrow V_i.outgoingsEdges$;
    if in_i=*0* **then**
      Q$\leftarrow Q \cup V_i$;
  while $Q \neq \emptyset$ **do**
    foreach *($V_i, V_{i+1} \in Q$)* **do**
      if *($out_i > out_{i+1}$)* **then**
        Dequeue V_i from Q ;
        Push V_i in L ;
        Remove all outgoing edges of V_i from DADG ;
      else
        Dequeue V_{i+1} from Q ;
        Push V_{i+1} in L ;
        Remove all outgoing edges of V_{i+1} from DADG ;
  Return L ;
  End

3.2.3 Concrete Composition

An abstract composition plan is a sequence of abstract capabilities that specify the temporal ordering of different actions, whose execution leads to the goal. This plan is then used by the concrete phase to *instantiate*, *verify* and *execute* the composition process. Below, we present the role and mission of each step.

3.2.3.1 Instantiation

The Instantiation step aims to find a solution to the composition process respecting the constraints and assumptions specified in the design phase. The plan generated by the abstract composition stage is considered as a template for the composite service, which in conjunction with the non-functional parameters specifications, drives the process of matching each service type to a corresponding service instance. Usually, for each abstract

capability existing in the plan, there is a set of alternative Web services instances with similar capability and different functional and non-functional attribute values. This leads to the general optimization problem of how to select Web services for each capability so that the overall functional and non-functional requirements of the composition are satisfied with most preferred compositions. Thus, this stage not only handles Multiple Criteria Decision Making (MCDM), but also guarantees the non-functional requirements at the same time.

As depicted in figure 3.6, given the first-stage generated plan and the non-functional requirements of the user, the instantiation phase will be invoked and the following process will take place: the *NFP handler* collects the non-functional properties from the existing service instances and provides an NFP vector for each instance. Then, taking the generated NFP vectors as inputs, the *selector* picks exactly one instance from the set of matched instances for each abstract capability existing in the plan w.r.t the required NFP parameters using the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) method [149, 150], which is described in the subsection **??**. Note that if there is no matching with the global constraints, the framework selects other instances near to the optimal instances. If no alternative instance is available, the framework automatically replace the abstract plan P_i with another plan P_{i+1}.

TOPSIS

TOPSIS is a theoretical rigorous method that is considered as one of the best techniques devoted for solving Multiple Criteria Decision Technique Method (MCDM) problems [151]. It considers qualitative and quantitative criteria with two hypothesis:

- Ideal alternative: denotes the best alternative for all attributes values.
- Negative ideal alternative: denotes the worst alternative for all attribute values.

Assuming that we have an evaluation $m*n$ matrix M with m alternatives in rows and n attributes or criteria in columns and we have the score x_{ij} of each alternative w.r.t each criterion. Let B and N be a set of benefit (more is better) and negative (less is better) attributes, respectively. Given a set of weights $W=\{w_0, .., w_j, ..., w_n\}$ for each criterion, M, B and N as inputs, TOPSIS selects the alternative that is the closest to the ideal solution and farthest from negative ideal alternative using the following steps:

1. Create a normalized decision matrix: using the formula *f1*, this step converts different attribute dimensions into normalized attributes, which leads to apply

comparisons across criteria.
f1: $r_{ij} = x_{ij}/(\sum x_{ij}^2)$, *where i=0, ...,m ; j=0, ... , n.*

2. Construct the weighted normalized decision matrix: this step consists in multiplying each column of the normalized decision matrix $M_{Nomalized}$ by its associated weight $w_j \in W$. Each element of the new matrix $M_{WeightedNomalized}$ is:
f2: $v_{ij} = w_j * r_{ij}$.

3. Determine the Positive and Negative ideal solutions (PIS and NIS): using two formulas (f3 and f4), this step allows to identify PIS and NIS as follows:

   - Ideal solution (PIS):
   f3: $A^* = \{v_1^*, ..., v_n^*\}$, *where* $v_j^* = \{max(v_{ij})$ *if* $j \in B; min(v_{ij})$ *if* $j \in N\}$
   - Negative ideal solution (NIS):
   f4: $A^- = \{v_1^-, ..., v_n^-\}$, *where* $v_j^- = \{min(v_{ij})$ *if* $j \in B; max(v_{ij})$ *if* $j \in N\}$

4. Calculate the separation measures for each alternative: this step consists in calculating the distance between the target alternative Alt_i and the positive ideal solution; and the distance between the alternative Alt_i and the negative ideal solution as follows:

   - The separation from the ideal alternative is:
   $S_i^* = [\sum (v_j^* - v_{ij})^2]^{\frac{1}{2}}$, where i = 1, ..., m.
   - The separation from the negative ideal alternative is:
   $S_i^- = [\sum (v_j^- - v_{ij})^2]^{\frac{1}{2}}$, where i = 1, ..., m.

5. Rank the alternatives by calculating the relative closeness to the ideal solution:
$C_i^* = S_i^-/(S_i^* + S_i^-)$, *where* $0 \leq C_i^* \leq 1$.

QoS based Selection

Some QoS properties have been emphasized in previous research works [152, 153] as the quality evaluation criteria of Web service. These propeties are:

1. Price ($q_{pr}(s)$) of a service s is the fee that its requesters have to pay for invoking it.

2. Duration ($q_{du}(s)$) of a service s measures the expected delay between the moment when a query is sent and the moment when the result is received. It is calculated by summing the service processing time, which is given by the service provider as a fixed value or as a method to inquire about it; and the average of transmission times of past executions;

3. Availability ($q_{av}(s)$) of a service s is the probability that it is accessible.

4. Reputation ($q_{rep}(s)$) of a service s is a measure of its trustworthiness. It is calculated by the average of different end users' ranged ranking ($[0, 1]$) on the service;

5. Successful execution rate ($q_{rat}(s)$) of a service s is the percentage that service requests are responded. Since the successful execution rate is measured in service-available state, it is slightly different from the availability. It is counted as the number of successful executions divided by the total number of service trials.

Assuming that QoS measures are quantitative and QoS values are static, QoS property values of component Web services can be aggregated using aggregation formulas presented in table 3.1.

Criterion	**Function**
Reputation	$A_{rep} = 1/n \sum_{i=1}^{n} q_{rep}(si)$
Price	$A_{pr} = 1/n \sum_{i=1}^{n} q_{pr}(si)$
Duration	$A_{du} = 1/n \sum_{i=1}^{n} q_{du}(si)$
Availability	$A_{av} = \prod_{i=1}^{n} q_{av}(si)$
Successful Execution Rate	$A_{rat} = \prod_{i=1}^{n} q_{rat}(si)$

TABLE 3.1: Aggregation functions for computing composition QoS

The Overall Reputation model is given by the average of all constituent services' reputation. The Overall Price, in turn, is calculated simply by summing the price of all constituent services. The Overall Duration model is calculated as the sum of all constituent services duration of a given composition. Availability and Successful Execution Rate models are defined as a simple product of factor of probability without taking into account whether or not a service is part of a critical path [152].

A matrix M is created, in which each row M_i corresponds to an instance CS_i while each column corresponds to a quality criteria q_j. Each instance score is normalized using the formula *f1*, where each score (r_{ij}) denotes the normalized value of QoS criterion q_j associated with a candidate instance CS_i and then multiplied by a given weight using the formula *f2*. Once the matrix M was normalized and weighted, for each service type included in the abstract plan, the steps (c), (d) and (e) are applied. The instance with higher score is selected. Then, the selected optimal instances are aggregated to determine the overall QoS of the composite service using the aggregation formulas presented in table 3.1. The NFP-aware instantiation phase is described in algorithm 3.

Algorithm 3: QoS-oriented web service composition algorithm

Input: NFR: Non-Functional requirements, W: list of NFP weights, P: abstract composition plan

Output: $P_{Executable}$: Executable plans

begin

NfpVector V $\leftarrow \emptyset$;
List $L_{instance} \leftarrow \{\}$;
Matrix M $\leftarrow \emptyset$;
Matrix $M_{WeightedNormalized} \leftarrow \emptyset$;
BIS $\leftarrow 0$; NIS $\leftarrow 0$; List res $\leftarrow \{\}$;
Instance $CS^* \leftarrow \{\}$;
foreach *($AS_i \in P$)* **do**
  $L_{instance} \leftarrow Instanciate(AS_i)$;
  foreach *($CS_i \in L_{instance}$)* **do**
    $qos_i \leftarrow NfpHandler(CS_i)$;
    $V \leftarrow V \cup (CS_i, qos_i)$;
  while $V \neq \emptyset$ **do**
    foreach *($v_i \in V$)* **do**
      $M \leftarrow M \cup v_i$;
      $V \leftarrow V \backslash \{v_i\}$
    End
  foreach *($v_i \in M$)* **do**
    $M_{WeightedNormalized} \leftarrow M_{WeightedNormalized} \cup (NormalizedAttribute(v_i) * w_i)$;
  $BIS \leftarrow IdealSolution(M_{WeightedNormalized})$;
  $NIS \leftarrow NegativeIdealSolution(M_{WeightedNormalized})$;
  foreach *($v'_i \in M_{WeightedNormalized}$)* **do**
    $S_i^* \leftarrow CalculateDistance(v'_i, BIS)$;
    $S_i^- \leftarrow CalculateDistance(v'_i, NIS)$;
    $C_i^* \leftarrow S_i^- / (S_i^* + S_i^-)$;
    $res \leftarrow res \cup (CS_i, C_i^*)$;
  $CS^* \leftarrow getInstance(Max(res))$;
  $P_{Executable} \leftarrow P_{Executable} \cup CS^*$;
CheckGlobalOptimization ($P_{Executable}$, NFR, $QoS_{aggregationFunctions}$) ;
Return $P_{Executable}$;
End

3.2.3.2 Verification

The verification of the composition constraints can be done either at design time (a-priori) or after runtime (a-posteriori) to repair design errors, and formally verify whether the process design does have certain desired properties. The a-priori verification is important for compositions because we need to check if the specified behavior is consistent, which is not a trivial task as soon as a composition process manages complex service dependencies. The a-posteriori verification is also important to provide knowledge about

the context of and the reasons of discrepancies between abstract models and related instances. This kind of verification is required since some interactions between Web services may be dynamically specified at runtime, causing unexpected interactions with other services, and making the a-priori verification method insufficient as it only takes into account static aspects.

Verification enables to detect conflicts such as deadlocks and identify hard constraints to be relaxed and side effects such as data expiry. The properties that can be checked include (but not limited to) temporal constraints, data retention such as the maximum time interval about data validity, behavioral constraints, and security requirements. These properties are handled by adding constraints to the composition design. Violations of these properties lead also to conflicts in the composition process specification. These conflicts can be broadly categorized into the syntactic and semantic categories. The syntactic conflicts result due to erroneous process specification and not following the syntactic rules for process specification, such as not following the naming conventions for instances. The reasoner allows identifying the syntactic errors providing error description that can be used to rectify the syntactic errors. The semantic conflicts include deadlocks, hard and conflicting constraints. For instance, the planner may not be able to find any plan which can satisfy the specification of the composite service. This may be due to the services in the repository, which are insufficient to create the specified composite service, and therefore an exception indicating the problem can be generated. The plan can also provide partial solution that is closest to the specification. Analysis of this partial plan can help in identifying what can be composed with existing services and what are the missing functionalities that need to be developed. Other types of failures can occur when a binding may not exist for a given service in the plan or some requirements specified may not be satisfied. If a binding does not exist for a service in a selected plan, a different plan that (possibly) does not use the particular service is selected. If no such plan exists, the user is asked to regenerate his goal without using that service. Similarly, in case some requirements are not satisfied, the user could be asked to relax some of these constraints that could not be satisfied. We have implemented these strategies as recovery policies.

3.2.3.3 Execution

Finally, the execution step deals with the deployment of the composition onto a runtime infrastructure. Now that each node in the selected plan is bound to a concrete Web service instance, a generator produces a concrete workflow that can be deployed onto a runtime infrastructure, to realize the composite service. For that, we first generate the WSDL description (name, interface, port types) for the composite service. Then, we

define partner link types to link the component services, and proceed to the generation of the composition flow (BPEL flow for instance). The selected plan gives the invocation order. We use an Eclipse Modeling Framework (EMF) model of BPEL (WSDL) that is automatically created from a BPEL (WSDL) schema[2] . The model provides in-memory representation of constructs and support for persistence to files (serialization) and loading from files (de-serialization). BPEL and WSDL manipulation become significantly simplified with the according EMF models. Execution events/messages are handled by the monitoring engine.

3.2.4 Composition Monitoring

Monitoring is event-based mechanism and can be easily performed since the same constraints are used for the design, the instantiation, and the execution of the composition. Properties to be monitored are specified by the user and added to process specification at both design and execution time. These properties include functional constraints (invocation and execution order), non-functional properties (security, QoS...), temporal constraints (response time, invocations delay), data constraints (data availability, validity and expiry). Services providers can also specify additional assumptions about the composition process in terms of events extracted from the specification. Runtime deviations and inconsistencies are monitored by using an extension of EYE reasoner.

The monitoring framework is depicted in figure 3.9.

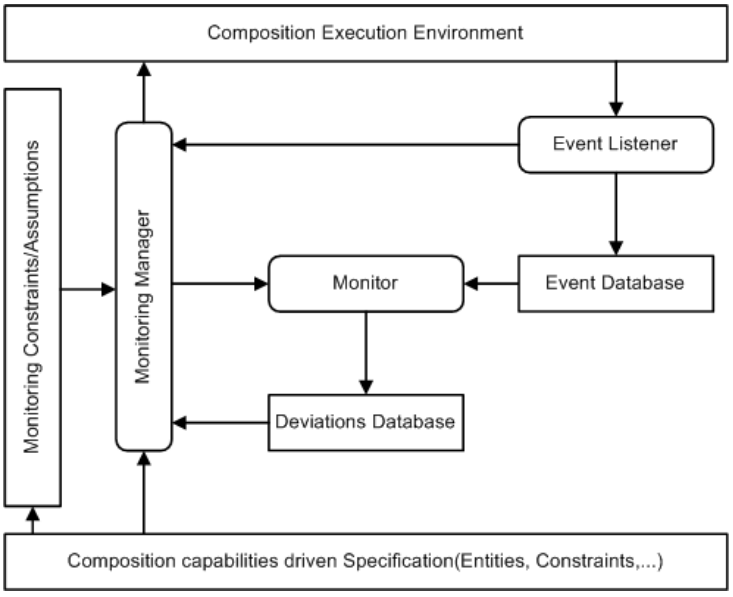


FIGURE 3.9: Monitoring Framework

[2]http://www.eclipse.org/emf

The monitoring process is initiated by the monitoring manager after receiving a request to start a monitoring activity as specified by a monitoring policy. First, it checks if the requested constraint or property can be monitored or not. This checking is based on the composition process identified in the policy, and the event reporting capabilities indicated by the type of the execution environment of the composition process. If the requested constraint can be monitored, the monitoring manager triggers an event listener to capture events from the composition execution environment and passes to it the events that should be collected. It also sends to the monitor the specification of the constraint to be checked. The event listener is responsible for collecting events from the execution environment. For each event it receives, it identifies its type and records it in an event database in case where this event is relevant to the property being monitored. Else, the received event is ignored. Then, events recorded in the event database during the execution of the composition process are passed to the monitor in order of their occurrence. Events generated by the composition process are of type process startup, termination, and messages exchanged (send, receive, reply, assign...) between the cooperating services and the composition process itself. We associate a context to each event indicating the event meta-data (source, type (inputMessage, outputMessage), timestamp...). The monitor may also derive some possible events or patterns that may have happened but not recorded (given the process specification). Following this step, it checks if these events are compliant with the properties being monitored. In case of inconsistency, the monitor records deviations in a deviation database. This deviation database is periodically polled by the monitoring manager to check for deviations to be reported. When requirements variations are detected, a deviation notification is sent to the composition manager. This notification indicates: (i) the requirement that has been violated, (ii) the malfunctioning service(s) that violated it, and (iii) diagnostic information regarding the violation. Based on the type of the deviation notification and service monitoring policies, recovery policies will be triggered.

The monitor has a set of activation conditions and associated actions. The monitor activation conditions include events contexts (temporal, spatial, semantic), policies, event conditions, directives (for reporting monitoring violations to the actions stage), timeValue(to delay the reporting in an attempt to give the service some time to recover from the violation). The monitor actions include terminate/ignore/reinstantiate and other directives.

Violations of monitoring properties are analyzed and used to initiate different recovery actions such as ignore the violation, terminate the process, re-invoke or substitute the service, find an alternative solution based on current process state or backtrack to some previous state and then seek an alternative solution. A substitute recovery action is described below. Based on the type of the deviation notification and service monitoring

policies, the composition system will generate queries to replace malfunctioning services (services that become unavailable during execution). The specification of the violated constraints serves to define these queries. A query contains structural and behavioral properties of the required services. The structural part of a query represents the interface of the required service (from local registry of the components services), while the behavioral part is specified as a conjunction of paths. Each path contains information about messages sent and received by the composition process, predicates, temporal constraints, and service states. The violated requirements are transformed to query paths according to set of rules. When the query is specified, we perform a structural matching (based on data types and signatures of service operations) and semantic matching to match between a service requested by a query and the services described in a service registry. This run time discovery process can be implemented as a Web service which exposes one operation that takes service queries as input and returns one or more candidate services that match with the query.

3.3 Conclusion

In this chapter, we have investigated a novel approach for Web services composition that integrates different stages of the process life-cycle in an unified and declarative way to bridge the gap between the process design, verification and monitoring. The proposed framework, based on the three stages of abstraction, composition, and monitoring, provides an easy way to specify functional and non-functional requirements of composite services in a precise and declarative manner using, and guides the user through the composition process while allowing detection of violations at both design and run time. Abstraction aims to provide a capabilities driven specification of the composition process. The capabilities models and abstract plans defined for the abstract composition stage are then used to instantiate, verify and execute the composition process. The Instantiation step aims to find a concrete solution to the composition process respecting the constraints and assumptions specified in the model resulting from the design phase. Then, since the proposed declarative capabilities based composition process specification may only be partially defined and may contain conflicts or inconsistencies, the verification step enables to identify any conflicts or hard constraints. The execution step deals with the deployment of the composition onto a runtime infrastructure. Finally, the monitoring deals with the actual execution of the composite service and is responsible for monitoring the execution and recording violation of any requirement of the goal service at runtime. For this purpose, we have proposed an event-based monitoring framework that allows specifying and reasoning about the monitoring properties during composition process execution. Violations of monitoring properties are analyzed and used to initiate

different recovery actions such as ignore the violation, terminate the process, re-invoke or substitute the service, find an alternative solution based on current process state or backtrack to some previous state and then seek an alternative solution. Formalization of the proposed models and specifications using Notation3 and Proofs are discussed in chapter 4.

Chapter 4

Proof based Web Services Composition

Contents

Proof mechanisms explain how a certain goal can be reached by applying rules as inferences. A proof justifies a statement through its decomposition into more elementary pieces, which may only be combined using a strictly constrained methodology [147]. The resulting pieces can in turn be proven until we get fundamental pieces that we cannot decompose and have chosen to accept as truth [148]. Proofs can play a crucial role in the dynamic world of Web services composition. They can guarantee the correctness of a composition before its execution, and even serve as an efficient method to automatically create such compositions. Despite the importance of proofs, they are largely neglected by semantic Web researchers and did not attract a lot of attention as they deserve.

