PSL UNIVERSITÉ PARIS

HABILITATION À DIRIGER DES RECHERCHES

DE L'UNIVERSITÉ PSL

Présentée à l'Université Paris-Dauphine

Service-oriented Systems Quality Matter: Contributions to Efficiency, Reliability and Trustworthiness

Présentation des travaux par

Joyce EL HADDAD

Le 16 décembre 2025

Discipline

Informatique

Composition du jury :

Cristina, BAZGAN PU, Université Paris Dauphine-PSL	*Présidente*
Salima, BENBERNOU PU, Université Paris Cité	*Rapporteure*
Schahram, DUSTDAR PU, TU Wien	*Rapporteur*
Nikolaos, GEORGANTAS CR, INRIA Paris	*Rapporteur*
Djamal, BENSLIMANE PU, Université Claude Bernard Lyon 1	*Examinateur*
Daniela, GRIGORI PU, Université Paris Dauphine-PSL	*Coordinatrice*

Dauphine | PSL UNIVERSITÉ PARIS

Acknowledgments

It has been a long journey to reach this point.

First of all, I would like to thank the committee members for agreeing to serve on my defence jury: Salima Benbernou, Schahram Dustdar, and Nikolaos Georgantas for reviewing the manuscript and for their kind reports; Djamal Benslimane who very quickly agreed to act as external examiner; Cristina Bazgan for chairing the committee and for our numerous discussions; and Daniela Grigori for serving as coordinator of the habilitation and for placing her trust in me. It was a great honor to have you all as members of the jury.

The research work presented in this manuscript is the result of teamwork that would not have been possible without the fruitful collaboration with many researchers from national and international institutions, as well as several doctoral and master's students. My sincere thanks go to (in chronological order) Serge Haddad, Maude Manouvrier, Marta Rukoz, Yudith Cardinale, Maria-Esther Vidal, Eduardo Blanco, Guillermo Ramirez, Faisal Abu-Khzam, Cristina Bazgan, Florian Sikora, Olivier Spanjaard, Lidia Fuentes, Nadia Gamez, Suzanne Pinson, Noura Faci, Zakaria Maamar, Daniela Grigori, Thomas Degueule, and Pascal Poizat. I am deeply grateful to all of you, I have learned a lot from each one of you. A big thank you also to all the Ph.D. students I have had the pleasure to work with over the years, Amine Louati, Wissam Gherissi, and Damien Jaime.

I would also like to express my sincere appreciation to all my colleagues at LAMSADE research center. A special thanks goes to Jérôme Lang, with whom I work closely since 2024 as deputy director of LAMSADE. I am truly grateful for our discussions and for his advice. To all my other colleagues, whom I have not mentioned for fear of forgetting some, I would like to thank you for creating such a pleasant atmosphere that makes our center such a nice place to work.

Last but not least, I would like to thank my family for their constant support. A huge thank you especially to my husband and to my son for their support throughout this long journey. Thank you both for your encouragement, which I will never forget, especially during our indoor climbing session.

Contents

CHAPTER 1

INTRODUCTION

This habilitation thesis manuscript summarizes my research work, which deals with efficiency, reliability, and trustworthiness in service-oriented systems. More specifically, I am interested in the optimisation of non-functional properties in service compositions and their enhancement with social aspects. In this context, I have developed approaches based on models derived from combinatorial optimization and graph theory, focusing mainly on discovering, selecting, composing, and executing services while ensuring quality optimization, failure recovery, and trustworthiness. My aim in this manuscript is not to provide an exhaustive account of all my research activities in this area, but rather to give an overview of my scientific approach when tackling a new research challenge.

1.1 Context

One of the key challenges facing organisations is the need to accelerate time-to-market while meeting user needs. Service orientation is the most appropriate software development approach for addressing this challenge, as it enables the creation of flexible architectures that can quickly adapt to user requirements.