Very few works such as [154, 155] have addressed this issue and showed that proof-based consumption of hypermedia APIs is a feasible strategy at Web scale. Based on this conclusion, this chapter shows how Web services compositions can be achieved through the generation of a proof based on semantic descriptions of the services functionality and QoS. To pragmatically verify the applicability of these compositions, the notion of pre-execution and post-execution proofs were used. First, we introduce the main ingredients for a proof based method. We explain Notations3 rule language depicting its formal syntax and semantics. We introduce EYE reasoner and outline the motivations beyond its usage compared to other semantic reasoners. We end this part by explaining proof methods using N3 and EYE. Second, we study the generation of service compositions using proofs. We discuss the mapping of capabilities based specifications into N3 and explain the composition strategy. We use an illustrative example to highlight our ideas. Third, we explain the use of proofs in verification purposes. Finally, we conclude the chapter.

4.1 Proof Ingredients

Proof mechanisms enable to discover knowledge derived from the truth and to distinguish that from what is false. The notion of proof was already present in the initial Semantic Web vision. Indeed, as shown in figure 4.1, the Proof layer is one of the higher layers of the semantic Web stack. This layer is used to check and justify a statement in a given context [147]. Results obtained by machines can be trusted when accompanied by an independently verifiable, machine-readable proof document [148].

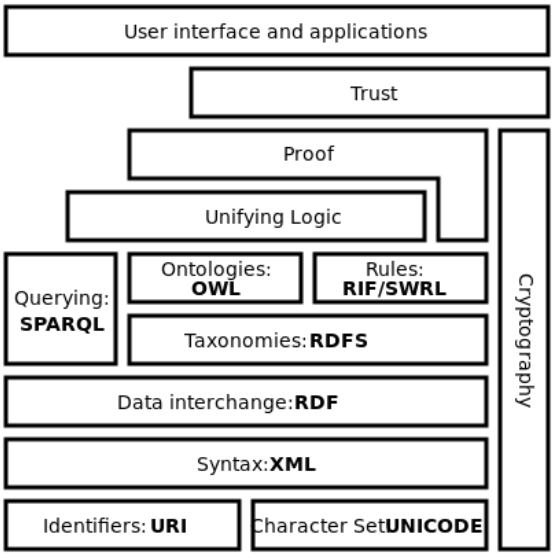


FIGURE 4.1: Semantic Web Stack

A proof is a conjunction of components, recursively constructed out of inferences and extractions. Regarding our objective to generate declarative compositions, proofs will be applied to show how a goal can be achieved instead of verifying the derivation of facts from static knowledge. In this section, we first introduce the main ingredients of a proof based method, which are N3 rule language and EYE reasoner. Next, we discuss the proof generation process.

4.1.1 Notation 3

Notation 3 (N3) is an assertion and logic language which is a superset of Resource Description Framework (RDF). N3 augments the RDF model with symbols for quantification, implication, and functional predicates, as well as providing a textual syntax alternative to RDF/XML. This language has been developed in the context of the Semantic Web Interest Group. The aims of the language are:

- to optimize expression of data and logic in the same language,
- to allow RDF to be expressed,
- to allow rules to be integrated smoothly with RDF,
- to allow quoting so that statements about statements can be made, and
- to be as readable, natural, and symmetrical as possible.

In our work, we chose N3 over other formalisms such as Prolog or Datalog for two main reasons: i) adding logic and rules to our capability-based service description, defined in RDF, to facilitate inferences for both discovery and composition processes; and ii) ensuring a declarative language for Web service composition process, compatible with the architectural principles of the Web. In what follows, we present the most relevant syntactic and semantic features of N3, based on the formalization introduced in [156, 157].

4.1.1.1 Formal Syntax

Definition 4.1. *(N3 Alphabet):* An N3 *alphabet* A consists of the following symbols and variables:

- U: a set of URIs.

- V: a set of quantified symbols, called variables, which can be universally or existentially quantified: V=$V_E \cup V_U$
  - V_E: a set of existential variables indicated by a $@forSome$ or "_:" notation. The existence of such type of variables in a formula F indicates that F is *true* if and only if there exists some value of the variable. Note that the blank node notation represents an unnamed existential variable.
  - V_U: a set of universal variables indicated by a $@forAll$ or "?" notation. The existence of such type of variable in a formula F indicates that F is true for any value of the variable.

  Note that specifying the two types of variables in the same context means that the scope of V_U is outside the scope of V_E. For example:

  $\{@forAll < \#r > .@forSome < \#s > . < \#s >< \#knows >< \#r >\}.$

  means:

  $\forall < \#r > (\exists < \#s > (< \#s >< \#knows >< \#r >)).$

- Expression E: each simple entity in A, such as URI, V, Literal, False and {}, is considered as an expression E.
- Statement S: a rule in the form $F \Rightarrow G$ where the subject F is the premise (antecedent) of the rule, and the object G is its consequent (conclusion).
- Brackets B: we distinguish three types of brackets, which are:
  - Square Bracket [ ]: usually used to represent blank node.
  - Angle Bracket <>: used to quote URIs.
  - Curly Bracket { } : used to express statements or formulas.
- Logical implication($\Rightarrow$): a property that relates two formulas to express implication. It is almost used to define a rule: {F} $\Rightarrow$ {G} means that G is true if and only if F is true.
- Literals L: String notation for representing a given value of a variable.

Definition 4.2. *(Formula F):* The set F of N3 formulas over alphabet A is defined as following:

1. Atomic: consists of a conjunction of simple expressions: $e_1 e_2 e_3$, where e_1, e_2 and $e_3 \in$ E.
2. Implication: consists of sub-formulas related by a logical implication: $f_1 \Rightarrow f_2$, where f_1 and $f_2 \in$ F.

3. Conjunction: obtained by composing formulae with logical connectives and quantifiers: $f_1 f_2$, where f_1 and $f_2 \in$ F.

Note that a formula/expression without any variables is considered as a ground formula/expression.

Definition 4.3. *(Substitution (subst)):* Let A be an N3 alphabet, f $\in F$ an N3 formula over A, V a set of variables and E a set of expressions.

A substitution is an operator that recursively takes place on variables but not on equals. It is in the form:

$\sigma_{\{v_1/e_1,v_2/e_2,v_3/e_3,...,v_n/e_n\}} = \sigma_{\{v_1/e_1\}}\sigma_{\{v_2/e_2\}}\sigma_{\{v_3/e_3\}}...\sigma_{\{v_n/e_n\}}$, where $v_i \in V$ and $e_i \in E$.

For example, a substitution operator $\sigma_{\{x/m\}}$ replaces all the occurrences of the variable x in f_1 with the expression m as following:

f_1*: {?x :says {?x :loves :Albert.}.}*$\sigma_{\{x/m\}}$= {*m :says {m :loves :Albert.}.}*.

4.1.1.2 Semantics of N3

Definition 4.4. *(Interpretation):* An interpretation I of an alphabet A consists of:

1. A non-empty domain Δ^I,
2. A mapping function δ^I that assigns each object to a subset of Δ^I and assigns each predicate to a subset of $\Delta^I * \Delta^I$.

Definition 4.5. *(Model):* A semantic model of N3 is considered as a pair $M = (I, W)$, where I is an interpretation and W is a set of states (seen as possible worlds). For each state $w \in W$, I associates an interpretation $I(w) = (\Delta^I, \theta_0^{I(w)}, .., \theta_n^{I(w)}, \varphi_0^{I(w)}, .., \varphi_n^{I(w)})$. Where the object $\theta_i^{I(w)} \subseteq \Delta^I$ and the predicate $\varphi_i^{I(w)} \subseteq \Delta^I * \Delta^I$.

Definition 4.6. *(Entailment):* Given an interpretation I = (Δ^I, δ^I); formula f, f_1, $f_2 \in F$ over an alphabet A and a state s∈S. f includes variables (one at least), while f1 and f2 are ground formulae. a, b and c ∈ E are ground expressions, P is a predicate, $\sigma_{x,n}$= subst(x, n) and $\sigma_{y,m}$= subst(y, m) where x∈ V_U, y∈ V_E and n, m ∈E. The entailment (satisfaction) of a set of formulas F in a model $M = (I, w)$, written as M⊨F, is defined by:

1. M⊨ f(a); iff $a^I \in f^{I(w)}$;
2. M⊨ P(b,c); iff $(b^I, c^I) \in P^{I(W)}$;

3. M$\models$ f; iff $\forall v_U$ of μ_{v_U}(f), $\exists$ μ_{v_E}(f) of v_E in f, so that: M$\models$ μ_{v_U} o μ_{v_E}(f).
4. M$\models$ $\{\sigma_{x,n} f\} \Rightarrow$ true; iff M$\models$ $\{f\} \Rightarrow$ true,
5. M$\models$ $\{f\} \Rightarrow$ true; iff $\exists$ at least one substitution $\sigma_{y,m}$ where M$\models$ $\{\sigma_{y,m} f\} \Rightarrow$ true;
6. M$\models$ $\{f_1\} \Rightarrow \{f_2\}$; iff M$\models$ f_2 if M$\models$ f_1;
7. M$\models$ $\{f_1\} \Rightarrow$ false; iff M $\not\models$ f_1;

4.1.2 Euler Yap Engine (EYE) Reasoner

Euler Yet another proof Engine (EYE) is an open source reasoning engine. It is a further incremental development of Euler which is an inference engine supporting logic based proofs. EYE is a semi-backward reasoning engine enhanced with Euler path detection. Like all N3 reasoners such as *cwm* [158], the EYE reasoner enables to generate and exchange proofs, which can be used for software synthesis or services composition [159]. Compared to other implicit reasoners such as Pellet [160] and Jena reasoners [161], EYE, whose features include backward-chaining and high performance, leverages the language's support of formulas and quantification for RDF to provide a logical framework for inferencing [162]. Also, EYE enables to define rules in a high level of abstraction in order to be re-used by different instances within different contexts. Other important aspect of this engine is that it supports semantic Web reasoning, enabling the dynamic features of the Web and enhancing decentralization and open-world assumptions. The non-availability of a resource does not mean that this resource does not exist anymore or its related triples are becoming false. In the current design of EYE things are layered and cascaded as shown in figure 4.2.

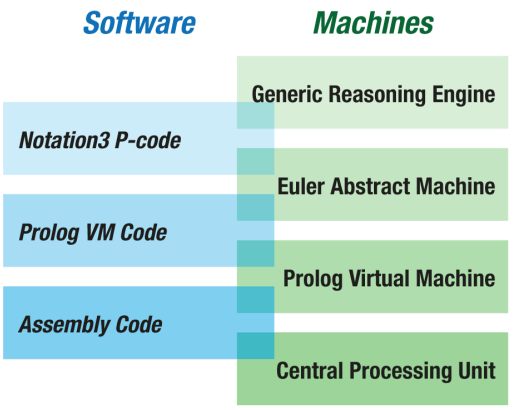


FIGURE 4.2: EYE Design [1]

4.1.3 Proof Study

In order to show how a proof can be generated using N3 and EYE, we consider a simple example of a Doodle service used to retrieve information (such as title, language,etc) about an existing Doodle poll. The identification code (id) of the existing Doodle poll can be considered as the Initial state (KB) and is formalized as follows:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<poll01> poll:hasId <wucm7t7z854877su>.
```

The rule can be expressed in N3 as follows:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix poll: <http://example.org/poll#>.

{?RequestedPoll poll:hasId ?id. }

=>

{
?RequestedPoll  poll:hasTitle _:title;
poll:hasLanguage _:language.
}.
```

By considering the knowledge base and rules in a separate resources, we can increase their reusability. These resources are then used by EYE reasoner to generate all possible statements that can be entailed by the current state. Otherwise, rules can be triggered recursively and lead to an infinite stream of triples. To avoid such enormous number of entailed triples and instruct the reasoner to extract just the required triple, a specific query should be defined. The query mechanism used by most of N3-reasoners is mainly based on filter rules that behave in a manner similar to the SPARQL CONSTRUCT queries. An example of a filter rule can be defined as follows:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix poll: <http://example.org/poll#>.

{?RequestedPoll ?p ?o}

=>

{?RequestedPoll ?p ?o.}.
```

Given the initial state, the service description (rule), and the user query as inputs, EYE reasoner executes the query and generates the following triples:

```
<poll01> poll:hasId <wucm7t7z854877su>.
<poll01> poll:hasTitle _:sk0.
<poll01> poll:hasLanguage _:sk1.
```

This conclusion was extracted from the proof illustrated in Listing 4.1, which explains how we get these triples.

```
@prefix poll: <http://example.org/poll#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix r: <http://www.w3.org/2000/10/swap/reason#>.

[ a r:Proof, r:Conjunction;
r:component <#lemma1>;
r:component <#lemma2>;
r:component <#lemma3>;
r:gives {
<poll01> poll:hasId <wucm7t7z854877su>.
<poll01> poll:hasTitle _:sk0.
<poll01> poll:hasLanguage _:sk1.
}].

<#lemma1> a r:Inference; r:gives {<poll01> poll:hasId <wucm7t7z854877su>};
r:evidence (<#lemma4>);
r:rule <#lemma5>.

<#lemma2> a r:Inference; r:gives {<poll01> poll:hasTitle _:sk0}; r:evidence (
<#lemma6>);
r:rule <#lemma5>.

<#lemma3> a r:Inference; r:gives {<poll01> poll:hasLanguage _:sk1}; r:evidence (
<#lemma6>);
r:rule <#lemma5>.

<#lemma4> a r:Extraction; r:gives {<poll01> poll:hasId <wucm7t7z854877su>};
r:because [ a r:Parsing; r:source <doodletranslator/ex/initialstate.n3>].

<#lemma5> a r:Extraction; r:gives {{?x1 ?x0 ?x2} => {?x1 ?x0 ?x2}};
r:because [ a r:Parsing; r:source <doodletranslator/ex/query.n3>].

<#lemma6> a r:Inference; r:gives {<poll01> poll:hasTitle _:sk0.
<poll01> poll:hasLanguage _:sk1}; r:evidence (
<#lemma4>);
r:rule <#lemma7>.

<#lemma7> a r:Extraction; r:gives {{?x0 poll:hasId ?x1} => {?x0 poll:hasTitle ?x2.
?x0 poll:hasLanguage ?x3}};
r:because [ a r:Parsing; r:source <doodletranslator/ex/poll_description.n3>].
```

Listing 4.1: Example of Proof

The proof starts from the conclusion, which is gradually decomposed into atomic components until heading toward the initial state. This is explained by the backward-chaining algorithm used by the EYE reasoner. The proof consists of a conjunction of three components, which are *#lemma*1, *#lemma*2 and *#lemma*3. We distinguish two types of lemma: Inference and Extraction.

For instance, *#lemma*1 is an inference instantiated from the rule *#lemma*5 with the knowledge detailed in *#lemma*4, and results in the fact that poll01 hasId "wucm...su".

However, the filter rule defined in *#lemma*5 is derived from the query source file, while the evidence predicate *#lemma*4 details the origin of the knowledge (initial state). Due to their direct point to the source files, *#lemma*4 and *#lemma*5 do not need further proving. For *#lemma*2 and *#lemma*3, they use *#lemma*5 as a filter rule, and another inference defined in *#lemma*6 that derives poll01 hasTitle _: sk0 and hasLanguage _: sk1. Note that there is no bindings in the proof (no universal variables), whereas, two existentially quantified variables are created with two blank nodes (_: sk0 and _: sk1). At this end, the detailed results are unknown but the description states that there would be details about the title and the language of the Poll01 .

The proof explained above is considered more or less simple, this is due to its use of pre-existing axioms to explain the used assertions. For a more complex reasoning, we propose to use the notion of proof to dynamically generate composite services, the facts of which are unknown beforehand.

4.2 Proof based Composition of Services Capabilities

Given a set of possible capabilities, our aim is to achieve a goal (requester goal) from an initial state. Furthermore, we might have some additional knowledge that can be incorporated. The above can be expressed in N3 as follows:

Definition 4.7. *Services composition problem.* Let F be the set of N3 formulas over an alphabet A which contains the predicates of a capability. A services composition problem consists of the following formulas:

- A set $H \subset F_g$ of ground formulas capturing all resource and application states the client is currently aware of, the ***initial state***.
- A formula $g \in F$, $n(g) \leq 1$, which does not contain existential variables, the ***goal*** state which indicates on a symbolic level what the client wants to achieve.
- A set R of capabilities descriptions or conjunctions of services descriptions, describing all services available to the requester, the ***description formulas***.
- A (possibly empty) set of N3 formulas B, the ***background knowledge***, where each $b \in B$ is either a ground formula or an implication $e1 \Rightarrow e2$: with $comp^n(e1 \Rightarrow e2) = \emptyset$ for all $n > 2$, which does not contain existential variables.

Below we discuss N3 capabilities descriptions and how they can serve in discovery and composition processes. Then, we study how proof can be generated using capability rules. We precise what kind of results it derives and how far the generated proof can meet the desired goal.

4.2.1 N3 capabilities descriptions

We recall that a service is considered as an uplet $S = \{Cap, NFP, Pa^*\}$, where:

- Cap: represents the capability or functionality of the service and consists of:
  - an action Verb (AV): concept used to define the action offered by the service;
  - set of attributes (DA): a finite set of domain specific attributes pairs (attribute name, attribute value);
  - Pre: a finite set of predicates that should be satisfied to execute the service;
  - Post: a finite set of facts that can be reached after executing the service and which result in a new world.
- NFP: a set of non-functional properties pairs $(propertyname, propertyvalue)$.
- Pa: the pattern used to represent the structural knowledge of a composite service and the connections between its different components.

We assume that A is an N3 alphabet, f, $g \in F$ a set of formulas over A, $s1$ is an atomic service with a capability *cap*, which has an action verb *av*. The N3 serialization of $s1$ can be represented as follows:

```
s1      a svc:Service;
svc:hasCapability cap.
cap a cap:Capability.
{preconditions} => {F(cap) postconditions.}.
```

Where:

- preconditions: set of formulas that describe the required resources to execute the service and access to its capability. Such formulas should not contain any existential variable.
- F(cap): is considered as a conjunction formula $\{F(cap) = g(av)h(at)\}$ that consists of:
  1. $g(av)$: ground formula that represents the action verb specific to a given capability. Such formula should contain only ground expressions. For instance:

     ```
     g(av): cap cap:hasActionVerb av.
     ```

2. $h(at)$: formula that includes (universal or existential or both) variables representing the attributes of a specific capability. Its N3 representation can be described as follows:

```
h(at): cap cap:attribute _:at1, _:at2, _:at3.
```

- postconditions: set of formulas used to describe the facts that can be derived after the execution of the service. In general, such formulas contain existential variables as well as all the universal variables contained in the preconditions.

4.2.2 Proof based Composition

We use EYE reasoner to generate capability-driven proofs. Inputs of EYE are domain ontology triples, capability rules, initial state and the goal. Two plans will be generated:

- Planning: in this stage, the composition problem consists of:
  - An initial State S includes a set of ground formulae that capture the current state.
  - A goal G: an N3 formula characterized by i) nesting level $n(G) \leq 1$; ii) does not include existential variables.
  - A set CR of abstract capabilities describing the available functionalities in the form of rules.
  - A background knowledge (KB) that consists of a set of formulas that can be either ground formulas or implication formulas (without existential variables).

  Given the above elements, the composer tries to generate a proof to check if $S \cup CR \cup BK$ can deliver G', where G' $\subseteq$ G. If verified, an abstract plan consisting in a set of capabilities action verbs and attributes names will be generated.

- The above generated plan with the set of services instances are then used to generate a concrete plan to be sent to an executor. This executor receives the instantiated plan and tries to execute it instance by instance. Each executed instance will be evaluated and then added to the knowledge base, where the reasoner checks if the goal is reached or not. If the goal is reached, the engine returns an answer back to the user, else, a new plan starting from the current state (after augmenting the knowledge base with the last instance effects) will be generated and the loop continues.

Algorithm 4: invokeReasoner Algorithm

Input: Initial State S, Goal specification G, set of capability type rules $CR_i \in CR$, KB, filter rule fr, CapInstances.
Output: Composition Plan meeting user goal **or** nil as failure.
begin
  P1 ← {} ;
  $n_{Cap} \leftarrow 0$;
  $n_{post} \leftarrow 0$;
  if *($\exists S_{gi} \subseteq S_G$) where the formula $S \models S_{gi}$ is True* **then**
    Generate proof P;
    P1 ← P.Filter (Cap, fr);
    n_{Cap} ← P1.nb(Cap);
    capCheck (n_{Cap}) ;
  else
    return Fail.
  Endif
  End

Algorithm 4 describes the abstract plan generation. Given the composition problem (S, G, CR, KB), the reasoner starts to generate the proof by checking if the initial state satisfies one-to-many parts of the goal.

Algorithm 5: Filter Proof Algorithm

Input: Proof P, Filter Rule fr.
Output: Filtered Proof FP.
begin
  FP← {};
  while $P \neq$ {} **do**
    FP← invokeReasoner (P, fr);
  End

If no entailment is satisfied, the reasoner will not be able to generate a proof, and therefore no solution exists for the current composition problem. Consequently, the process halts with failure. Else, the composition will be invoked and a proof (P) will be delivered and scanned for applications of abstract capabilities rules (CR). The number of these applications is set to n_{pre}. If $n_{pre} = 0$, the process will be halted with success and the required composition will be derived. Else, a new proof (FP) will be generated from the proof (P) using a filter rule fr and containing the abstract capabilities (CR_i=action verbs + attributes names) of the plan. Each capability will be received by the executable composition generator to be concretized with the best instance among the existing capability instances w.r.t NFP requirements. The selected instance will be then executed and the attribute values will be parsed to ground formulas F_G by the executor. This latter will re-invoke the reasoner with the new composition problem (S$\cup F_G$, G, $AC \setminus \{AC_i\}$,

KB) and the process will be iterated until $n_{pre} = 0$. Note that in case no instance for a given CR_i was found, the reasoner will re-generate a new proof starting from the current state and the process will be iterated until reaching the goal.

Algorithm 6: capCheck Algorithm

Input: Initial State S, Goal specification G, set of capability type rules $CR_i \in CR$, KB, filter rule fr, NFP.
Output: Composition Plan meeting user goal **or** nil as failure.

```
begin
    if (n_Cap=0) then
        Return S ;
    else
        for (each CR_i ∈ P1) do
            instance_i ← CR_i.getInstance(CapInstances, NFP);
            instance_i.Execute();
            Eff ← instance_i.getEffect() ;
            S ← S ∪ Eff;
            invokeReasoner (S, G, CR, KB) ;
            if (∃S_gi ⊆ S_G) where the formula S ⊨ S_gi is True then
                Generate proof P';
                P'1 ← P'.Filter (Cap, fr);
                n_post ← P'1.nb(Cap);
            else
                n_post ← n_Cap ;
            if (n_post ≥ n_Cap) then
                CR ← CR\{CR_i} ;
                invokeReasoner (S, G, CR, KB);
            else
                n_Cap ← n_post;
                capCheck (n_Cap) ;
            Endif
        Endfor
    Endif
End
```

4.2.3 Composition Scenario: Programmable Dinner Scenario

To better understand the concept of proof-based composition and reasoning over dynamic capability rules, we return back to the Programmable Dinner scenario defined in section 3.1.4, which is a composition of a set of services rules serialized in N3 as presented below.

- ***Get Weather Forecast service***:

```
@prefix pd: <http://example.org/programmableDinner#>.
@prefix ex: <http://example.org/action#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@prefix svc: <http://localhost/neologism/svc#>.
@prefix cap: <http://localhost/neologism/cap#> .
@prefix av: <http://localhost/neologism/av#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix tmp: <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix om: <http://www.wurvoc.org/vocabularies/om-1.6/> .
?WeatherForcast a svc:Service;
svc:hasCapability ?WeatherForcastCapability;
svc:hasNFP ?wfnfp.
?WeatherForcastCapability a cap:Capability.
?event a pd:Event.
?loc a geo:SpatialThing.
?date a geo:TemporalThing.
{
?event geo:location ?loc;
tmp:atDateTime ?date.
}
=>
{
?WeatherForcast svc:hasCapability ?WeatherForcastCapability.
?WeatherForcastCapability cap:hasActionVerb pd:GetForecast;
cap:attribute _:temp.
_:temp  a pd:Temperature;
pd:hasValue _:tval.
?event  geo:location ?loc;
tmp:atDateTime ?date;
pd:hasTemperature _:temp.
}.
```

- ***Find Restaurant Service***:

```
?GetRestaurant a svc:Service;
svc:hasCapability ?GetRestaurantCapability;
svc:hasNFP ?grnfp.
?GetRestaurantCapability a cap:Capability.
?event a pd:Event.
{?event geo:location ?loc. }
=>{
?GetRestaurant svc:hasCapability ?GetRestaurantCapability.
?GetRestaurantCapability cap:hasActionVerb pd:GetRestaurantName;
                                                cap:attribute _:rn.
                                                _:rn a pd:Restaurant;
                                               geo:location ?loc.
?event geo:location ?loc; pd:hasRestaurant _:rn.}.
```

- ***Create Map-with-Markers Service***:

```
?CreateMapWithMarkers a svc:Service;
svc:hasCapability ?CreateMapWithMarkersCapability;
svc:hasNFP ?cmnfp.

?CreateMapWithMarkersCapability a cap:Capability.
{?event geo:location ?loc;pd:hasRestaurant ?rn.}
=>
{?CreateMapWithMarkers svc:hasCapability ?CreateMapWithMarkersCapability.
?CreateMapWithMarkersCapability cap:hasActionVerb pd:CreateMapWithMarkers;
cap:attribute _:map.
_:map   a pd:Map; pd:markersFor ?rn.
?event pd:hasMap _:map.         }
```

- ***Get similar-movies Service***:

```
?SuggestMovie a svc:Service;
svc:hasCapability ?SuggestMovieCapability;
svc:hasNFP ?smnfp.
?SuggestMovieCapability a cap:Capability.
{?event a pd:Event.
?mtitle a pd:MovieTitle.}
=>
{?SuggestMovie svc:hasCapability ?SuggestMovieCapability.
?SuggestMovieCapability cap:hasActionVerb pd:SuggestMovie;
cap:attribute _:moviname.
_:moviname      a pd:Movie;
pd:hasTitle _:t.
_:t a pd:MovieTitle;
pd:contains ?mtitle.
?event pd:hasMovie _:moviname.
}.
```

- ***Identify Theaters Service***:

```
?SuggestTheatre a svc:Service;
                                svc:hasCapability ?SuggestTheatreCapability;
                                svc:hasNFP ?gtnfp.
?SuggestTheatreCapability a cap:Capability.
{
?event geo:location ?loc;
           tmp:atDateTime ?date;
           pd:hasMovie ?moviname.}
=>
{?SuggestTheatre svc:hasCapability ?SuggestTheatreCapability.
?SuggestTheatreCapability cap:hasActionVerb pd:SuggestTheatre;
                                          cap:attribute _:th.
_:th a pd:Theatre; geo:location _:h.
?event pd:hasTheatre _:th.
}.
```

- ***Send Invitation service***:

```
?SendInvitation a svc:Service;
                                svc:hasCapability ?SendInvitationCapability;
                                svc:hasNFP ?sinfp.
?SendInvitationCapability a cap:Capability.
{
?fm a pd:Mail.
?event a pd:Event.
?event geo:location ?loc;
                tmp:atDateTime ?date;
                pd:hasTemperature ?temp;
                pd:hasRestaurant ?rn;
                pd:hasMap ?map;
                pd:hasMovie ?moviname;
                pd:hasTheatre ?th.
}
=>
{
?SendInvitation svc:hasCapability ?SendInvitationCapability.
?SendInvitationCapability cap:hasActionVerb pd:SendInvitation;
                                                    cap:attribute _:msend.
_:msend a pd:InvitationState;
                pd:hasValue _:boolean.
?event geo:location ?loc;
                tmp:atDateTime ?date;
                pd:hasTemperature ?temp;
                pd:hasRestaurant ?rn;
                pd:hasMap ?map;
                pd:hasMovie ?moviname;
                pd:hasTheatre ?th.
?event pd:hasInvitationState _:msend.
}.
```

Assuming that the initial state consists of a given location, a date, a mail list and a movie title:

```
loc:Gent a geo:SpatialThing.
date:27012015 a geo:TemporalThing.
mail:sanabaccar_at_ceslab.org a pd:Mail.
mail:mohsen_at_gmail.com a pd:Mail.
mail:ons12_at_gmail.com a pd:Mail.
pd:Titanic a pd:MovieTitle.
pd:ProgramDinner a pd:Event;
geo:location loc:Gent;
tmp:atDateTime date:27012015.
```

The goal set by the user is to send an invitation to a list of friends with a list of restaurants in a given location marked in maps. The invitation includes all the details of the event such as date, location, movie title and theater name. The invitation should return the state "sent":

```
{
?event geo:location ?loc;
tmp:atDateTime ?date;
pd:hasTemperature ?temp;
pd:hasRestaurant ?rn;
pd:hasMap ?map;
pd:hasMovie ?moviname;
pd:hasTheatre ?th.
?event pd:hasInvitationState ?msend.
}
=>
{
?event geo:location ?loc;
tmp:atDateTime ?date;
pd:hasTemperature ?temp;
pd:hasRestaurant ?rn;
pd:hasMap ?map;
pd:hasMovie ?moviname;
pd:hasTheatre ?th.
?event pd:hasInvitationState ?msend.
}.
```

Listing 4.2 represents an example of an N3 serialization of two concrete instances for the abstract capability *Get Weather Forecast*.

```
<http://www.accuweather.com/> a svc:Service ;
svc:hasCapability<//localhost/neologism/cap#Capability/GetForecast_b1ae8a0f>;
svc:hasNFP <http://localhost/neologism/NFP#ServiceNFP/6679e6d8> .
<http://localhost/neologism/cap#Capability/GetForecast_b1ae8a0f>
a cap:Capability ;
cap:attribute      <http://example.org/programmableDinner#forecast/204c0b47> ;
cap:hasActionVerb  pd:GetForecast .
<http://localhost/neologism/NFP#ServiceNFP/6679e6d8>  a              nfp:Nfp ;
cap:attribute <http://example.org/programmableDinner#ServiceCost>
, <http://example.org/programmableDinner#ResponseTime>
, <http://example.org/programmableDinner#Availability> .
<http://example.org/programmableDinner#ServiceCost> a nfp:Cost ;
nfp:hasAttributeValue "10" .
<http://example.org/programmableDinner#ResponseTime> a nfp:ResponseTime ;
nfp:hasAttributeValue "20" .
<http://example.org/programmableDinner#Availability> a nfp:Availability ;
nfp:hasAttributeValue "0.5" .

<http://www.weather.com/> a svc:Service ;
svc:hasCapability<//localhost/neologism/cap#Capability/GetForecast_a9a1db92>;
svc:hasNFP <http://localhost/neologism/NFP#ServiceNFP/87308e37> .
<http://localhost/neologism/cap#Capability/GetForecast_a9a1db92>
a cap:Capability ;
cap:attribute      <http://example.org/programmableDinner#forecast/204c0b47> ;
cap:hasActionVerb  pd:GetForecast .
<http://localhost/neologism/NFP#ServiceNFP/87308e37> a nfp:Nfp ;
cap:attribute  <http://example.org/programmableDinner#ServiceCost>
, <http://example.org/programmableDinner#ResponseTime>
, <http://example.org/programmableDinner#Availability> .
<http://example.org/programmableDinner#ServiceCost> a nfp:Cost ;
nfp:hasAttributeValue "7" .
<http://example.org/programmableDinner#ResponseTime> a nfp:ResponseTime ;
nfp:hasAttributeValue "23" .
<http://example.org/programmableDinner#Availability> a nfp:Availability ;
nfp:hasAttributeValue "0.8" .
```

Listing 4.2: Alternative instances for GetWeatherForecast Capability serialized in N3

The reasoner takes the defined descriptions, the initial knowledge and the goal as inputs and generates a proof. Note that the existing of proof means that the desired goal is *theoretically* reachable over the current knowledge and the existing *abstract* capabilities.

Going through the generated proof, we can observe that the proof starts by describing its main entity, which is based on a conjunction of one-to-many components. Then, it explains step by step how the reasoner arrives at the conclusion. As shown in the listing below, the generated proof starts from the result and describes one component $\#lemma1$.

```
[ a r:Proof, r:Conjunction;
r:component <#lemma1>;
r:gives {
pd:ProgramDinner geo:location loc:Gent.
pd:ProgramDinner tmp:atDateTime <http://example.org/date#27012015>.
pd:ProgramDinner pd:hasTemperature _:sk2.
pd:ProgramDinner pd:hasRestaurant _:sk6.
pd:ProgramDinner pd:hasMap _:sk9.
pd:ProgramDinner pd:hasMovie _:sk16.
pd:ProgramDinner pd:hasTheatre _:sk20.
pd:ProgramDinner pd:hasInvitationState _:sk24.
}].
```

As we have mentioned so far, lemmas can be classified into two main types: Inference and Extraction. The Inference lemmas instantiate the related filter rule with the existing knowledge, whereas the Extraction lemmas describe the origin of the knowledge without the need to prove it.

As illustrated in the listing below, *#lemma*1 is an inference to give the invitation state as well as the details of the programmable dinner such as the date, the temperature at this date, the restaurant where the dinner program will take place, the movie to be watched, etc. *#lemma*1 uses the rule described in *#lemma*10 and instantiates the triples that exist in *#lemma*2- *#lemma*9 to arrive to this conclusion.

```
<#lemma1> a r:Inference; r:gives {pd:ProgramDinner geo:location loc:Gent.
pd:ProgramDinner tmp:atDateTime <http://example.org/date#27012015>.
pd:ProgramDinner pd:hasTemperature _:sk2.
pd:ProgramDinner pd:hasRestaurant _:sk6.
pd:ProgramDinner pd:hasMap _:sk9.
pd:ProgramDinner pd:hasMovie _:sk16.
pd:ProgramDinner pd:hasTheatre _:sk20.
pd:ProgramDinner pd:hasInvitationState _:sk24}; r:evidence (
<#lemma2>
<#lemma3><#lemma4>
<#lemma5><#lemma6>
<#lemma7> <#lemma8><#lemma9>);
r:rule <#lemma10>.
```

As shown in listing 4.3, the Extraction lemmas such as *#lemma*2 and *#lemma*3 are directly point to the source file that must be parsed to get the triple or rule. These lemmas with a filter rule (*#lemma*11) will be then used by an Inference lemma (*#lemma*4) to instantiate the capability *GetForecast* and its related attributes. The same process will be recursively iterated so that each derived triple should be justified by an Inference lemma, which should be justified in its turn via Extraction lemmas until arriving at the axioms existing in the input files and instantiating all the capabilities required to reach the given goal.