Service-oriented Systems : Architecture and Computing

Service-Oriented Architecture (SOA) is a specific architectural implementation of service orientation that emphasises the creation and use of *services* as fundamental building blocks for developing software systems (Erl [84]). It encompasses a set of principles describing the foundations for fast development of software systems through the interconnection of heterogeneous and distributed services in order to achieve specific business objectives (Papazoglou et al. [189, 190]). Services are autonomous, platform-independent, computational units that can be described, published, discovered, accessed and used independently via standard protocols (Stojanovic et al. [212]). Although SOA is not tied to a specific technology, it has been commonly implemented using Web services and, more recently, Microservices. As defined by the World Wide Web Consortium, a Web service is a software system designed to support interoperable machine-to-machine interaction over a network (Booth et al. [36]). Each Web service is identified by a URI and exposes its functionality over the Internet using standard XML-based languages and protocols. It has an interface described using Web Services Description Language (Chinnici et al. [64]), and published in a UDDI (Universal Description, Discovery, and Integration) registry or repository (Bellwood et al. [23]).

The principles of Service-Oriented Computing (SOC) paradigm, which organises and structures software systems based on the notion of services, are integrated into the SOA architectural style. In SOC, software components are designed to provide services to other components within a network, promoting a modular, loosely coupled, and interoperable architecture. Today, this paradigm is widely employed in the development of distributed systems. It abstracts the internal details of services, exposing only their interface to consumers, as we will describe below.

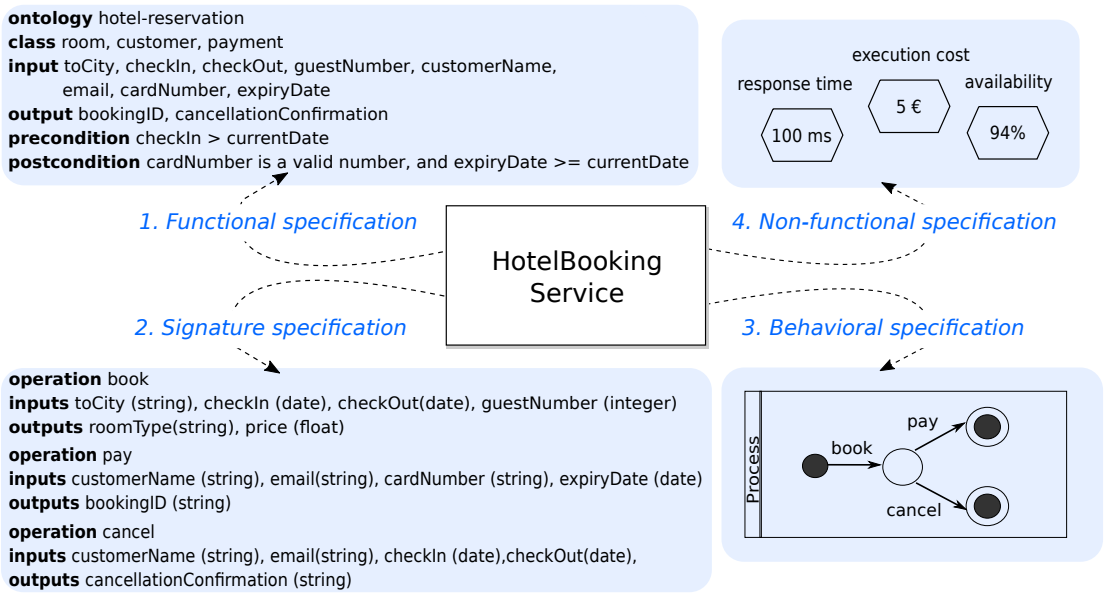


Figure 1.1: An Example of a HotelBooking Service Specifications

Service Specifications

As services are intended to be discovered and used by other applications or services, they need to be described and understood in terms of functional capabilities as well as behavioural specifications and non-functional requirements. This is known as the service *interface*, which comprises four description specifications (Beugnard et al. [29], Canal et al. [43]):

- *Functional specification*: this includes the formal description of the *capability* of the service, *i.e.*, what the service can do. It also specifies the conditions on service inputs (pre-conditions) and outputs (post-conditions) that must be satisfied before and after the service execution, respectively. In the field of Web services, this level of description is related to the Semantic Web, using ontologies and languages such as WSMO (De Bruijn et al. [69]), OWL-S (Martin et al. [165]) or SAWSDL (Farrell and Lausen [87]). Ontologies are used to semantically define inputs and outputs data terms and their relationships within the service domain. This corresponds to the description provided for the HotelBooking service shown in Figure 1.1.