```
<#lemma2> a r:Extraction; r:gives {pd:ProgramDinner geo:location loc:Gent};
r:because [ a r:Parsing; r:source <file:///c:/pd/is.n3>].
<#lemma3> a r:Extraction; r:gives {pd:ProgramDinner tmp:atDateTime;
r:because [ a r:Parsing; r:source <file:///c:/pd/is.n3>].
<#lemma4> a r:Inference; r:gives {_:sk0 svc:hasCapability _:sk1.
_:sk1 cap:hasActionVerb pd:GetForecast.
_:sk1 cap:attribute _:sk2.
_:sk2 a pd:Temperature.
_:sk2 pd:hasValue _:sk3.
pd:ProgramDinner pd:hasTemperature _:sk2};r:evidence (<#lemma2><#lemma3>);
<#lemma9> a r:Inference; r:gives {_:sk22 svc:hasCapability _:sk23.
_:sk23 cap:hasActionVerb pd:SendInvitation.
_:sk23 cap:attribute _:sk24.
_:sk24 a pd:InvitationState.
_:sk24 pd:hasValue _:sk25.
pd:ProgramDinner pd:hasInvitationState _:sk24}; r:evidence (
<#lemma18> <#lemma14> <#lemma2> <#lemma3> <#lemma4> <#lemma5>
<#lemma6> <#lemma7> <#lemma8>);
r:rule <#lemma19>.
<#lemma11> a r:Extraction; r:gives {{?x0 geo:location ?x1.
?x0 tmp:atDateTime ?x2} => {_:x3 svc:hasCapability _:x4.
_:x4 cap:hasActionVerb pd:GetForecast.
_:x4 cap:attribute _:x5.
_:x5 a pd:Temperature.
_:x5 pd:hasValue _:x6.
?x0 geo:location ?x1.
?x0 tmp:atDateTime ?x2.
?x0 pd:hasTemperature _:x5}};
r:because [ a r:Parsing; r:source <file:///c:/pd/rules.n3>].
<#lemma19> a r:Extraction; r:gives {{?x0 a pd:Mail.
?x1 a pd:Event.
?x1 geo:location ?x2.
?x1 tmp:atDateTime ?x3.
?x1 pd:hasTemperature ?x4.
?x1 pd:hasRestaurant ?x5.
?x1 pd:hasMap ?x6.
?x1 pd:hasMovie ?x7.
?x1 pd:hasTheatre ?x8} => {_:x9 svc:hasCapability _:x10.
_:x10 cap:hasActionVerb pd:SendInvitation.
_:x10 cap:attribute _:x11.
_:x11 a pd:InvitationState.
_:x11 pd:hasValue _:x12.
?x1 geo:location ?x2.
?x1 tmp:atDateTime ?x3.
?x1 pd:hasTemperature ?x4.
?x1 pd:hasRestaurant ?x5.
?x1 pd:hasMap ?x6.
?x1 pd:hasMovie ?x7.
?x1 pd:hasTheatre ?x8.
?x1 pd:hasInvitationState _:x11}};
r:because [ a r:Parsing; r:source <file:///c:/pd/rules.n3>].
```

Listing 4.3: Proof-based composition for Programmable Dinner Scenario

The proof can then be used to generate the abstract plan by simply defining filter rules that extract the candidate services (identified by their action-verbs) and the possible dependencies between them. $P1$ and $P2$ defined below represent two generated plans.

1. *P1: [GetForecast, GetRestaurantName, CreateMapWithMarkers, SuggestMovie, SuggestTheatre, SendInvitation]*
2. *P2: [GetForecast, SuggestMovie, SuggestTheatre, GetRestaurantName, CreateMapWithMarkers, SendInvitation]*

Candidate capabilities should include the following action verbs: i) GetForecast, ii) GetRestaurantName, iii) CreateMapWithMarkers, iv) SuggestMovie, v) SuggestTheatre, vi) SendInvitation. For the capabilities attributes, they are temporary unknown (but they exist). Using the same instantiation and substitution operations, the blank nodes of each capability will be instantiated one by one.

4.3 Correctness of Web services composition proofs

Once the composition process was generated using proofs, the correctness of these proofs has to be verified to garantee the consistency of the generated plan. In this context, [148] introduced two different kinds of proofs, *pre-proof* and *post-proof*, for a composition problem based on proofs. The idea beyond this extension to the notion of proofs in classical Semantic Web vision [163] is to support dynamic data. We propose to use these types of proofs to handle dynamic data in Web services environments.

Given the initial state S, the background knowledge KB, the set of abstract capabilities CR, and the goal G, a *pre-proof* assumes that the execution of all abstract capabilities will behave as expected. This means $S \cup CR \cup KB \models G1$, where $G1 \subseteq G$.

A *post-proof* considers an additional evidence about the goal. This evidence is provided by the current execution results, which gives: $S \cup AC \cup KB \cup \{executionresults\} \models G2$, where $G1$ and $G2$ are instances of G.

The verification process is important because we can never guarantee that a composition that has proven to work in theory will always and reliably achieve the desired result in practice, since the individual steps can fail. Some faults cannot be predicted and may cause a composition not to reach a goal that would normally be possible. In our case, the pre-proof assumes that correct behaviors of all involved services will lead to a composition satisfying the desired goal. The pre-proof can be validated at design time.

However, the post-proof can be validated only after executing the composition process and getting results.

However, checking the correctness of proofs requires to formalize them in a machine readable way. The N3 proof vocabulary created in the context of the Semantic Web Application Platform (SWAP) [158] provides this formalization. A proof is considered as a conjunction of N3 formulas describing inference steps a reasoner has performed to come to a certain conclusion. These inference steps, also called proof steps, are *Axiom*, *Conjunction elimination*, *Conjunction introduction*, and *Generalized modus ponens*. Given a set of formulas F over an N3 alphabet A, $\Gamma \subseteq F$ a set of formulas and $f, f1, f2, g \in F$, then the proof steps are characterized as deduction rules as indicated below.

1. Axiom: If $f \in \Gamma$ then $\Gamma \vdash f$
2. Conjunction elimination: If $\Gamma \vdash f1f2$ then $\Gamma \vdash f1$ and $\Gamma \vdash f2$.
3. Conjunction introduction: Let $\Gamma \vdash f1$ and $\Gamma \vdash f2$ and let $\rho_1, ..., \rho_m$ be existential renamings such that $f'2 = \rho_1 \circ ...., \circ\rho_m(f2)$ with $comp(f1) \cap comp(f'2) \cap V_E = \emptyset$ then $\Gamma \vdash f1f'2$.
4. Generalized modus ponens: If $\Gamma \vdash \{f1\} \Rightarrow \{f2\}$ and $\Gamma \vdash g$ and there exist universal replacements $\mu_1, ..., \mu_n$ such that $\mu_1 \circ ...., \circ\mu_n(\{f1\} \Rightarrow \{f2\}) = \{f'1\} \Rightarrow \{f'2\}$ and $f'1 = g$ then $\Gamma \vdash f'2$.

Definition 4.8. *Correctness of proof calculus.* Let Γ be a set of N3 formulas and f a formula over the same N3 alphabet A. Then, If $\Gamma \vdash f$ then $\Gamma \models f$.

Checking the correctness of a given proof consists in proving that every proof step is correct. [164] discussed the correctness of proof steps and provided a detail demonstration for each step. Applying this to the Web services composition problem, we consider the correctness of Web services composition proofs as indicated in the definition below.

Definition 4.9. *Correctness of Web services composition proofs.* Let (S, CR, KB, G) be a composition problem and $G1$ and instance of G then If $S \cup CR \cup KB \vdash G1$ then $S \cup CR \cup KB \models G1$

To handle the correctness proof of the composition problem, the generalized modus ponens step proof can be considered as the most important step because in this step implication rules such capabilities descriptions are applied. In this context, let A be an N3 alphabet, $f \in F(G)$ a ground formula and $\{f1\} \Rightarrow \{f2\} \in F(G)$ an implication of nesting level 2 where all universal variables which occur in $f2$ also occur in $f1$. If

the generalized modus ponens is applicable to f and $\{f1\} \Rightarrow \{f2\}$ then the resulting formula does not contain universal variables. This was proved in [164].

Let A be an N3 alphabet and $I = (D, \delta, \omega)$ be an interpretation of its formulas. Let $q, p, p1, ..., pn \in U$ be N3 representations of proof steps and $s1, s2, s3 \in U$. Then, the N3 proof vocabulary looks as follows:

1. Proof step types:
   - $I \models q$ a *r:Proof.* iff q is the proof step which leads to the proven result.
   - $I \models q$ a *r:Parsing.* iff q is a parsed axiom.
   - $I \models q$ a *r:Conjunction.* iff q is a conjunction introduction.
   - $I \models q$ a*r:Inference.* iff q is a generalized modus ponens.
   - $I \models q$ a *r:Extraction.* iff q is a conjunction elimination.
2. Proof predicates:
   - $I \models q$ *r:gives* $\{f\}$. iff $f \in F$ is the formula obtained by applying q.
   - $I \models q$ *r:source u.* iff q is a parsed axiom and $u \in U$ is the URI of the parsed axiom's source.
   - $I \models q$ *r:component p.* iff q is a conjunction introduction and p is a proof step which gives one of its components.
   - $I \models q$ *r:rule p.* iff q is a generalized modus ponens and p is the proof step which leads to the applied implication.
   - $I \models q$ *r:evidence* $(p1, ..., pn)$. iff q is a generalized modus ponens and $p1, ..., pn$ are the proof steps which lead to the formulas used for the unification with the antecedent of the implication.
   - $I \models q$ *r:because p.* iff q is a conjunction elimination and p is the proof step which yields the to-be-eliminated conjunction.
3. Substitutions
   - $I \models q$ *r:binding s1.* iff q includes a substitution $s1$.
   - $I \models s1$ *r:variable s2.* iff $s1$ is a substitution whose domain is $\{s2\}$.
   - $I \models s1$ *r:boundTo s3.* iff $s1$ is a substitution whose range is $\{s3\}$.

The reasoner (in our case EYE) requires to be given all formulas about $S \cup CR \cup KB$ and the desired goal G to be proven in order to produce a proof for a composition problem. The goal is provided as as the consequence of a filter rule $\{f\} \Rightarrow \{G\}$. Upon receiving

this sequence, the reasoner will proceed to prove an instance of f and returns each provable ground instance of G if possible, or a provable instance containing existentials otherwise.

4.4 Conclusion

In this chapter, we have explained a novel solution to automated composition and execution of Web services. A crucial part in generating a composition is the ability to determine whether it will satisfy a given goal without any undesired effects. This has led us to the approach of a pragmatic proof, wherein services capabilities are incorporated as inference rules. One major advantage of using proof-based composition is that it does not require new algorithms and tools, but can be applied with existing Semantic Web reasoners. Those reasoners can easily incorporate external sources of knowledge such as ontologies or business rules. Furthermore, the performance of composition generation improves with the evolution of those reasoners. Also, the fact that a third-party tool is used allows independent validation of the composition.

We have introduced the main ingredients for a proof based method. More precisely, we have motivated the use of N3 rule language and discussed its formalization depicting syntax and semantic features. Then, we have presented EYE reasoner and outlined the motivations beyond its usage compared to other semantic reasoners. We ended this part by explaining proof methods using N3 and EYE. In a second part, we have studied the generation of service compositions by discussing the mapping of capabilities based specifications into N3 and explaining the composition strategy relying on a Programmable Dinner composition scenario. Finally, we have explained the use of proofs in verification purposes. Next chapter will show how the proposed contribution was implemented and study the performance of the proof based composition strategy.

Chapter 5

Implementation and Performance Study

Contents

In this chapter, we discuss the implementation details of the prototype that realizes the major features of our approach to support the full round-trip composition life-cycle in a unified and declarative way. The purpose of the implementation prototype is firstly to demonstrate the feasibility of the proposed solution, and secondly to use it as a test bed to evaluate the declarative integrated approach. The second part of this chapter is dedicated to the performance study of this approach. We start by evaluating its different components and comparing them with some existing composition frameworks. We consider various criteria such as services dependencies, number of parameters (preconditions and effects), number of instances, time consumption and scalability.

5.1 Implementation

We have studied the feasibility of our proposed composition framework described in chapter 3 by implementing a Java-based prototype, which is generic enough to be used by different composition scenarios. The composition process starts when the user specifies a composition query using a user-friendly interface, allowing to define the required functionalities and the desired non-functional attributes (such as QoS parameters), as well as their related constraints. In a second step, the application converts the composition design into the corresponding capabilities models and the process will be launched. As illustrated in figure 5.1, three main phases will take place:

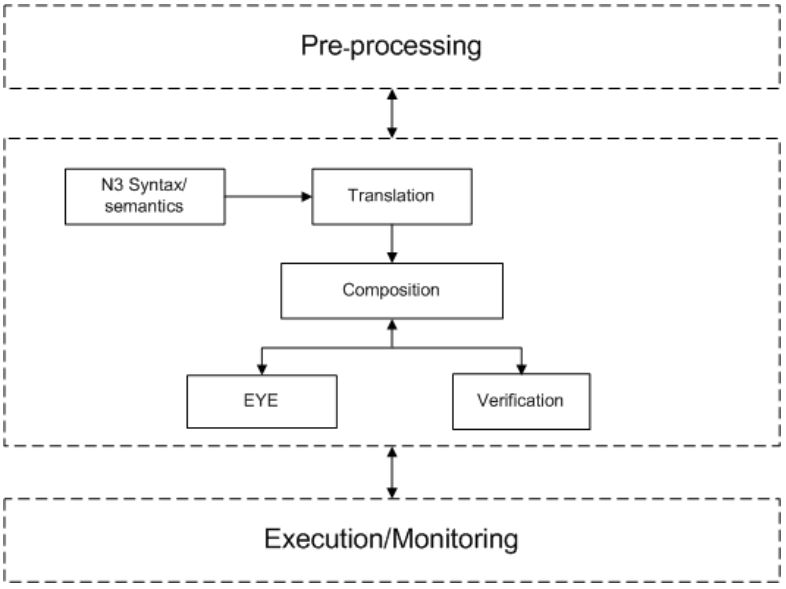


FIGURE 5.1: Implementation Architecture

- *The pre-processing phase:* enables to parse the query, extract the functional (FR) and non-functional (NFR) requirements and define the knowledge base such as constraints, capability ontology, instance ontology and type ontology.
- *The translation phase:* focuses on converting automatically the composition design into capabilities models. These capabilities models are then transformed into N3 specification. Specified constraints are also translated into N3 formulas.
- *The reasoning phase:* uses the EYE reasoner during the planning phase, to either provide possible composition solutions or detect conflicts. In case of conflicts the verification phase attempts to identify the cause of conflict and may resolve it by asking to update the composition design (or the hard constraints) or re-invoking the reasoner. In case of multiple solutions, the user may give options or preferences

to rank or select one particular solution, which is then used by the Java application to perform the actual services execution.

Figure 5.2 illustrates a snapshot of the query editor console of the prototype. The query editor consists of two parts: first part (left panel) enables the users to define the functional and non-functional user requirements. The required functionalities are represented as a set of capabilities, each of which is described by a an action names (action verbs) and a set of its related attributes (names and values). The lower left panel allows the users to express their non-functional requirements, such as QoS, by defining the attributes and their related constraints. A query viewer is included to check the parsed query after defining it. The second part of the console (right panel) allows the users to view the generated plans and their related concrete compositions. Taking the set of the defined capabilities as inputs, a simple click on *Start Generating Plans* button will invoke the reasoner and none-to-many plans may be generated and scanned in the console. The non functional requirements will be then taken as inputs with the generated plan to provide an optimal executable composition that consists of a set of Web services calls (URLs).

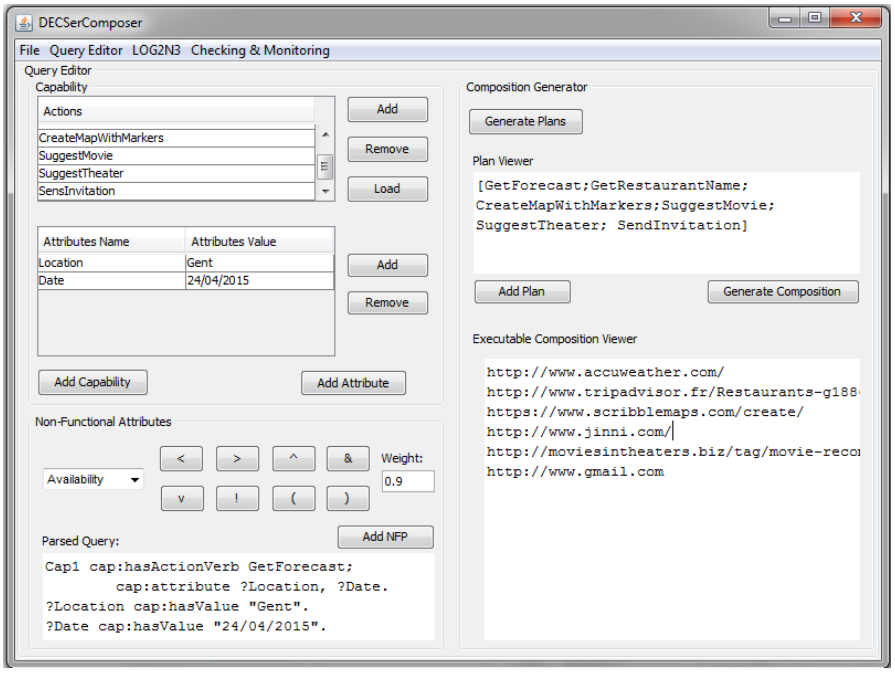


FIGURE 5.2: Snapshot of the Query Editor Console

For monitoring purposes, this engine uses log4j [1] to generate logs of the events during the execution of the composition process. This event log is fed to our framework in order to provide the runtime information that is required for monitoring. The output of log4j

[1] http://logging.apache.org/log4j/2.x/

is analyzed by the event receiver of the prototype in order to extract the events which are taken into account during the monitoring process.

Figure 5.3 shows a snapshot of the monitoring console of the prototype. The upper left panel of the monitoring console shows the formulas that express the constraints to monitor. Using the console, the user of our framework can select one or more of the formulas to monitor. Once selected, a formula appears in the lower left panel of the console. When monitoring is activated, the cases which violate and satisfy the selected formulas are shown in the monitoring console (see upper right panel of the console). The user can select any of these cases in order to see the exact instantiation of the formula (template) that underpins the case. This instantiation includes the events that have been unified with the different predicates in the formula, the source and timestamp of each of these events and the truth values of the concepts that the events have been unified with.

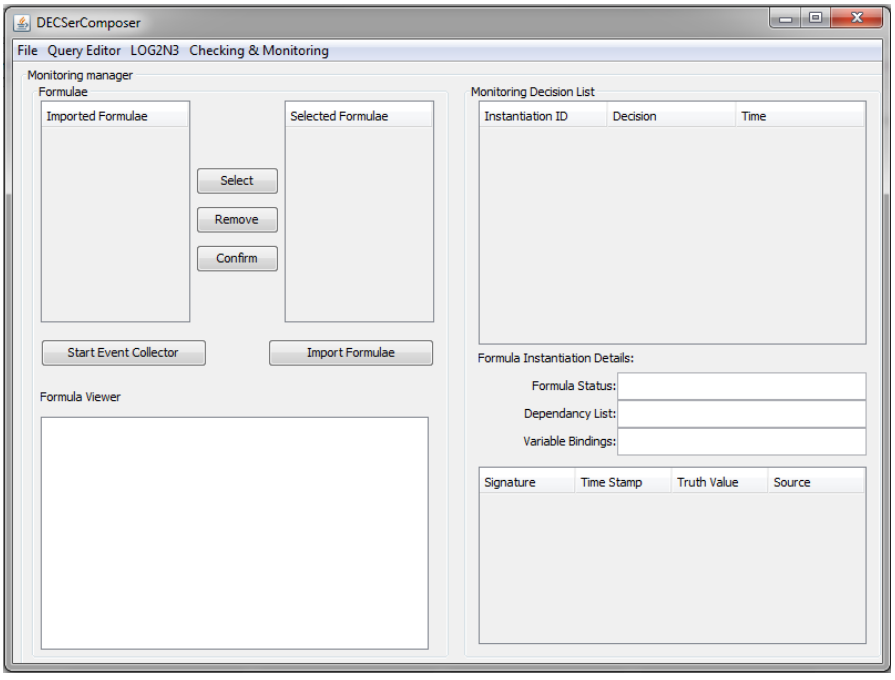


FIGURE 5.3: Snapshot of the monitoring console

The monitoring engine is based on EYE Reasoner to (i) process the events that are recorded in the event log by an event extractor based on the order of their occurrence, (ii) identify other expected events that should have happened but have not been recorded (these events can be derived from the composition requirements by deduction), and (iii) check if these events are compliant with the constraints of the composition process. The satisfiability of a requirement can be checked against the recorded behavior of the composition process. More specifically, checking whether the set of the recorded events

that have been generated by the execution of the composition process entail the negation of a requirement. When requirements variations are detected, a deviation notification is sent to the composition manager. This notification indicates: (i) the requirement that has been violated, (ii) the malfunctioning service(s) that violated it, and (iii) diagnostic information regarding the violation. Based on the type of the deviation notification and the service monitoring policies, recovery techniques will be triggered. A deviation viewer is also incorporated with the monitoring console.

5.2 Performance Study

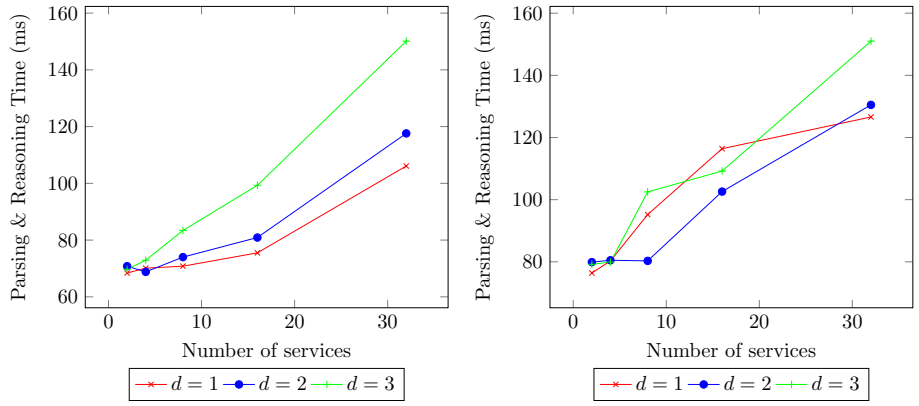
Due to the lack of testing dataset for capability-based web services (described in N3), and most of existing web service is based on OWL-S and WSDL, we can not test our approach directly. Therefore, we constructed a test dataset and we published it using the OpenLink Virtuoso [165] as data store to carry out simulation experiments and validate our WSC approach. The benchmark, discussed in this section, was executed on one 2.6 GHz Intel core i5 processor, 4 GB of RAM, and 64-bit Windows7 OS. Generally, a precise evaluation for a service composition method needs to analyze composite services built by the method and measure the execution time of the whole composition process. However, in contrast to traditional Web service composition methods, in our approach the composition should be regenerated at each step. Therefore, the feasibility of the approach depends on whether the composition cost is within reasonable limits. Consequently, an evaluation should assess whether the composition happens sufficiently fast for realistic composition scenarios and in presence of a reasonable number of services to be used in the composition process. Our composition strategy relies also on the use of EYE reasoner. Therefore, the performance of the approach depends also on how well this reasoner performs on creating proofs for compositions of varying lengths and complexity.

The main criteria we are considering are the composition speed-up, time consumption, and the execution time required for selecting the optimal instance for each capability candidate and generating the best executable composition. We claim that the composition time depends mainly from the time to determine the composition plan.

To conduct fair experiments, we need a sufficient number of services and ontologies with a variety of sizes. However, it is very hard to collect or manually construct appropriate data. For this reason, we have just used our programming dinner scenario, described in section 3.1.4, and we have varied the number of capabilities and instances (for each capability). Below, we depict the main experiments and results.

5.2.1 Parsing and Reasoning performance

The experiments performed at this level aim to study the performance of the EYE reasoner and consist in continuously increase the number of services and vary their associated dependencies. We use the benchmark [2] framework, developed for hypermedia API composition, and we assume that each service existing in the plan depends on one-to-many other services candidates. Results obtained by applying *EYE* and *cwm* [158] reasoners are depicted in figures 5.4(a) and 5.4(b) respectively.



(a) Benchmark results using EYE reasoner

(b) Benchmark results using cwm reasoner

FIGURE 5.4: Parsing and reasoning time using EYE and Cwm

As shown in figure 5.4, the overall time required to parse the existing atomic/composite services and reason over them is quite reasonable for both reasoners and increases linearly with the number of services descriptions as well as their related dependencies. In the simplest case, each service candidate depends only on its precedent service (d=1). The most complex case, in our evaluation scenario, assumes that each service depends on three services (d=3). However, the resulted benchmarks indicate that the EYE reasoner allows for faster processing of the existing descriptions and outputs answers in a constant way compared to *cwm*. For instance, reasoning over a plan length equal to 32 services with a number of dependencies d equal to 1 using EYE is reduced to 20 ms compared to that reasoned by *cwm*. This difference increases linearly according to the continuous increase of the number of services.

The main reason to the difference in performance between EYE and *cwm* is related to their reasoning mechanisms. EYE is a backward-chaining reasoner, which starts from the goal and works towards the initial state, whereas *cwm* is forward-chaining, exploring inferences from the initial state onwards until the goal has been reached.

[2] Available at http://github.com/RubenVerborgh/RESTdesc-Composition-Benchmark

Experiments depicted in figure 5.5 aim to measure the required time to generate plans with different sizes by increasing the number of capabilities, varying their related dependencies (we assumed that each capability depends on all the other capabilities) and fixing their corresponding parameters to 14 (4 preconditions and 10 postconditions). Results shown in figure 5.5 are perfectly acceptable. For instance, the generation of a plan with 7 capabilities, 6 dependencies and 14 parameters for each capability takes less than one second. Plans with more than 20 capabilities takes less than two seconds to be generated.

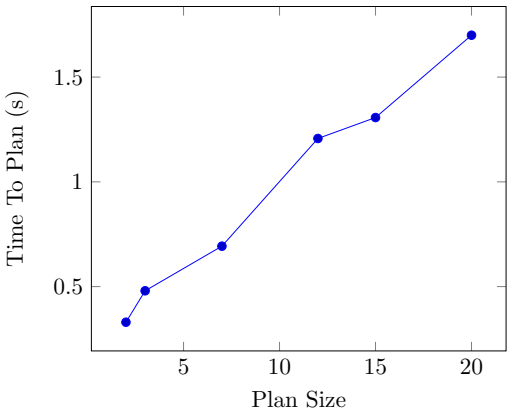


FIGURE 5.5: Time Required to generate abstract plans with different sizes

Moreover, we varied the number of attributes/predicates from 2 to 4 for the preconditions and from 6 to 10 for the effects of the candidate services. Figure 5.6 shows the impact of the number of preconditions and effects predicates on the required time of the composition.

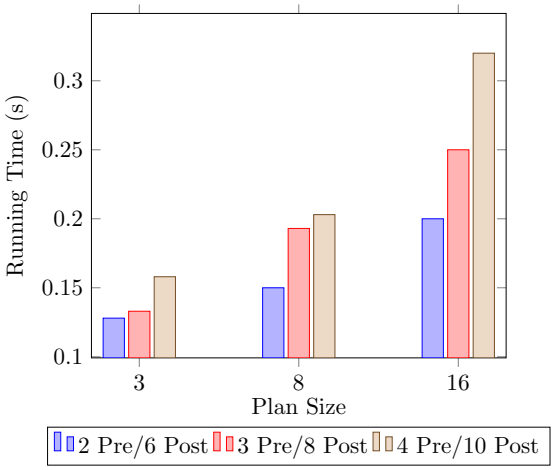


FIGURE 5.6: Required Time to generate abstract plans of different sizes varying the number of preconditions and postconditions

The most important observation is that, for a fixed size of plan, as the number of parameters increased, the planning time is increased. Which improves the effective impact of the number of parameters in the execution time.

5.2.2 QoS-Aware Service Selection performance

This section focuses on measuring the performance of generating the optimal composition based on a fixed number of QoS parameters. Experiments consist in measuring the time required to determine the optimal executable plan by varying the number of the abstract capabilities and the number of instances for each capability.

In figure 5.7, we keep constant the plan length (number of abstract capabilities) and vary the number of concrete instances. The plan length was fixed to 7 (7 abstract capabilities). the number of instance varies for each capability in the abstract plan. Results show that the execution time increases linearly according to the continuous increase of the number of instances.

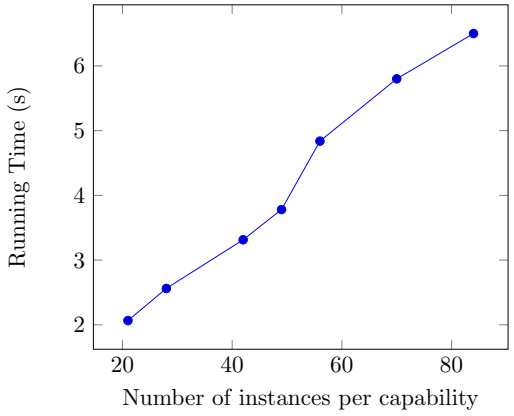


FIGURE 5.7: Time Required to identify optimal executable compositions with 7 abstract capabilities

In figure 5.8, we consider plans with different sizes (between 3 and 10 abstract capabilities) and vary the number of instances for each abstract capability existing in each plan. We measure the time required for binding the executable counterparts of each plan. It is obvious that when the number of capabilities/instances increases, the execution times for generating executable compositions increases accordingly. However, results show that the time to define an optimal composition increases constantly for different sizes of the plans and with different numbers of instances.

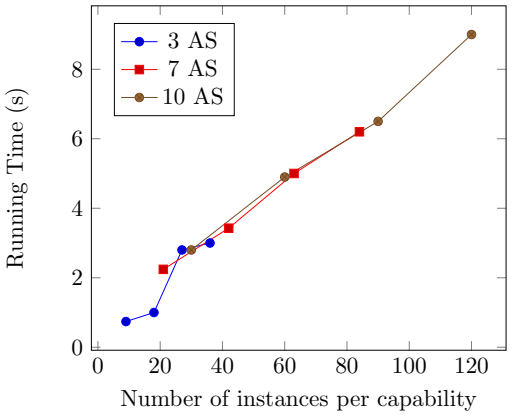


FIGURE 5.8: Time for optimal plan bindings

5.2.3 Comparative study

In this section, we focus on comparing the performance of our composition model with some existing approaches in the same context. Our comparative study consists of three parts: we start by comparing the first level (abstract composition) module of our composition framework, then the second part is devoted to compare the second level (concrete composition) module. Some experiments to evaluate the whole two-phase composition framework are carried out in the third part.

5.2.3.1 Comparison of DECSerComposer Reasoning-module with DSOL-Engine (DEng)

To this end, we measure the execution time of the composition process displayed by our framework and we apply a comparative study of the results with the performance benchmarks of DSOL-based framework, which is described in [12]. So that, we have compared the elapsed time for the path finding of our method to the DEng[3] tool and evaluated the performance of each method based on the results. We have measured the elapsed time to identify the composition paths using the time stamp of the system. For fair evaluation, the original implementations of both of the approaches were compared. To conduct fair experiments, we need a sufficient number of services and ontologies with a variety of sizes. As show in figure 5.9, we measure the time elapsed to find every composition path with varying the number of abstract services (AS) from 5 to 20 and each capability (AS) includes 4 preconditions and 10 postconditions. As illustrated by the experimental results, the time required to dynamically generate plans increases linearly with the increase of the number of capabilities. However, the time to plan taken by our DECSerComposer is quite reduced compared to that of DEng. This difference in

[3]https://github.com/leandroshp/dsol

planning time between the two composers can be dramatically increased in case of using a larger number of capabilities (tasks) in the composition scenario. Moreover, although the continuous increase of the size of the generated plans, the elapsed time to find path does not increase rapidly in our framework. Which makes our proposed approach a good candidate for composition process, especially over Web scale.

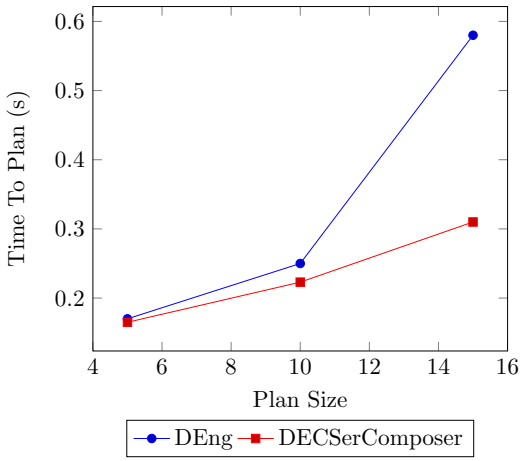


FIGURE 5.9: Time Required to generate plans with different sizes

5.2.3.2 Comparison of DECSerComposer Selection-module with MOACO

To verify the results obtained from the QoS-aware selection-module of our proposed framework, we carry out a comparative study with MOACO-based selection approach [64], which aims to provide an optimal Web service composition. Note that MOACO is considered one of the best multi-objective Ant Colony Optimization (ACO) algorithm for the bi-objective Traveling Salesman Problem (TSP) [166, 167]. To this end, we describe in table 5.1 different composition scenarios that are based on varying the number of both abstract services (S_t) and service instances (S_i) and we measure the time required to obtain optimal plan bindings for each scenario.

TABLE 5.1: Test Scenarios w.r.t number of service types and service instances

Test Scenario	S_t	S_i
1	5	10
2	10	20
3	20	20
4	20	40

Figure 5.10 shows the time required to identify the optimal concrete compositions. As it can be seen, for the same composition scenarios, MOACO-based approach is quietly outperforms our proposed selection module, which is slightly slower in term of execution

time. This can be explained by the high number of parameters (17 parameters) and the multiple mathematical steps used by our TOPSIS-based selector to identify the best instance according to each task existing in the plan. Moreover, the difference becomes smaller as the number of services instances increases, which improves the effectiveness of our approach at large scale data set like the Web.

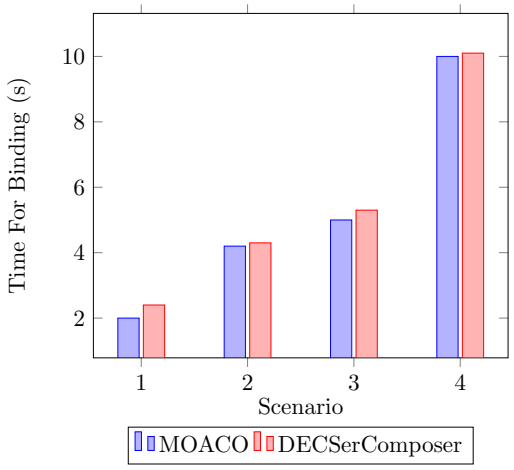


FIGURE 5.10: Time Required to generate optimal concrete compositions with different composition scenarios

5.2.3.3 Comparison with RFC and 2P approaches

In this section, we use a large scale data set to compare the scalability of our approach (DECSerComposer) with that of the two declarative approaches (RFC) presented in [168] and the two-phase approach (2P) proposed in [169, 170] with respect to the number of services (instances). We compared the three approaches based on the execution time. The execution time is the time spent on performing the Web services composition when a user sends a query to the system. To this extent, we randomly generate synthetic Web services data sets containing 10,000 Web services (instances). The parameters of Web services are uniformly distributed and each parameter is randomly assigned into a Web service. Then, we generate 1000 synthetic queries. The values of the parameters for these queries are randomly selected from the parameters of all generated Web services, and therefore the parameter domain of these queries has the same value as the parameter domain of the services set. The number of parameters are randomly chosen according to the number of parameters of the services set. Moreover, we consider a plan size of 3 capabilities in each generated query.

Figure 5.11 depicts the performance results for varying the number of Web services. Our approach performed significantly better then two phase algorithm and RFC system. As shown in the figure, the difference between the execution times of the three approaches

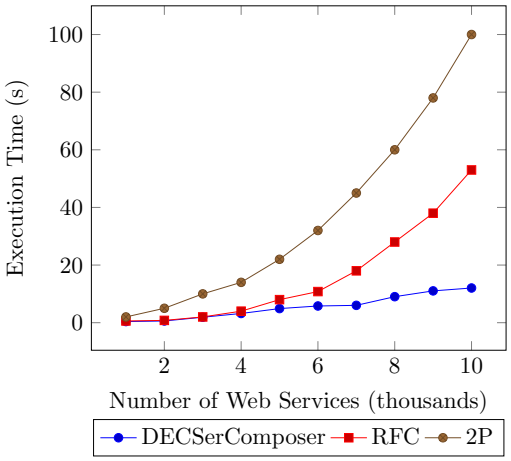


FIGURE 5.11: Comparison of our approach with RFC and 2P systems with $\#N_{parameters}$=2-6

increases linearly. This result was expected since, in our approach, we have introduced several optimizations compared to the two other approaches. First, the separation between services types and services instances helped in reducing the space search. Second, the reasoning mechanism has an important impact on the overall composition execution time.

5.3 Conclusion

In this chapter, we have discussed the implementation and studied the performance of the proposed approach. We have also compared our approach to three declarative approaches for Web services compositions, which are DSOL, NRC and 2P. Experiments results clearly show the added value of the proof based composition as a viable strategy to improve the composition process.

Regarding the limitations of the approach presented in this thesis, one observation concerns the abstraction level in modeling services abstract capabilities and their related instances. We have extensively modeled different components at an abstract level and although the models presented are very expressive, they may need to be modified and updated for modeling some concrete low-level details. As the proposed approach is extensible and is based on expressive capabilities, the proposed models can thus be modified and new ones can be added to handle other requirements such as those related to privacy dimensions or security properties. Then, the proposed tool for the process specification is in early phases and only serves as a proof of concept prototype. Although it can handle partial process specification, can automatically generate N3 models and can directly invoke the EYE reasoner and parse the results returned, it does not handle process verification and monitoring.

Chapter 6

Conclusions and Future Work

Contents

The current chapter summarizes this dissertation and provides potential avenues for future work based on the aforementioned contributions. We first review the problem description and motivations behind investigating an integrated declarative approach for Web services compositions. Then, we draw conclusions about the contributions of this work. The research addressed in this thesis is by no means complete and leaves enough room for further research. Consequently, we provide an outlook on several aspects that remain open as future work. We plan to address some of these issues in our ongoing research work.

6.1 Summary

6.1.1 Problem definition

Web services composition has been a major research topic in the past years. A vast number of service composition approaches have been proposed in literature. However, despite the huge number of research efforts and fast development of composition models

and approaches over the last years, several problems still need to be addressed. These problems are related to the proliferation of partial solutions, the lack of expressiveness and flexibility to handle functional and non-functional user requirements, the lack of integration, and the lack of integrated monitoring and recovery actions.

In this dissertation, the focus was on two major bottlenecks in the current process of modeling compositions. The first bottleneck is related to the expert level required to achieve a given composition. Typical procedural style of modeling, inspired by workflow/business process paradigms do not provide the required abstractions, and therefore fail to support dynamic, self-managed compositions and could not adapt to changes that can happen continuously, unpredictably, and inevitably lead to failures. A language such as the Business Process Execution Language for Web Services (WS-BPEL) is completely an expert language. Thus, specifying and developing a composition using WSBPEL is a lengthy, costly, and high-risk process. A second bottleneck in current services compositions concerns their life-cycle and their management, also called their governance. The challenge is how to achieve a full governance of the composition allowing its continuous and dynamic improvement. Traditional approaches focus only on some stages of process life-cycle and little initiatives, however, have been proposed to integrate these related criteria using a unified formalism. Also, it is not always possible to have a complete transformation between modeling approaches mainly if we consider non-functional requirements into account. Lack of integration presents a major barrier to learn from run-time failures and provide recovery actions.

The challenge at the ***modeling phase*** concerns the languages and models used to specify services and their interactions. Mainstream SOC languages, like BPEL and BPMN, are built with classical procedural constructors which contain explicit and complete information about the process flow. Although this adds a lot to the control over the composition process, however as there is trade off between the control and flexibility, this control comes at the expense of process flexibility and thus making the process rigid to adapt to continuously changing situations and possibly not even conforming to the process specification requirements.

Regarding the ***verification phase***, the focus is on the actions to take once composition specifications are given. Flexible composition of services and processes with non-functional concerns entails the danger that important rules or constraints of the service or process models get violated or overlooked. Today's state-of-the-art in verifying and validating instances of service or process models, however, can hardly cope with the complexity and dynamics of an end-to-end business compliance framework – both at design time and runtime. Furthermore, they are hard to use, especially for non-programmers. In addition, existing formal verification methods are not integrated with the existing

service or process models, and hence a semi-automated verification is hard to achieve in an end-to-end business compliance framework.

The proposed approaches for the composition verification, in general, require mapping the process (mostly defined using procedural approaches such as BPEL) to some formal logic (such as petri-nets, automata or process logic) and then using model checkers to verify the composition process. This transformation based approach has two major limitations, first the proposed verification approaches are based on traditional procedural approaches which have less expressibility, flexibility and adaptability and dynamism as compared to the declarative ones. Further, the limited expressiveness makes it difficult to verify the non-functional properties associated with the composition process.

Finally, at the ***monitoring phase***, the challenge is to define novel principles and techniques for cross-layer monitoring of composition processes, which is a challenging task due to the versatility and the dynamicity of a service composition. Monitoring frameworks should also deal with the scalability of the monitoring/analysis process, because it is crucial that the solution will be able to handle a large number of services, interactions, and events. Another challenge is dedicated to the feedback control and analysis of the composition, i.e. the conformance monitoring and analysis. It is therefore mandatory to be able to express constraints and properties to be monitored and analyzed. Then, the last challenge is to incorporate the monitoring /analysis process within the execution framework in order to have a more efficient approach when compared to solution based on external components. A major problem in traditional approaches is that the run-time monitoring activity which is tightly coupled with the composition process was not well integrated to these approaches, and very few proposals handles it by adding a new layer for the composition monitoring and thus do not provide the important execution time violations feedback to the composition process. One other common pattern of traditional approaches is they are highly procedural, which make the possibility to learn from run-time violations and to change the process instance/model at execution time very difficult. Once again, we believe that these proposals are more oriented to low-level analysis (services/components), while we are more interested by business activity monitoring. Moreover, the current monitoring systems depend mainly on the availability of tools that are able to specify requirements, to verify these requirements, to monitor if the execution is compliant to the specification, and to analyze what really happens during the execution of the composition.