- *Signature specification*: this describes the actions the service can perform. It refers to the public interface of the service, as in the public interface of a Java class. It defines the name of operations offered by the service, as well as their required inputs and returned outputs. As for example, the HotelBooking service in Figure 1.1 allows users to search and book for rooms (book operation), and to manage their bookings by either purchasing their room reservation (pay operation) or cancelling their reservation (cancel operation).

- *Behavioral specification*: this describes how the functionality of a service can be achieved, both in terms of the internal process it follows (*i.e.*, the state transformation performed by the service) and in terms of its interactions with other services. Returning to the HotelBooking service example, the service requires users to invoke its operations in a specific order. Here, both pay and cancel operations require input data provided by book operation. Consequently, book operation must be executed first, followed by either the pay or cancel operation. This behavior cannot be deduced solely from the signature specification and therefore must be explicitly defined. Several proposals have addressed this issue by extending service interfaces with behavioral descriptions (Bertoli et al. [28], Grigori et al. [106], Ben Mokhtar et al. [176]).

- *Non-functional specification*: this refers to the qualities and constraints that describe how a service performs, rather than what it does. Many non-functional properties, such as responsiveness of the

service measured by estimating its maximum response time, are important to be taken into account before using a service. For instance, in the case of our Hotelbooking service, invoking its functionality (booking a room) may be constrained by temporal availability, specifying when the service can be invoked. Such constraints represent non-functional properties and are addressed at the non-functional specification level of the service interface. This description level is relevant at both the architectural level (Losavio et al. [151], Singh et al. [207]) and the component service level (Oriol et al. [184], Zeng et al. [257]). Non-functional properties at the component service level can be defined with varying degrees of granularity: coarse-grained at the service level (*i.e.*, viewing the service is as a black box and associating non-functional properties to the entire service), mid-grained at the path level (*i.e.*, associating non-functional properties with specific operation sequences, such as the book-pay in Hotelbooking service), and fine-grained at the transition level (*i.e.*, assigning non-functional properties to each individual transition within the service's behavioral specification, where each transition corresponds to a specific capability).

Most work on service-oriented systems associate interfaces with only a sub-part of these four specification levels. In my research, my primary focus has been on the non-functional specification level.

Non-functional Specification

Quality models are fundamental to the evaluation of a system quality. They serve as essential tools for specifying non-functional properties and requirements, establishing measures and performing quality evaluations (Oriol et al. [184]). As defined by ISO/IEC 25000 series of standards on system and software quality models, a quality model is a hierarchical set of quality characteristics and their relationships. A quality (*sub-*)*characteristic* is a category of quality attributes representing inherent properties of an entity, such as software, services, or processes within a system. However, since quality (sub-)characteristics do not provide direct measurements, they must to be further decomposed into more specific, fine-grained concepts, referred to as quality *attributes* or *criteria*. When these attributes are clearly defined in measurable terms, they are referred to as quality *metrics* or *indicators*, which enable effective evaluation and monitoring of system quality.

Example 1.1.1. An example of quality characteristics is *Performance*. It can be decomposed into sub-characteristics such as *time behavior*, *capacity*, and *resource utilization*. Time behavior, for instance, can be further decomposed into quality attributes like *response time*, *latency* or *throughput*. An example of quality metric is latency that could be measured as the *time delay* between sending a request to the service and receiving its response.

Quality attributes can be classified into several categories depending on various factors, including domain dependency (*domain-dependent* vs. *domain-independent*), granularity (*atomic* vs. *composite*), measurement type (*quantitative* vs. *qualitative*), and measurement periodicity (*static* vs. *dynamic*). A comprehensive survey on service quality description can be found in Kritikos et al. [132].

- Domain dependency: the domain dependency of quality attributes refers to whether an attribute is specific to a particular application domain (*i.e.*, domain-dependent) or applicable across multiple domains (*i.e.*, domain-independent). *Domain-dependent attributes* are attributes specific to services of a particular domain. *Domain-independent attributes* are attributes that apply to all services, regardless of application domain.