To conclude, this dissertation is motivated by the requirement to investigate a novel approach for Web services composition that integrates the above stages of the process life-cycle in an unified and declarative way, thereby reducing development time and integration efforts.

6.1.2 Proposed approach

As discussed above, the motivation of our work stems from the process modeling, design-time verification, execution-time monitoring and adaptation in an integrated and declarative way to cater for dynamically changing situations. Our objective was twofold. First, we aim to provide a service specification language, designed with a declarative and logic-based approach and powered by reasoning mechanisms to meet both functional and non-functional user requirements and highly expressive interaction models without the need to over-specifying them. Second, using this declarative specification language, we target to develop a comprehensive and well-integrated framework to enable the mastery of complexity and dependability of service compositions by achieving a full governance of the composition. Declarative approach results in a highly flexible composition process that may be needed to cater for dynamically changing situations while integration simplifies the approach by using the similar formalism for composition design, verification and monitoring. To achieve this objective, we have proposed an integrated declarative framework for Web Services Composition Modeling and Engineering as depicted in figure 1.1. Based on the three stages of abstraction, composition, and monitoring, our solution provides an easy way to specify functional and non-functional requirements of composite services in a precise and declarative manner, and guides the user through the composition process while allowing detection of violations at both design and run time. The staged approach is designed keeping in mind the best knowledge engineering practices of modularity, conciseness, and scalability, while providing a fair amount of control to the composition process. It focuses on the specification of *what* level without having to state the *how*, which enables to preserve the autonomous nature of interacting services and represent expressive interaction models without the need to over-specifying them. Details of each stage was depicted in figure 3.6. The proposed framework offer required tools to specify requirements, to verify these requirements, to monitor if the execution is compliant to the specification, and to analyze what really happens during the execution of the composition. The key features of this framework are:

1. Users/Requesters provide the high-level description of the service desired (goal) using a user-friendly interface that enables end users to specify and express their requirements and preferences that mark the boundary of the solution (requested service) by following some instructions to build the query. A query parser is integrated to parse and validate the query by checking syntactic correctness and decoupling the functional requirements from the non-functional parameters (such as QoS properties). Because users provide high-level specification of the composite service which may not be realizable using the published component services, our framework guides the users for iterative refinement of the goal service specification.

2. A declarative capabilities driven specification that uses Notations3 to specify the components of the composition process and define patterns for specifying the functional and non-functional aspects for process specification. We use capabilities to express compositions, modeling both their functional and non-functional requirements. A Web service is described as a structured entity featured via a set of capabilities, non-functional features and workflow (pattern) properties (in case of a composite service). Such description considers a service as an access mechanism to a capability, which is, in its turn, a structured entity that describes what a service can do via an action verb and set of domain-specific attributes. Different services can be interconnected at different levels of abstraction/concreteness by establishing links between them.

3. A proof based approach using EYE reasoner for the process design-time verification. The capabilities driven description is used to provide a two staged Web service composition approach, which is purely declarative and support flexible self-managed compositions. First, an abstraction stage consists in constructing a composition of available services that provide the desired functionality by semantically generating a composition plan of abstract capabilities. Second, a concrete stage concretizes the abstract composition into an executable composition by selecting the appropriate concrete capabilities instances based on non functional aspects using the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS). Verification is applied both to the single services and to the whole, dynamically evolving, composition. Proofs obtained through formal verification could be provided and advertised. In this way, other services can reason upon the declarative specifications, and possibly check the advertised proofs of properties, leaving no doubts on the dependability on the services.

4. An event-based monitoring framework that allows to reason about the events and does not require defining and extracting events from process specification, as the events are first class objects of both design and monitoring framework. As the proposed monitoring approach builds upon capabilities and N3 based composition design, it allows for the specification of monitoring properties that are based on both functional and non-functional (such as temporal, security or their combinations) requirements. These properties are expressed as N3 formulas and can be added to the process specification both during process design and during the process execution. A continuous monitoring of the on-line behavior of services enable to provide reliable trust levels and a dependable quality of service.

5. Implementation of the above key features in a composition management system that realizes all algorithms and models proposed in the dissertation. This management system presents all the functionalities from design phase to monitoring and provides the corresponding tools for each phase. It was tested and evaluated using two real scenarios such as multi-sensor surveillance application.

6.2 Outlook and Future Work

In the following, we provide an outlook on several aspects that remain open as future work. We plan to address some of these issues in our ongoing research work.

As for the composition system resulting from the implementation of the different modules, while the current prototype is fully operational, we think there is still space to further improve performance and also to improve reliability and robustness. First, we are working on increasing the expressiveness of the capabilities based modeling by adding other types of QoS facilities into the Composer. In the service management layer of the SOC stack [10], QoS can be expressed on a high level in form of Service Level Agreement (SLAs) between two partners, which is guaranteed by service advertisements. Second, We plan to extend the composer to improve the run-time support for adding new tasks/services to a running composition, allowing the service architect to easily intervene to overcome the most complex exceptions that may happen during a long running orchestration. Third, to fully validate our approach we want to test its feasibility in additional real world case studies before integrating it in a widely adopted tool such as Eclipse.

Some other aspects remain open as future work and are outlined below.

6.2.1 Services mashups and cloud service compositions

With the development of Web 2.0 technologies, one noteworthy trend over the Web is the rapid growing services mashups [171] which combines existing services, such as Web APIs, RESTful services into a single integrated service. Most service mashup solutions are semi-automatic which assume that all available Web resources are known and available on the Web. User Generated Services (UGS) enable users to build mashups by finding, combining and reusing Web resources manually. However, to facilitate the creation of service mashups, an automated composition way is highly needed. Regarding cloud services [172] which are developed as self-contained component, how to compose services in cloud environments and achieve high resource utility which is customized for

client requests has become an important research issue. In order to realize such composition, multi-attribute semantics involving functional and non-functional semantics should be taken into account for the service selection and planning procedure.

To this end, our composition strategy can be further developed and extended for the service mashups and the cloud service composition by parsing the semantics from annotated Web resources and modifying the problem modeling component for the further reuse.

6.2.2 Quality of Experience driven Service Composition

Current work in web service selection and discovery clearly show that non-functional aspects, and mainly QoS parameters such as response time and availability, are exploited as the key decision making criteria. However, handling the QoS information may face some problems such as:

- the QoS information may not be updated frequently to reflect a specific environment and platform.
- the QoS information published by service providers is often limited and cannot respond to various user concerns about service quality.
- the QoS information does not reflect the end-user's perspective on the quality of services.
- the QoS information does not take into account the user feedback.

One alternative to overcome the above difficulties is to consider Quality of Experience (QoE) attributes to guide service selection to generate an optimal execution path that meets a user imposed QoE constraints. Contrary to QoS, QoE reflects quality from the end user point of view. The primary source of QoE is on-line reviews. Reviews come from users with diverse platforms and different geographical locations. Hence it is more credible source of information. End-users express their experience via online reviews to reveal their satisfactions and disappointments about services. The idea in this direction is to study the feasibility of adopting the perceived quality from end-user's perspective, as a measure customer satisfaction with a service, for service selection and composition. However, the task of extracting QoE attributes from user reviews is challenging. User reviews are written in natural language and presented as unstructured data. Therefore, it is not trivial for computers to understand, analyze, and aggregate QoE from the web [173], [174], [175]. Without an automatic aggregation and search tool, finding and

going through a large number of reviews to manually find QoE information for service selection and composition is not feasible. Several research questions can be considered: How to extract QoE from online reviews? How do QoE attributes relate with QoS attributes? What about the effectiveness of QoE attributes in a service composition process? How to handle the problem if we cannot find enough review data available mainly for new and unpopular services? How to address bootstrapping problem for QoE attribute identification?

To this end, our composition strategy can be further developed and extended for the QoE driven composition by incorporating an automatic tool for mining QoE from user reviews. This tool will enable to analyze the natural language content, identify QoE attributes, and represent them in a structured way that can be used by our service composition algorithms.

6.2.3 Pervasive services composition

The proliferation of ubiquitous, interconnected computing devices (e.g., PDAs, 3G mobile phones), as well as recent advances in radio-frequency identification (RFID) technology and sensor networks, are fostering the emergence of environments where Internet applications and services made available to mobile users are a commodity [176], [177], [178], [179]. Composing services across multiple mobile devices in such an environment presents new challenges that do not occur in traditional services composition settings [177]. In particular, composition mechanisms in pervasive environments need to address context awareness, heterogeneity and contingencies of devices (e.g., unpredictable availability of services and mobile devices), and personalization (e.g., service provisioning based on user preferences). Since the devices where services are running are usually resource constraint (e.g., limited memory and battery life), special considerations are necessary for the efficiency and performance of composite services. From our analysis of existing services composition prototypes (see Section 5), it is clear that relatively few research focuses on services composition in pervasive environments. Extensive research efforts are therefore needed in this direction.

Bibliography

[1] Ruben Verborgh and Jos De Roo. Drawing conclusions from linked data on the web. *IEEE Software*, 32(5), May 2015. URL http://online.qmags.com/ISW0515?cid=3244717&eid=19361&pg=25.

[2] A. R. Riad and Q. F. Hassan. Service-oriented architecture - a new alternative to traditional integration methods in b2b applications. *JCIT*, 3(1):31–41, 2008. URL http://dblp.uni-trier.de/db/journals/jcit/jcit3.html#RiadH08.

[3] Ahmed Elfatatry. Dealing with change: Components versus services. *Commun. ACM*, 50(8):35–39, August 2007. ISSN 0001-0782. doi: 10.1145/1278201.1278203. URL http://doi.acm.org/10.1145/1278201.1278203.

[4] Srinivas Padmanabhuni, Jai Ganesh, and Deependra Moitra. Web services, grid computing, and business process management: Exploiting complementarities for business agility. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04), June 6-9, 2004, San Diego, California, USA*, pages 666–673, 2004. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2004.127.

[5] Marc N. Haines and Marcus A. Rothenberger. How a service-oriented architecture may change the software development process. *Commun. ACM*, 53(8):135–140, August 2010. ISSN 0001-0782. doi: 10.1145/1787234.1787269. URL http://doi.acm.org/10.1145/1787234.1787269.

[6] M. P. Papazoglou and D. Georgakopoulos. Introduction: Service-oriented computing. *Commun. ACM*, 46(10):24–28, October 2003. ISSN 0001-0782. doi: 10.1145/944217.944233. URL http://doi.acm.org/10.1145/944217.944233.

[7] H. Rajan and M. Hosamani. Tisa: Toward trustworthy services in a service-oriented architecture. *Services Computing, IEEE Transactions on*, 1(4):201–213, Oct 2008. ISSN 1939-1374.

[8] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *Int. J. Web Grid Serv.*, 1(1):1–30, August 2005. ISSN 1741-1106. doi: 10.1504/IJWGS.2005.007545. URL http://dx.doi.org/10.1504/IJWGS.2005.007545.

[9] George Baryannis, Olha Danylevych, Dimka Karastoyanova, Kyriakos Kritikos, Philipp Leitner, Florian Rosenberg, and Branimir Wetzstein. Service composition. In *Service Research Challenges and Solutions for the Future Internet - S-Cube - Towards Engineering, Managing and Adapting Service-Based Systems*, pages 55–84, 2010. doi: 10.1007/978-3-642-17599-2_3. URL http://dx.doi.org/10.1007/978-3-642-17599-2_3.

[10] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, November 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.400. URL http://dx.doi.org/10.1109/MC.2007.400.

[11] Girish Chafle, Gautam Das, Koustuv Dasgupta, Arun Kumar, Sumit Mittal, Sougata Mukherjea, and Biplav Srivastava. An integrated development environment for web service composition. In *ICWS*, pages 839–847. IEEE Computer Society, 2007. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2007.38.

[12] Gianpaolo Cugola, Carlo Ghezzi, and Leandro Sales Pinto. DSOL: a declarative approach to self-adaptive service orchestrations. *Computing*, 94(7):579–617, 2012. doi: 10.1007/s00607-012-0194-z. URL http://dx.doi.org/10.1007/s00607-012-0194-z.

[13] Guoquan Wu, Jun Wei, Chunyang Ye, Xiaozhe Shao, Hua Zhong, and Tao Huang. Runtime monitoring of data-centric temporal properties for web services. In *IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, July 4-9, 2011*, pages 161–170, 2011. doi: 10.1109/ICWS.2011.124. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2011.124.

[14] Zhanlei Ma, Lin Liu, Hongji Yang, and John Mylopoulos. Adaptive service composition based on runtime requirements monitoring. In *ICWS*, pages 339–346, 2011. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2011.83.

[15] Annapaola Marconi, Marco Pistore, Piero Poccianti, and Paolo Traverso. Automatedweb service composition at work: the amazon/mps case study. In *ICWS*, pages 767–774, 2007. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2007.50.

[16] William N. Robinson. Monitoring web service requirements. In *11th IEEE International Conference on Requirements Engineering (RE 2003), 8-12 September 2003, Monterey Bay, CA, USA.*, pages 65–74, 2003. doi: 10.1109/ICRE.2003.1232738. URL http://dx.doi.org/10.1109/ICRE.2003.1232738.

[17] Gustav Bostrom, Pablo Giambiagi, and Tomas Olsson. Quality of service evaluation in virtual organizations using slas. In *Interoperability for Enterprise Software and Applications*, pages 211–224. ISTE, 2010. ISBN 9780470612200. doi: 10.1002/9780470612200.ch18. URL http://dx.doi.org/10.1002/9780470612200.ch18.

[18] Danilo Ardagna, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani. PAWS: A framework for executing adaptive web-service processes. *IEEE Software*, 24(6):39–46, 2007. doi: 10.1109/MS.2007.174. URL http://doi.ieeecomputersociety.org/10.1109/MS.2007.174.

[19] Luciano Baresi, Carlo Ghezzi, and Sam Guinea. Smart monitors for composed services. In *Proceedings of the 2Nd International Conference on Service Oriented Computing*, ICSOC '04, pages 193–202, New York, NY, USA, 2004. ACM. ISBN 1-58113-871-7. doi: 10.1145/1035167.1035195. URL http://doi.acm.org/10.1145/1035167.1035195.

[20] Khaled Mahbub and George Spanoudakis. A framework for requirents monitoring of service based systems. In *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA, November 15-19, 2004, Proceedings*, pages 84–93, 2004. doi: 10.1145/1035167.1035181. URL http://doi.acm.org/10.1145/1035167.1035181.

[21] Halvard Skogsrud, Boualem Benatallah, and Fabio Casati. A trust negotiation system for digital library web services. *Int. J. on Digital Libraries*, 4(3):185–207, 2004. doi: 10.1007/s00799-004-0083-y. URL http://dx.doi.org/10.1007/s00799-004-0083-y.

[22] Salima Benbernou, Hassina Meziane, Yin Hua Li, and Mohand-Said Hacid. A privacy agreement model for web services. In *IEEE SCC*, pages 196–203. IEEE Computer Society, 2007. URL http://doi.ieeecomputersociety.org/10.1109/SCC.2007.14.

[23] Ehtesham Zahoor, Olivier Perrin, and Claude Godart. An integrated declarative approach to web services composition and monitoring. In *Web Information Systems Engineering - WISE 2009, 10th International Conference, Poznan, Poland, October 5-7, 2009. Proceedings*, pages 247–260, 2009. doi: 10.1007/978-3-642-04409-0_28. URL http://dx.doi.org/10.1007/978-3-642-04409-0_28.