Example 1.1.2. An example of domain-independent attribute is *availability* that represents the expected percentage of time the service is operational and accessible, making it applicable to all services, regardless of the domain. In contrast, *certification level* of data sources is a domain-dependent attribute specific to stock market quotes services.

- Granularity: *Atomic attributes* are quality attributes that cannot be further decomposed into smaller measurable components. They are self-contained and independent in their evaluation. *Composite attributes* are derived from multiple atomic attributes and are computed based on their combined values. They depend on other attributes for their assessment.

Example 1.1.3. An example of composite attribute could be *throughput* since it can be computed by evaluating *network bandwidth* and *processing speed*. In contrast, *execution time* is atomic as it does not rely on any other attribute.

- Measurement type: quality metrics or indicators are measures that provide an estimation or evaluation of specified quality attributes. Their values can be either *quantitative* (*numerical*) or *qualitative* (*categorical*), depending on whether the attribute can be measured objectively or is assessed through subjective means (Agresti [5]):
  - *Quantitative attributes.* These are measurable attributes expressed numerically, offering precise data that can be statistically analyzed or compared. Quantitative attributes are represented by integer or real values and can be further divided into discrete or continuous types. Discrete attributes take on specific values, while continuous attributes can take any value within a range. Special types of these attributes include interval and ratio-scaled attributes.
  - *Qualitative attributes.* These attributes are assessed through observation, user feedback, and expert judgment rather than through direct measurement. They can be classified as nominal, ordinal, or binary. Nominal attributes represent different categories or labels without any inherent order (*e.g.*, types of errors). Ordinal attributes have a meaningful order or ranking (*e.g.*, user satisfaction levels), while binary attributes can take only two distinct values (*e.g.*, yes/no or true/false). Binary attributes can be nominal or ordinal, depending on whether they have inherent order.

Example 1.1.4. An example of quantitative attribute is *execution cost* that represent the monetary charge for using a service either per request or per time period, *e.g.*, 0.5 cents per request. In contrast, *user satisfaction* is a qualitative attribute representing a subjective evaluation of the user experience based on service performance, taking a value in the set $\{Excellent, Average, Poor\}$.

- Measurement periodicity: *Static attributes* remain constant over time and do not change based on system state or usage. They are typically defined at design time and do not require continuous measurement. *Dynamic attributes* vary over time based on system performance, external conditions, or user interactions. They require periodic or real-time evaluation.

Example 1.1.5. An example of static attribute is *security* as it does not change over time, while *availability* is dynamic since it fluctuates over time and is computed according to a schedule.

To formally describe quality (sub-)characteristics, three types of quality documents are essential throughout the service lifecycle (Kritikos et al. in [132]): *Service Quality Model* used to describe concrete quality attributes, *Quality-based Service Description* which defines quality capabilities and requirements as a set of constraints on quality attributes and metrics, and *Service Level Agreement* (SLA) which specifies the contractual quality commitments between service providers and consumers, detailing service levels and penalties for non-compliance.

In web services field, when it comes to quality dimension, the concept of Quality-of-Service (QoS) has emerged as a key term in numerous research studies to express non-functional specifications and requirements (Jaeger et al.[122], Menasce [171], O'Sullivan et al. [185], Ouzzani and Bouguettaya [186]). Although QoS has been extensively studied across multiple domains, including networking (Chen et al.[62],

Cruz [66]) and distributed systems (Hutchison et al.[116], Le et al. [135]), its role in service-oriented computing is particularly significant. Based on the QoS performance of Web services, various approaches have been proposed for Web service selection (Yu and Bouguettaya [250], Zemni et al. [255], Zheng et al. [265]), Web service composition (Ben Mabrouk et al. [24], Zeng et al. [257]), fault-tolerant Web services (Fang et al.[86], Zheng and Lyu [266]), Web service recommendation (Zheng et al. [267], Zheng et al. [268]), and for Web service reliability prediction (Cardoso et al. [56], Grassi and Patella [102]).

In the remainder of this manuscript, I will use the terms attribute, criterion, indicator, or metric as synonyms. I will also use the term Quality of Service (QoS) to refer to the set of static, domain-independent, and quantitative attributes provided by a service, such as response time or execution cost. My work integrates a range of attributes, including composite ones such as trust, and qualitative ones such as transactional properties, in order to design efficient, resilient and trustworthy composite services.