[24] Ehtesham Zahoor, Olivier Perrin, and Claude Godart. Disc: A declarative framework for self-healing web services composition. In *ICWS*, pages 25–33. IEEE Computer Society, 2010. ISBN 978-0-7695-4128-0. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2010.70.

[25] Tim Berners-Lee and Dan Connolly. Notation3 (n3): A readable rdf syntax. W3c team submission, W3C, January 2008. URL http://www.w3.org/TeamSubmission/n3/.

[26] Uwe Keller, Rubén Lara, Holger Lausen, Axel Polleres, and Dieter Fensel. Automatic location of services. In *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, pages 1–16. 2005. doi: 10.1007/11431053_1. URL http://dx.doi.org/10.1007/11431053_1.

[27] Sami Bhiri, Wassim Derguech, and Maciej Zaremba. Modelling capabilities as attribute-featured entities. In José Cordeiro and Karl-Heinz Krempels, editors, *WEBIST (Selected Papers)*, volume 140 of *Lecture Notes in Business Information Processing*, pages 70–85. Springer, 2012. ISBN 978-3-642-36607-9. URL http://dx.doi.org/10.1007/978-3-642-36608-6_5.

[28] Jos De Roo. Euler yet another proof engine, 1999–2013. URL http://eulersharp.sourceforge.net/.

[29] Web services glossary. http://www.w3.org/TR/ws-gloss/. Accessed: 2004-02-11.

[30] Web services architecture. URL http://www.w3.org/TR/2002/WD-ws-arch-20021114/. Accessed: 2002-11-14.

[31] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[32] Cesare Pautasso and Erik Wilde. Restful web services: principles, patterns, emerging technologies. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 1359–1360, 2010. doi: 10.1145/1772690.1772929. URL http://doi.acm.org/10.1145/1772690.1772929.

[33] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 805–814, 2008. doi: 10.1145/1367497.1367606. URL http://doi.acm.org/10.1145/1367497.1367606.

[34] Victor Saquicela, Luis Manuel Vilches Blázquez, and Óscar Corcho. Adding semantic annotations into (geospatial) restful services. *Int. J. Semantic Web Inf. Syst.*, 8(2):51–71, 2012. URL http://dx.doi.org/10.4018/jswis.2012040103.

[35] Richard Hull and Jianwen Su. Tools for composite web services: a short overview. *SIGMOD Record*, 34(2):86–95, 2005. doi: 10.1145/1083784.1083807. URL http://doi.acm.org/10.1145/1083784.1083807.

[36] Ulrich Köster, Mirco Stern, and Birgitta K. Ries. A Classification of Issues and Approaches in Automatic Service Composition. In *First International Workshop on Engineering Service Compositions (WESC05)*, Amsterdam, Netherlands, December 2005.

[37] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C.K. Liu, R. Khalaf, Dieter Koenig, M. Marin, V. Mehta, S. Thatte, D. Rijn, P. Yendluri, and A. Yiu. Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.

[38] Frank Leymann. Web Services Flow Language (WSFL 1.0). Technical report, IBM, may 2001. URL http://xml.coverpages.org/WSFL-Guide-200110.pdf.

[39] Cesare Pautasso and Gustavo Alonso. The jopera visual composition language. *Journal of Visual Languages and Computing (JVLC)*, 16:119–152, 2005. URL http://dx.doi.org/10.1016/j.jvlc.2004.08.004.

[40] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, and Sinisa Zimek. Web service choreography interface (wsci) 1.0. World Wide Web Consortium, Note NOTE-wsci10-20020808, August 2002. URL http://www.w3.org/TR/2002/NOTE-wsci-20020808.

[41] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web services choreography description language version 1.0 (w3c candidate recommendation). 2005. URL http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.

[42] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Bpel4chor: Extending bpel for modeling choreographies. In *ICWS*, pages 296–303. IEEE Computer Society, 2007. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2007.59.

[43] OMG. Business Process Model and Notation (BPMN), Version 2.0. Object Management Group, January 2011. URL http://www.omg.org/spec/BPMN/2.0.

[44] Farhan Hassan Khan, Saba Bashir, M. Younus Javed, Aihab Khan, and Malik Sikandar Hayat Khiyal. Qos based dynamic web services composition & execution. *CoRR*, abs/1003.1502, 2010. URL http://arxiv.org/abs/1003.1502.

[45] Louis Felipe Cabera, George Copeland, Max Feingold, Tom Freund, Jim Johnson, Chris Kaler, Johannes Klein, David Langworthy, Anthony Nadalin, David Orchard, Ian Robinson, Tony Storey, and Satish Thatte. Web Services Atomic Transaction (WS-AtomicTransaction). Technical report, BEA Systems, International Business Machines Corporation, Microsoft Corporation Inc., November 2004.

[46] Louis Felipe Cabera, George Copeland, Tom Freund, Johannes Klein, David Langworthy, Frank Leymann, David Orchard, Ian Robinson, Tony Storey, and Satish Thatte. Web Services Business Activity Framework (WS-BusinessActivity). Technical report, BEA Systems Inc, IBM Corporation, Microsoft Corporation, November 2004.

[47] Luis Felipe Cabera, George Copeland, Max Feingold, Tom Freund, Jim Johnson, Chris Kaler, Johannes Klein, David Langworthy, Anthony Nadalin, David Orchard, Ian Robinson, John Shewchuk, and Tony Storey. Web Service Coordination (WS-Coordination). Technical report, BEA Systems, International Business Machines Corporation, Microsoft Corporation Inc., November 2004.

[48] Sami Bhiri, Walid Gaaloul, Mohsen Rouached, and Manfred Hauswirth. Semantic web services for satisfying SOA requirements. In *Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web*, pages 374–395. 2009. doi: 10.1007/978-3-540-89784-2_15.

[49] David Martin et al. Owl-s: Semantic markup for web services, 2004. URL http://www.w3.org/Submission/OWL-S/.

[50] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language overview. Technical Report REC-owl-features-20040210, W3C, 2004.

[51] John Domingue Dieter Fensel Martin Hepp Uwe Keller Michael Kifer Birgitta König-Ries Jacek Kopecky Rubén Lara Holger Lausen Eyal Oren Axel Polleres Dumitru Roman James Scicluna Michael Stollberg Jos de Bruijn, Christoph Bussler. Web service modeling ontology (wsmo), 2005. URL http://www.w3.org/Submission/WSMO/.

[52] Christina Feier et al. Towards intelligent web services: the web service modeling ontology (wsmo). In *2005 International Conference on Intelligent Computing (ICIC'05)*, 2005. URL http://oro.open.ac.uk/23147/.

[53] Kanmani Munusamy et al. Semantic web service process mediation in wsmo:current solutions and open issues. Rome, Italy, September 2011. ThinkMind, SERVICE COMPUTATION 2011, The Third International Conferences on Advanced Service Computing. ISBN 978-1-61208-152-6.

[54] Liliana Cabral, John Domingue, Stefania Galizia, Alessio Gugliotta, Vlad Tanasescu, Carlos Pedrinaci, and Barry Norton. Irs-iii: A broker for semantic web services based applications. In Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and LoraM. Aroyo, editors, *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 201–214. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-49029-6. doi: 10.1007/11926078_15. URL http://dx.doi.org/10.1007/11926078_15.

[55] Enrico Motta. An overview of the ocml modelling language. In *In Proceedings KEML'98: 8th Workshop on Knowledge Engineering Methods and Languages*, pages 21–22, 1998.

[56] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 553–562, New York, NY, USA, 2004. ACM. ISBN 1-58113-844-X. doi: 10.1145/988672.988747. URL http://doi.acm.org/10.1145/988672.988747.

[57] Kunal Verma, Karthik Gomadam, Amit P. Sheth, John A. Miller, and Zixin Wu. The meteor-s approach for configuring and executing dynamic web processes. Technical report, University of Georgia, Athens, June 2005. URL http://lsdis.cs.uga.edu/projects/meteor-s/techRep6-24-05.pdf.

[58] Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6):60–67, November 2007. ISSN 1089-7801. doi: 10.1109/MIC.2007.134. URL http://dx.doi.org/10.1109/MIC.2007.134.

[59] Kunal Verma and Amit Sheth. Semantically annotating a web service. *IEEE Internet Computing*, 11(2):83–85, 2007. ISSN 1089-7801. doi: http://doi.ieeecomputersociety.org/10.1109/MIC.2007.48.

[60] Lianyong Qi, Ying Tang, Wanchun Dou, and Jinjun Chen. Combining local optimization and enumeration for qos-aware web service composition. In *Proceedings of the 2010 IEEE International Conference on Web Services*, ICWS '10, pages 34–41, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4128-0. doi: 10.1109/ICWS.2010.62. URL http://dx.doi.org/10.1109/ICWS.2010.62.

[61] Zachary J. Oster, Ganesh Ram Santhanam, and Samik Basu. Identifying optimal composite services by decomposing the service composition problem. In *Proceedings of the 2011 IEEE International Conference on Web Services*, ICWS '11, pages 267–274, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4463-2. doi: 10.1109/ICWS.2011.110. URL http://dx.doi.org/10.1109/ICWS.2011.110.

[62] Zachary J. Oster, Ganesh Ram Santhanam, and Samik Basu. Decomposing the service composition problem. In *Proceedings of the 2010 Eighth IEEE European Conference on Web Services*, ECOWS '10, pages 163–170, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4310-9. doi: 10.1109/ECOWS.2010.15. URL http://dx.doi.org/10.1109/ECOWS.2010.15.

[63] Mahdi Bakhashi, Abbas Olfat, Ghasem Olfat, and Farhad Mardukhi. A fuzzy-based user-centric approach for selecting the optimal composition of services. *Journal of Theoretical and Applied Information Technology*, pages 72–79, 2005.

[64] Fang Qiqing, Peng Xiaoming, Liu Qinghua, and Hu Yahui. A global qos optimizing web services selection algorithm based on moaco for dynamic web service composition. 1:37–42, May 2009. doi: 10.1109/IFITA.2009.91.

[65] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 881–890, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi: 10.1145/1526709.1526828. URL http://doi.acm.org/10.1145/1526709.1526828.

[66] Peter Bartalos and Mária Bieliková. Automatic dynamic web service composition: A survey and problem formalization. *Computing and Informatics*, 30(4):793–827, 2011. URL http://www.cai.sk/ojs/index.php/cai/article/view/198.

[67] V.R. Chifu, I. Salomie, A. Riger, and V. Radoi. A graph based backward chaining method for web service composition. In *Intelligent Computer Communication and Processing, 2009. ICCP 2009. IEEE 5th International Conference on*, pages 237–244, Aug 2009. doi: 10.1109/ICCP.2009.5284755.

[68] Yan Xu, Bin Li, and Jun Wu. A petri-net coverability model for automatic web service composition. In *Industrial and Information Systems, 2009. IIS '09. International Conference on*, pages 31–34, April 2009. doi: 10.1109/IIS.2009.20.

[69] Antonio Brogi and Sara Corfini. Ontology- and behavior-aware discovery of web service compositions. *Int. J. Cooperative Inf. Syst.*, 17(3):319–347,

2008. doi: 10.1142/S0218843008001853. URL http://dx.doi.org/10.1142/S0218843008001853.

[70] Shalil Majithia, David W. Walker, and W. A. Gray. A framework for automated service composition in service-oriented architectures. In Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *ESWS*, volume 3053 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2004. ISBN 3-540-21999-4. URL http://dx.doi.org/10.1007/978-3-540-25956-5_19.

[71] Bochao Wang, A. Haller, and F. Rosenberg. Generating workflow models from owl-s service descriptions with a partial-order plan construction. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 714–715, July 2011. doi: 10.1109/ICWS.2011.88.

[72] Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972. doi: 10.1137/0201008. URL http://dx.doi.org/10.1137/0201008.

[73] Keita Fujii and Tatsuya Suda. Semantics-based dynamic web service composition. *Int. J. Cooperative Inf. Syst.*, 15(3):293–324, 2006. URL http://dx.doi.org/10.1142/S0218843006001372.

[74] Keita Fujii and Tatsuya Suda. Semantics-based context-aware dynamic service composition. *ACM Trans. Auton. Adapt. Syst.*, 4(2):12:1–12:31, May 2009. ISSN 1556-4665. doi: 10.1145/1516533.1516536. URL http://doi.acm.org/10.1145/1516533.1516536.

[75] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic service composition based on behavioral descriptions. *Int. J. Cooperative Inf. Syst.*, 14(4):333–376, 2005. doi: 10.1142/S0218843005001201. URL http://dx.doi.org/10.1142/S0218843005001201.

[76] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, and Fabio Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2):429–451, 2008. doi: 10.1142/S0129054108005759. URL http://dx.doi.org/10.1142/S0129054108005759.

[77] Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-time monitoring of instances and classes of web service compositions. In *2006 IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA*, pages 63–71, 2006. doi: 10.1109/ICWS.2006.113. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2006.113.

[78] R. Kazhamiakin and M. Pistore. Static verification of control and data inweb service compositions. In *Web Services, 2006. ICWS '06. International Conference on*, pages 83–90, Sept 2006. doi: 10.1109/ICWS.2006.124.

[79] Baojun Tian and Yanlin Gu. Formal modeling and verification for web service composition. pages 2733–2737, 2013.

[80] René David and Hassane Alla. Petri nets for modeling of dynamic systems: A survey. *Automatica*, 30(2):175 – 202, 1994. ISSN 0005-1098. doi: http://dx.doi.org/10.1016/0005-1098(94)90024-8. URL http://www.sciencedirect.com/science/article/pii/0005109894900248.

[81] Idir Aït Sadoune and Yamine Aït Ameur. A proof based approach for formal verification of transactional bpel web services. In *Proceedings of the Second International Conference on Abstract State Machines, Alloy, B and Z*, ABZ'10, pages 405–406, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-11810-0, 978-3-642-11810-4. doi: 10.1007/978-3-642-11811-1_39. URL http://dx.doi.org/10.1007/978-3-642-11811-1_39.

[82] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010. ISBN 0521895561, 9780521895569.

[83] Mohamed Graiet, Raoudha Maraoui, Mourad Kmimech, Mohamed Tahar Bhiri, and Walid Gaaloul. Towards an approach of formal verification of mediation protocol based on web services. In *iiWAS'2010 - The 12th International Conference on Information Integration and Web-based Applications and Services, 8-10 November 2010, Paris, France*, pages 75–82, 2010. doi: 10.1145/1967486.1967502. URL http://doi.acm.org/10.1145/1967486.1967502.

[84] Giusy Di Lorenzo, Nicola Mazzocca, Francesco Moscato, and Valeria Vittorini. Towards semantics driven generation of executable web services compositions. *JSW*, 2(5):1–15, 2007. URL http://dx.doi.org/10.4304/jsw.2.5.1-15.

[85] G. Di Lorenzo, F. Moscato, N. Mazzocca, and V. Vittorini. Automatic analysis of control flow inweb services composition processes. In *Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on*, pages 299–306, Feb 2007. doi: 10.1109/PDP.2007.27.

[86] SU Huan-cheng, HUANG Zhi-qiu, and LIU Lin-yuan. Interface automata-based formal model for bpel4ws web service composition. *Application Research of Computers*, page 1774–1777, 2009.

[87] Claus Traulsen, Jérôme Cornet, Matthieu Moy, and Florence Maraninchi. A SystemC/TLM semantics in Promela and its possible applications. In *14th Workshop on Model Checking Software SPIN*, July 2007.

[88] Gerard Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003. ISBN 0-321-22862-6.

[89] M. Emilia Cambronero, Gregorio Díaz, Valentín Valero, and Enrique Martínez. Validation and verification of web services choreographies by using timed automata. *The Journal of Logic and Algebraic Programming*, 80:25 – 49, 2011. ISSN 1567-8326. doi: http://dx.doi.org/10.1016/j.jlap.2010.02.001. URL http://www.sciencedirect.com/science/article/pii/S1567832610000032. The 2nd Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'08).

[90] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2015. ISSN 1433-2779. doi: 10.1007/s10009-014-0361-y.

[91] Shuhao Li, Sandie Balaguer, Alexandre David, Kim G. Larsen, Brian Nielsen, and Saulius Pusinskas. Scenario-based verification of real-time systems using uppaal. *Formal Methods in System Design*, 37(2-3):200–264, 2010.

[92] Muthumanickam Krishnan Danapaquiame Nagamouttou, Ilavarasan Egambaram and Poonkuzhali Narasingam. A verification strategy for web services composition using enhanced stacked automata model. *SpringerPlus 2015*, 27 February 2015.

[93] Asit Dan, Doug Davis, Robert Kearney, Alexander Keller, Richard P. King, Dietmar Kuebler, Heiko Ludwig, Mike Polan, Mike Spreitzer, and Alaa Youssef. Web services on demand: Wsla-driven automated management. *IBM Systems Journal*, 43(1):136–158, 2004. doi: 10.1147/sj.431.0136. URL http://dx.doi.org/10.1147/sj.431.0136.

[94] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. Comprehensive qos monitoring of web services and event-based sla violation detection. In *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing*, MWSOC '09, pages 1–6, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-848-3. doi: 10.1145/1657755.1657756. URL http://doi.acm.org/10.1145/1657755.1657756.

[95] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. End-to-end support for qos-aware service selection, binding, and mediation in

vresco. *IEEE Trans. Serv. Comput.*, 3(3):193–205, July 2010. ISSN 1939-1374. doi: 10.1109/TSC.2010.20. URL http://dx.doi.org/10.1109/TSC.2010.20.

[96] S–Cube Network of Excellence. Software services and systems, 2012. URL http://www.s-cube-network.eu/.

[97] Sam Guinea, Gabor Kecskemeti, Annapaola Marconi, and Branimir Wetzstein. Multi-layered monitoring and adaptation. In *Proceedings of the 9th International Conference on Service-Oriented Computing*, ICSOC'11, pages 359–373, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-25534-2. doi: 10.1007/978-3-642-25535-9_24. URL http://dx.doi.org/10.1007/978-3-642-25535-9_24.

[98] Bruno Wassermann and Wolfgang Emmerich. Monere: Monitoring of service compositions for failure diagnosis. In Gerti Kappel, Zakaria Maamar, and Hamid R. Motahari Nezhad, editors, *ICSOC*, volume 7084 of *Lecture Notes in Computer Science*, pages 344–358. Springer, 2011. ISBN 978-3-642-25534-2. URL http://dx.doi.org/10.1007/978-3-642-25535-9_23.

[99] Michaël Mrissa and Mohand-Said Hacid. Combining configuration and query rewriting for Web service composition. Technical Report RR-LIRIS-2009-045, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/École Centrale de Lyon, December 2009. URL http://liris.cnrs.fr/publis/?id=4563.

[100] Mahmoud Barhamgi and Djamal Benslimane. Composing data-providing web services. In *VLDB PhD Workshop*, 2009. URL http://www.vldb.org/pvldb/2/vldb09-1058.pdf.

[101] Karim Benouaret, Djamal Benslimane, Allel Hadjali, and Mahmoud Barhamgi. Top-k web service compositions using fuzzy dominance relationship. In *Proceedings of the 2011 IEEE International Conference on Services Computing*, SCC '11, pages 144–151, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4462-5. doi: 10.1109/SCC.2011.86. URL http://dx.doi.org/10.1109/SCC.2011.86.

[102] Didier Dubois, Hung T. Nguyen, and Henri Prade. Possibility theory, probability and fuzzy sets: misunderstandings, bridges and gaps. . In D. Dubois and H. Prade, editors, *Fundamentals of Fuzzy Sets* , The Handbooks of Fuzzy Sets Series, pages 343–438. Kluwer, Boston, Mass., 2000.

[103] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, Amsterdam, 2004. ISBN 978-1-55860-856-6. URL http://www.sciencedirect.com/science/book/9781558608566.

[104] Incheon Paik and Daisuke Maruyama. Automatic web services composition using combining HTN and CSP. In *Seventh International Conference on Computer and Information Technology (CIT 2007), October 16-19, 2007, University of Aizu, Fukushima, Japan*, pages 206–211, 2007. doi: 10.1109/CIT.2007.61. URL http://dx.doi.org/10.1109/CIT.2007.61.

[105] Ronny Hartanto and Joachim Hertzberg. Fusing dl reasoning with htn planning. In *Proceedings of the 31st annual German conference on Advances in Artificial Intelligence*, KI '08, pages 62–69, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85844-7. doi: 10.1007/978-3-540-85845-4_8. URL http://dx.doi.org/10.1007/978-3-540-85845-4_8.

[106] Wan M.N. Wan Kadir Sayed Gholam Hassan Tabatabaei and Suhaimi Ibrahim. A semantic web service discovery and composition based on htn-planning and description logic. 2013. URL http://comp.utm.my/pars/files/2013/04/Semantic-Web-Service-Discovery-and-Composition-Based-on-HTN-Planning-and-Descrip pdf.

[107] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 3, pages 135–180. Elsevier, 2008. URL download/2007/BaHS07a.pdf.

[108] Dong M. Jiang Y. Zhang H Shi, Z. A logic foundation for the semantic web. In *Science in China, Series F 48(2)*, pages 161–178, 2005.

[109] Liang Chang, Fen Lin, and Zhongzhi Shi. A dynamic description logic for representation and reasoning about actions. In *Proceedings of the 2nd international conference on Knowledge science, engineering and management*, KSEM'07, pages 115–127, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-76718-5, 978-3-540-76718-3. URL http://dl.acm.org/citation.cfm?id=1775431.1775448.

[110] Guohua Shen, Zhiqiu Huang, Xiaodong Zhu, and Jun Yang. Reasoning about web services with dynamic description logics. In Mark Burgin, Masud H. Chowdhury, Chan H. Ham, Simone A. Ludwig, Weilian Su, and Sumanth Yenduri, editors, *CSIE (6)*, pages 106–110. IEEE Computer Society, 2009. ISBN 978-0-7695-3507-4. URL http://doi.ieeecomputersociety.org/10.1109/CSIE.2009.232.

[111] Wei Liu Yu Yue Du Bao Qi Guo Chun Yan Qiang Xu. A fast algorithm for web service composition based on dynamic description logic. In *Information Technology Journal, 9*, 1150-1157. Asian Network for Scientific Information, 2011. doi: 10.3923/itj.2010.1150.1157. URL http://scialert.net/abstract/?doi=itj.2010.1150.1157.

[112] Zhixiong Jiang, Leqiu Qian, Xin Pen, and Shisheng Zhu. Dynamic description logic for describing semantic web services. In *Computer and Computational Sciences, 2007. IMSCCS 2007. Second International Multi-Symposiums on*, pages 212 –219, aug. 2007. doi: 10.1109/IMSCCS.2007.16.

[113] Wenjia Niu, Zhongzhi Shi, Changlin Wan, Liang Chang, and Hui Peng. A ddl-based model for web service composition in context-aware environment. In *ICWS*, pages 787–788, 2008.

[114] Liangzhao Zeng, Anne H. Ngu, Boualem Benatallah, Rodion Podorozhny, and Hui Lei. Dynamic composition and optimization of web services. *Distrib. Parallel Databases*, 24(1-3):45–72, December 2008. ISSN 0926-8782. doi: 10.1007/s10619-008-7030-7. URL http://dx.doi.org/10.1007/s10619-008-7030-7.

[115] Onur Aydin, Nihan Kesim Cicekli, and Ilyas Cicekli. Automated web services composition with the event calculus. In *Engineering Societies in the Agents World VIII, 8th International Workshop, ESAW 2007, Athens, Greece, October 22-24, 2007, Revised Selected Papers*, pages 142–157, 2007. doi: 10.1007/978-3-540-87654-0_7. URL http://dx.doi.org/10.1007/978-3-540-87654-0_7.

[116] Murray Shanahan. Artificial intelligence today. chapter The Event Calculus Explained, pages 409–430. Springer-Verlag, Berlin, Heidelberg, 1999. ISBN 3-540-66428-9. URL http://dl.acm.org/citation.cfm?id=1805750.1805767.

[117] Vikas Agarwal, Girish Chafle, Koustuv Dasgupta, Neeran Karnik, Arun Kumar, Sumit Mittal, and Biplav Srivastava. Synthy: A system for end to end composition of web services. *Web Semant.*, 3(4):311–339, December 2005. ISSN 1570-8268. doi: 10.1016/j.websem.2005.09.002. URL http://dx.doi.org/10.1016/j.websem.2005.09.002.

[118] Ehtesham Zahoor, Olivier Perrin, and Claude Godart. An integrated declarative approach to web services composition and monitoring. In Gottfried Vossen, Darrell D. E. Long, and Jeffrey Xu Yu, editors, *WISE*, volume 5802 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2009. ISBN 978-3-642-04408-3. URL http://dx.doi.org/10.1007/978-3-642-04409-0_28.

[119] Holleis. P. Programming interactive physical prototypes. 1st Int'l Workshop on Design and Integration Principles for Smart Objects (DIPSO 07), 2007.

[120] Molood Makhlughian, Seyyed Mohsen Hashemi, Yousef Rastegari, and Emad Pejman. Web service selection based on ranking of qos using associative classification. *CoRR*, abs/1204.1425, 2012.

[121] Shirin Sohrabi and Sheila A. McIlraith. Optimizing web service composition while enforcing regulations. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, pages 601–617. Springer, 2009. ISBN 978-3-642-04929-3. URL http://dx.doi.org/10.1007/978-3-642-04930-9_38.

[122] Anupriya Ankolekar, Massimo Paolucci, and Katia Sycara. Towards a formal verification of owl-s process models. In *Proceedings of the 4th International Conference on The Semantic Web*, ISWC'05, pages 37–51, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3–540-29754-5, 978-3-540-29754-3. doi: 10.1007/11574620_6. URL http://dx.doi.org/10.1007/11574620_6.

[123] Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. Verifying a-priori the composition of declarative specified services. In Matteo Baldoni, Cristina Baroglio, Jamal Bentahar, Guido Boella, Massimo Cossentino, Mehdi Dastani, Barbara Dunin-Keplicz, Giancarlo Fortino, Marie-Peirre Gleizes, Joao Leite, Viviana Mascardi, Julian Padget, Juan Pavon, Axel Polleres, Amal El Fallah Seghrouchni, Paolo Torroni, and Rineke Verbrugge, editors, *2nd Federated Workshop on Multi-Agent Logics, Languages, and Organisations (MALLOW'009) - 2nd International Workshop on Agents, Web-Services and Ontologies: Integrated Methodologies*, volume 494. CEUR Electronic Workshop Proceedings, 2009. URL http://www.ceur-ws.org/Vol-494/mallowawesomepaper2.pdf.

[124] M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proceedings of the 2006 International Conference on Business Process Management Workshops*, BPM'06, pages 169–180, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-38444-8, 978-3-540-38444-1. doi: 10.1007/11837862_18. URL http://dx.doi.org/10.1007/11837862_18.

[125] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Trans. Comput. Log.*, 9(4), 2008. doi: 10.1145/1380572.1380578. URL http://doi.acm.org/10.1145/1380572.1380578.

[126] Marco Montali, Maja Pesic, Wil M. P. van der Aalst, Federico Chesani, Paola Mello, and Sergio Storari. Declarative specification and verification of service choreographiess. *ACM Trans. Web*, 4(1):3:1–3:62, January 2010. ISSN 1559-1131. doi: 10.1145/1658373.1658376. URL http://doi.acm.org/10.1145/1658373.1658376.

[127] Wil M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures, 16.07. - 21.07.2006*, 2006. URL http://drops.dagstuhl.de/opus/volltexte/2006/829.

[128] Ehtesham Zahoor, Kashif Munir, Olivier Perrin, and Claude Godart. A bounded model checking approach for the verification of web services composition. *Int. J. Web Service Res.*, 10(4):62–81, 2013. doi: 10.4018/ijwsr.2013100103. URL http://dx.doi.org/10.4018/ijwsr.2013100103.

[129] Ehtesham Zahoor, Olivier Perrin, and Claude Godart. Web services composition verification using satisfiability solving. In *2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, June 24-29, 2012*, pages 242–249, 2012. doi: 10.1109/ICWS.2012.75. URL http://dx.doi.org/10.1109/ICWS.2012.75.

[130] R Kowalski and M Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, January 1986. ISSN 0288-3635. doi: 10.1007/BF03037383. URL http://dx.doi.org/10.1007/BF03037383.

[131] Erik T. Mueller. Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, 14(5):703–730, 2004. doi: 10.1093/logcom/14.5.703. URL http://logcom.oxfordjournals.org/content/14/5/703.abstract.

[132] K. Taylor, P. Brebner, M. Kearney, D. Zhang, K. Lam, and V. Tosic. Towards declarative monitoring of declarative service compositions. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 315–322, April 2007. doi: 10.1109/ICDEW.2007.4401011.

[133] Vladimir Tosic, Wei Ma, Bernard Pagurek, and Babak Esfandiari. Web service offerings infrastructure (wsoi) - a management infrastructure for xml web services. In *NOMS (1)*, pages 817–830. IEEE, 2004. URL http://dx.doi.org/10.1109/NOMS.2004.1317770.

[134] Abdelkarim Erradi, Piyush Maheshwari, and Vladimir Tosic. Ws-policy based monitoring of composite web services. *Web Services, European Conference on*, 0: 99–108, 2007. doi: http://doi.ieeecomputersociety.org/10.1109/ECOWS.2007.31.

[135] L. Baresi and S. Guinea. Event-based multi-level service monitoring. In *Web Services (ICWS), 2013 IEEE 20th International Conference on*, pages 83–90, June 2013. doi: 10.1109/ICWS.2013.21.

[136] Marco Comuzzi and George Spanoudakis. A framework for hierarchical and recursive monitoring of service based systems. In *Fourth International Conference on Internet and Web Applications and Services, ICIW 2009, 24-28 May 2009, Venice/Mestre, Italy*, pages 383–388, 2009. doi: 10.1109/ICIW.2009.63. URL http://dx.doi.org/10.1109/ICIW.2009.63.

[137] D. Skogan, R. Groenmo, and I Solheim. Web service composition in uml. In *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*, pages 47–57, Sept 2004. doi: 10.1109/EDOC.2004.1342504.

[138] R. Groenmo and M.C. Jaeger. Model-driven semantic web service composition. In *Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific*, pages 8 pp.–, Dec 2005. doi: 10.1109/APSEC.2005.81.

[139] Luokai Hu, Shi Ying, Kai Zhao, and Rui Chen. A semantic web service description language. In *Proceedings of the 2009 WASE International Conference on Information Engineering - Volume 02*, ICIE '09, pages 449–452, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3679-8. doi: 10.1109/ICIE.2009.205. URL http://dx.doi.org/10.1109/ICIE.2009.205.

[140] Wassim Derguech and Sami Bhiri. Business process model overview: Determining the capability of a process model using ontologies. In *BIS*, pages 62–74, 2013.

[141] Joachim Peer. A pop-based replanning agent for automatic web service composition. In *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, pages 47–61, 2005. doi: 10.1007/11431053_4. URL http://dx.doi.org/10.1007/11431053_4.

[142] Wenbin Li, Youakim Badr, and Frederique Biennier. Towards A Capability Model for Web Service Composition. In *ICWS 2013*, pages 609–610, June 2013. URL http://liris.cnrs.fr/publis/?id=6183.

[143] Wassim Derguech, Souleiman Hasan, Sami Bhiri, and Edward Curry. Organizing Capabilities using Formal Concept Analysis. In *22th IEEE International Conference on Enabling Technologies: Infrastructures for Collaborative Enterprises*, Hammamet, Tunisia, 2013. URL http://www.edwardcurry.org/publications/WETICE_2013.pdf.

[144] Wassim Derguech and Sami Bhiri. Capability modelling - case of logistics capabilities. In Marcello La Rosa and Pnina Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 519–529. Springer, 2012. ISBN 978-3-642-36284-2. URL http://dx.doi.org/10.1007/978-3-642-36285-9_53.

[145] Gianpaolo Cugola, Leandro Sales Pinto, and Giordano Tamburrelli. Qos-aware adaptive service orchestrations. In *2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, June 24-29, 2012*, pages 440–447, 2012. doi: 10.1109/ICWS.2012.104. URL http://dx.doi.org/10.1109/ICWS.2012.104.

[146] Vikas Agarwal, Girish Chafle, Koustuv Dasgupta, Neeran Karnik, Arun Kumar, Sumit Mittal, and Biplav Srivastava. Synthy: A system for end to end composition of web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4), 2011. ISSN 1570-8268. URL http://www.websemanticsjournal.org/index.php/ps/article/view/79.

[147] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, may 2001. URL http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[148] Ruben Verborgh. *Serendipitous Web Applications through Semantic Hypermedia*. PhD thesis, Ghent University, Ghent, Belgium, February 2014. URL http://ruben.verborgh.org/phd/ruben-verborgh-phd.pdf.

[149] Amir Yousefli, A. Deheshvar, and T. Komijani. Ranking vague sets using topsis method. *Journal of Intelligent and Fuzzy Systems*, 25(4):853–858, 2013. URL http://dx.doi.org/10.3233/IFS-120663.

[150] Yong-Bin Li and Jian-Ping Zhang. Topsis method for hybrid multiple attribute decision making with 2-tuple linguistic information and its application to computer network security evaluation. *Journal of Intelligent and Fuzzy Systems*, 26 (3):1563–1569, 2014. doi: 10.3233/IFS-130876. URL http://dx.doi.org/10.3233/IFS-130876.

[151] Ding-Yuan Cheng, Kuo-Ming Chao, Chi-Chun Lo, and Chen-Fang Tsai. A user centric service-oriented modeling approach. *World Wide Web*, 14(4):431–459, 2011. doi: 10.1007/s11280-011-0115-7. URL http://dx.doi.org/10.1007/s11280-011-0115-7.

[152] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, May 2004. ISSN 0098-5589. doi: 10.1109/TSE.2004.11.

[153] Frederico Alvares De Oliveira and José M. Parente de Oliveira. Qos-based approach for dynamic web service composition. *j-jucs*, 17(5):712–741, mar 2011.

[154] Ruben Verborgh, Thomas Steiner, Erik Mannens, Rik Van de Walle, and Joaquim Gabarró Vallés. Proof-based automated Web API composition and integration. In *Proceedings of the International Conference on Advanced IT, Engineering and Management*, pages 181–182, February 2013.

[155] Ruben Verborgh, Vincent Haerinck, Thomas Steiner, Davy Van Deursen, Sofie Van Hoecke, Jos De Roo, Rik Van de Walle, and Joaquim Gabarró. Functional composition of sensor web apis. In *Proceedings of the 5th International Workshop on Semantic Sensor Networks, SSN12, Boston, Massachusetts, USA, November 12, 2012*, pages 65–80, 2012. URL http://ceur-ws.org/Vol-904/paper6.pdf.

[156] Tim Berners-Lee. Notation 3 logic, August 2005. URL http://www.w3.org/DesignIssues/Notation3.

[157] Tim Berners-Lee and Dan Connolly. Notation3 (n3): A readable rdf syntax, March 2011. URL http://www.w3.org/TeamSubmission/n3/.

[158] Tim Berners-Lee. Cwm, 2000–2009. URL http://www.w3.org/2000/10/swap/doc/cwm.html.

[159] James L. Rash, Christopher Rouff, Walt Truszkowski, Diana F. Gordon, and Michael G. Hinchey, editors. *Formal Approaches to Agent-Based Systems, First International Workshop, FAABS 2000 Greenbelt, MD, USA, April 5-7, 2000, Revised Papers*, volume 1871 of *Lecture Notes in Computer Science*, 2001. Springer. ISBN 3-540-42716-3.

[160] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53, June 2007. ISSN 1570-8268. doi: 10.1016/j.websem.2007.03.004. URL http://dx.doi.org/10.1016/j.websem.2007.03.004.

[161] Georgios Meditskos and Nick Bassiliades. Dlejena: A practical forward-chaining owl 2 rl reasoner combining jena and pellet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(1), 2010. ISSN 1570-8268. URL http://www.websemanticsjournal.org/index.php/ps/article/view/176.

[162] Tim Berners-lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3logic: A logical framework for the world wide web. *Theory Pract. Log. Program.*, 8(3):249–269, May 2008. ISSN 1471-0684. doi: 10.1017/S1471068407003213. URL http://dx.doi.org/10.1017/S1471068407003213.

[163] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001. URL http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[164] Sofie Van Hoecke Sam Coppens Jos De Roo Thomas Steiner ErikMannens Ruben Verborgh, Dörthe Arndt and Rik Van deWalle. The pragmatic proof: Hypermedia api composition and execution. *Theory and Practice of Logic Programming*, 2014.

[165] Virtuoso universal server. http://virtuoso.openlinksw.com.

[166] C. Garcia-martinez, O. Cordon, and F. Herrera. An empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp. In *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence*, pages 61–72. Springer, 2004.

[167] Carlos García-Martínez, Oscar Cordón, and Francisco Herrera. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. *European Journal of Operational Research*, 180(1):116–148, 2007. doi: 10.1016/j.ejor.2006.03.041. URL http://dx.doi.org/10.1016/j.ejor.2006.03.041.

[168] Joonho Kwon, Hyeonji Kim, Daewook Lee, and Sukho Lee. Redundant-free web services composition based on a two-phase algorithm. In *ICWS*, pages 361–368, 2008.

[169] Srividya Kona, Ajay Bansal, and Gopal Gupta. Automatic composition of semanticweb services. In *ICWS*, pages 150–158, 2007.

[170] Ajay Bansal, Srividya Kona, M. Brian Blake, and Gopal Gupta. An agent-based approach for composition of semantic web services. In *WETICE*, pages 12–17, 2008.

[171] Djamal Benslimane, Schahram Dustdar, and Amit P. Sheth. Services mashups: The new generation of web applications. *IEEE Internet Computing*, 12(5):13–15, 2008. doi: 10.1109/MIC.2008.110. URL http://doi.ieeecomputersociety.org/10.1109/MIC.2008.110.

[172] Karim Benouaret, Djamal Benslimane, and Allel HadjAli. On the use of fuzzy dominance for computing service skyline based on qos. In *IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, July 4-9, 2011*, pages 540–547, 2011. doi: 10.1109/ICWS.2011.93. URL http://doi.ieeecomputersociety.org/10.1109/ICWS.2011.93.

[173] Bipin Upadhyaya, Ying Zou, Iman Keivanloo, and Joanna W. Ng. Quality of experience: What end-users say about web services? In *2014 IEEE International Conference on Web Services, ICWS, 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 57–64, 2014. doi: 10.1109/ICWS.2014.21. URL http://dx.doi.org/10.1109/ICWS.2014.21.

[174] Bipin Upadhyaya, Ying Zou, Joanna W. Ng, Tinny Ng, and Diana H. Lau. Towards quality of experience driven service composition. In *2014 IEEE World Congress on Services, SERVICES 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 18–20, 2014. doi: 10.1109/SERVICES.2014.13. URL http://dx.doi.org/10.1109/SERVICES.2014.13.

[175] Shaohua Wang, Bipin Upadhyaya, Ying Zou, Iman Keivanloo, and Joanna W. Ng. Automatic propagation of user inputs in service composition for end-users. In *2014 IEEE International Conference on Web Services, ICWS, 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 73–80, 2014. doi: 10.1109/ICWS.2014.23. URL http://dx.doi.org/10.1109/ICWS.2014.23.

[176] Giovanni Acampora, Matteo Gaeta, Vincenzo Loia, and Athanasios V. Vasilakos. Interoperable and adaptive fuzzy services for ambient intelligence applications. *ACM Trans. Auton. Adapt. Syst.*, 5(2):8:1–8:26, May 2010. ISSN 1556-4665. doi: 10.1145/1740600.1740604. URL http://doi.acm.org/10.1145/1740600.1740604.

[177] J. Bronsted, K.M. Hansen, and M. Ingstrup. Service composition issues in pervasive computing. *Pervasive Computing, IEEE*, 9(1):62–70, Jan 2010. ISSN 1536-1268. doi: 10.1109/MPRV.2010.11.

[178] M. Reza Rahimi, Nalini Venkatasubramanian, Sharad Mehrotra, and Athanasios V. Vasilakos. Mapcloud: Mobile applications on an elastic and scalable 2-tier cloud architecture. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, pages 83–90, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4862-3. doi: 10.1109/UCC.2012.25. URL http://dx.doi.org/10.1109/UCC.2012.25.

[179] Quan Z. Sheng, Jian Yu, and Schahram Dustdar. *Enabling Context-Aware Web Services: Methods, Architectures, and Technologies*. Chapman & Hall/CRC, 1st edition, 2010. ISBN 1439809852, 9781439809853.