Service Composition

Building complex software systems by integrating multiple services is a fundamental aspect of service-oriented computing, known as *service composition* process. It enables to integrate independent and reusable services to achieve specific business goals, such as booking a trip or processing an online order. Service composition has been a major research domain that has evolved significantly over the years (Bouguettaya et al. [37], Dustdar and Schreiner [76], Papazoglou et al. [190]). It has been often restricted to Web services (Alonso et al. [9]). However, the composition problem extends far beyond Web services and has been a long-standing challenge in software engineering as for example in Object-Oriented Programming (Wegner [236]), in Component-Based Programming (Szyperski et al. [217]) and more recently in Cloud Computing (Ranjan et al. [196], Wang et al. [231]) and Internet-of-Things (Gubbi et al. [108], Truong et al. [222]). The common goal across these domains is to reuse existing software entities (*i.e.*, objects, components, or services) as black boxes and compose them to build more complex applications.

To better understand service composition, we explore below its key steps:

- **Service Description and Discovery.** Service description plays a crucial role in defining, advertising, and facilitating the discovery of services. In the context of Web services, several standards exist for service description including WSDL which defines SOAP-based services by detailing available functions, input/output parameters, and bindings specifications. Beyond syntactic descriptions, semantic technologies enhance service description by providing rich semantic meaning, enabling more accurate service discovery (McIlraith et al., [168]). Once services are described, they must be discovered based on user requirements. Service discovery involves identifying available services that meet the required functionalities. In service-oriented computing, discovery approaches generally fall into two main categories: Keyword-based discovery that searches for services based on keywords in WSDL descriptions (Cheng et al. [63], He et al. [115]), and Ontology-based semantic discovery that uses semantic reasoning to identify services with similar functionalities (Benatallah et al. [25], Sycara et al. [215]). Some others approaches also take into account the service behavior specification during the discovery process (Grigori et al. [105], Shen and Su [204]).

- **Service Selection and Composition.** After discovering multiple functionally equivalent candidate services, the next step is to select the most appropriate ones based on non-functional requirements. Service selection can occur either at design-time or run-time (Hwang et al. [118]). With design-time selection, services are chosen before execution, without considering dynamic changes in non-functional attributes (Eid et al. [77]). While in run-time selection, services are selected during execution, allowing for adaptation to changes in non-functional attributes (Atrey et al. [19], Stantchev and Schropfer [211]). In both cases, the selection step consists of identifying the best candidate services or compositions based on non-functional criteria. The criterion may be single such as the fastest execution time or the less expensive, or multiple, which requires the use of appropriate Multiple Criteria Decision Making techniques to aggregate criteria effectively. In service-oriented computing, various QoS-based apporaches have been proposed for service selection (Bonatti and Festa [35], Cardellini et al. [46], Yu et al. [253], Zheng et al. [265]) and for service composition (Alrifai et Risse [10], Ardagna et Pernici [17], Cardellini et al. [45], Cardellini et al. [46], Zeng et al. [257]).

- **Service Execution and Recovery.** Once services or compositions are selected at design-time, the next step is their execution. During this phase, challenges beyond service invocation must be addressed, particularly failure recovery to ensure correct and reliable execution. During the execution of a composite service, various situations can cause failures in its component services. The failure of a single service may result in the failure of the entire composite service execution. Many failure recovery approaches have been proposed to manage and recover from failures during composite service execution (Gao et al. [94]). Some of the proposed approaches use forward recovery that attempts to complete the execution of the composite service by retrying or replacing the failed component (Dolog et al. [71], Issarny et al. [120], Yan et al. [243]). Other approaches use backward recovery to restore the system to a previously correct state, typically implemented using rollback, compensation, or checkpointing mechanisms (Angarita et al. [14], Dialani et al. [70], Mansour and Dillon [164], Marzouk et al. [166], Xu et al. [241]).

Service composition is a cornerstone of service-oriented computing, and its effectiveness relies on overcoming several challenges including accurate service discovery, efficient and trustworthy selection and composition, reliable execution, and supporting failure recovery mechanisms. In this manuscript, the presented work focuses specifically on these critical aspects of the service composition process. Each challenge is addressed using combinatorial optimization and graph theory, with the primary objective of optimizing non-functional properties.

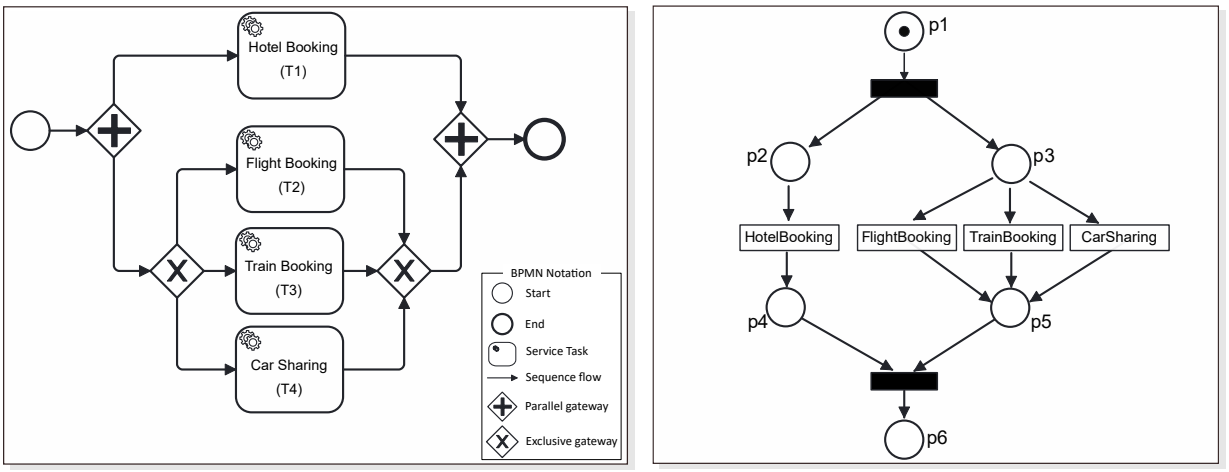
Service Composition Representation

To model and describe service compositions, a variety of languages and graphical representations are commonly used (Paik et al. [187]). These representations help in understanding the structure, flow, and interactions within a composite service. Among them, *Diagram-based representations* are the most widely used. They consist of symbols that represent component services of a composition and connectors that indicate control flow, data flow, or other relationships between these components (Lemos et al. [137]). In the context of service-oriented computing, diagram-based representations can generally be categorized into two main types:

- *Process-based representation*: it models the structure of a service composition (*i.e.*, its service components and their relationships) using workflow models. These define control flow constructs (sequential, parallel, and conditional), along with data flow and communication between service components. A workflow specifies an abstract composition in which each task can later be instantiated with a concrete component service. Once all tasks are mapped to specific services, the workflow yields a concrete composite service. Multiple composite services may result from the same workflow, depending on service assignments. Common languages for modeling service compositions in this paradigm include Business Process Model and Notation (BPMN) (Object Management Group [107]), Business Process Execution Language (BPEL) (Andrews et al. [13]), Yet Another Workflow Language (YAWL) (Van Der Aalst and Ter Hofstede [227]), and state charts (Zeng et al. [256]).

- *Graph-based representation*: it captures the logical structure including concurrency and dependencies between service components, as well as the flow of execution. Common models include Petri Nets (Hamadi and Benatallah [110]) and dependency graphs (Liang et al. [143], Zheng and Yan [264]). A Petri net is a bipartite graph where nodes are either places or transitions. Places (depicted as circles) represent states. Transitions (depicted as squares) represent service invocations or internal steps for modeling control flow constructs. Places and transitions are connected by directed arcs. A Petri net may have an associated marking, which assigns tokens to places, representing data of a composition. Dependency graphs, on the other hand, represent input-output dependencies between services by using a directed graph where services are modeled as nodes and their dependencies as edges.

To illustrate service composition representation, consider the following motivating scenario:



(a) BPMN representation (b) Petri net representation

Figure 1.2: TravelArrangement composition representation

Motivating example. A computer science researcher working in Paris has had her paper accepted for presentation at an international conference in Turin, Italy. To attend the conference, she must complete several required tasks before traveling. This include registration, travel arrangements, and validation tasks. For travel arrangements, she is required to use the digital travel management platform of her university, referred to as TravelArrangement, which offers both housing and transportation services. Figure 1.2(a) illustrates process-based representation of the TravelArrangement composite service using BPMN, while Figure 1.2(b) illustrates its graph-based representation using a Petri net.

In this manuscript, some of our approaches model service compositions with process-based representations, while for others we employ graph-based representations.

Exact vs. Heuristics Optimization Approaches

Combinatorial optimization involves searching for an optimal solution within a finite set of solutions (Schrijver [201]). It is a fundamental concept in computational sciences and many other fields, aiming to find the best possible solution to a given problem, often under constraints. Exact methods systematically explore the entire solution space or apply mathematical techniques to ensure *optimal* solutions. While these methods work well for small-scale problems, they often become computationally expensive for large instances. In contrast, heuristic methods are widely used to efficiently find *near-optimal* (*i.e.*, good high-quality) solutions, even though they do not guarantee global optimality. These methods are particularly useful for large-scale problems where exact methods become impractical due to their high computational cost.

In service-oriented computing, a decision problem arises concerning the selection of the best component services or compositions to meet user functional and non-functional requirements, called *QoS-aware service composition* problem. In the literature, various approaches formulate this problem as a combinatorial optimization one and propose *exact* or *heuristic* methods to resolve it. The first exact solutions were proposed by Zeng et al. [257], and Bonatti and Festa [35]. Since then, researchers introduced various approaches navigating from exact solutions to near-optimal heuristics and metaheuristics aiming to improve quality while maintaining efficiency, as reviewed by Jatoth et al. [126] systematic literature review. The quest for finding high-quality near-optimal solutions has led recent approaches to increasingly adopt hybrid methods (Gavvala et al. [95], Mistry et al. [174]). While promising, such approaches lie beyond the scope of this manuscript.

1.2 Research Challenges

To illustrate the research challenges we have addressed, back again to the motivating scenario.

Motivating example. As depicted in Figure 1.2(a), for travel arrangements, the researcher is required to use TravelArrangement, the digital travel management platform of her university, which offers both housing and transportation services. For accommodation, she can use HotelBooking to book a room. For transportation, she can make a reservation using either FlightBooking, TrainBooking, or CarSharing. This scenario involves two types of control flow structures: parallel and conditional. For each researcher query, there are three possible composition plans to accomplish her travel arrangements: (T1,T2), (T1,T3), or (T1,T4).

As shown in Table 1.1, each task is associated with a set of available candidate component services offering the same functionality but differing in their Quality-of-Service (QoS) and transactional properties. This variation arises from the presence of multiple service providers competing to deliver similar services. For instance, when booking a hotel room, our researcher can choose between two available services: s_{11} provided by HB1 with a response time of $15mn$ and a cost of 1€; or s_{12} offered by HB2 with a response time of $10mn$, a cost of 3€, and the additional advantage of compensation.

Table 1.1: Available candidate component services for TravelArrangement

Task	Service	Response Time	Execution Cost	Transactional Property	Provider
HotelBooking	s_{11}	$15mn$	1€	–	HB1
(T1)	s_{12}	$10mn$	3€	*compensatable*	HB2
FlightBooking	s_{21}	$20mn$	1€	–	FB1
(T2)	s_{22}	$10mn$	10€	*compensatable*	FB2
	s_{23}	$8mn$	5€	*retriable*	FB3
TrainBooking	s_{31}	$20mn$	5€	–	TB1
(T3)	s_{32}	$18mn$	8€	–	TB2
CarSharing	s_{41}	$5mn$	2€	*compensatable*	CS1
(T4)	s_{42}	$12mn$	15€	–	CS2

When multiple providers offer functionally equivalent services, it becomes essential to differentiate between candidate services, particularly when only one service is needed to fulfill a task. A common strategy involves selecting the most suitable service based on non-functional attribute values. For example, in the given scenario, when booking a hotel room, our researcher may prefer service s_{11} for its lower cost, or s_{12} for its compensation feature.

With this context in mind, next, we summarize the research issues and challenges that need to be dealt with while building non-functional based compositions in service-oriented systems:

- *RC1- Optimality and Multi-criteria handling*: services must be selected based on their non-functional properties such as QoS attributes. For instance, in our motivating scenario, if the researcher specifies cost optimization as requirement in her query, the plan (T1,T2) will be picked for travel arrangements with services s_{11} and s_{21}, as the best composition since it offers the less cost. However, each service advertises several QoS attributes, some of which may be conflicting: some services might have low response times but are costly, while others may be cheaper but slower (*e.g.*, s_{32} versus s_{31}, s_{41} versus s_{42}). In this case, the challenge lies in finding an optimal composition that reduces costs while maintaining performance and efficiency. This challenge arises because trade-offs between QoS attributes are often necessary, making it difficult to find an optimal composition of services that satisfies all constraints and requirements. Service selection and composition optimization is inherently a multi-criteria multi-objective problem.

- *RC2- Computational Complexity*: as the number of available services increases, the number of possible service combinations grows exponentially. Consider a process with n tasks, where m different

services are available for each task, then the total number of possible compositions is m^n (*e.g.*, with 6 tasks and 10 candidate services per task, the total number of compositions is $1.000.000$). Finding the best combination is NP-hard, meaning that an exhaustive search approach quickly becomes computationally impractical. To address this challenge, heuristic algorithms are necessary to efficiently identify near-optimal service compositions within a reasonable timeframe.

- *RC3- Atomicity and Consistency*: service composition involves multiple component services often executing concurrently within a distributed environment. These concurrently running services may access or modify shared data, making consistency a key challenge. Ensuring that the operations performed by one component service remain consistent with the states of other services is crucial for reliable execution. For example, consider a scenario in which our researcher uses TravelArrangement to book both a flight and a hotel room for a five-day trip, with a total budget of $1000€$ and an objective to optimize the execution cost. When starting the booking process, she expects all reservations to be completed successfully. The process is initiated using services s_{11} and s_{21}, each composed of three sequential operations: making a reservation, checking the budget, and proceeding with payment. Both services execute concurrently and independently complete their budget checks, each confirming the full budget of $1000€$. As a result, s_{11} proceeds to reserve a hotel room for $700€$, while s_{21} attempts to purchase a flight costing $500€$. This results in a budget overrun of $200€$. To restore consistency, the researcher would likely need to cancel one or both of the reservations. Doing so may require her contacting customer support, potentially incurring cancellation fees and resulting in additional effort and frustration. This situation not only leads to a poor user experience but may also damage the reputation of service providers. To avoid this, transactional guarantees are essential in service composition. A transactional composite service ensures that all component services either complete successfully or are fully rolled back or compensated, maintaining system consistency and data integrity. For example, in a transactional composition involving services s_{12} and s_{23}, if an inconsistency occurs during the execution of s_{23} (*e.g.*, flight payment), then hotel reservation in s_{12} can be automatically compensated, avoiding incomplete reservation.

- *RC4- Recovery from failures*: component services often run in distributed and unreliable networks where failures are common. If one service fails unexpectedly, the entire composition may be left in an incomplete or inconsistent state, providing users with only partial results. During composite service execution, when a component service fails, the system should be able to handle and recover from the failure. Recovery can be implemented using two strategies: backward recovery, and forward recovery. Backward recovery use transactional principles, particularly the all-or-nothing semantics. It restores the system to a previously correct state by rolling back changes made after the occurrence of a failure. In contrast, forward recovery aims to transition the system into a correct state. It may substitute failed services to enable the composition to continue functioning. For service composition, both strategies can be combined in two ways: if a failed service cannot be retried and no suitable alternative exists, the service execution is aborted, and all completed services are compensated. Alternatively, failed services are first compensated, the system rolls back to a consistent state, and execution resumes using alternative tasks. For example, consider the scenario in which our researcher uses HotelBooking service s_{12} and FlightBooking service s_{23}. If FlightBooking service s_{23} fails after the hotel room has already been purchased, our researcher will face an issue: she has paid for a room that she can no longer use, as her flight ticket booking failed. With a proper recovery approach, the flight reservation could succeed (service s_{23} retried), or an alternative solution could be offered (switch to service s_{22}), or flight reservation be automatically canceled (rolled back) and hotel reservation be compensated.

- *RC5- Multi-dimensional Trust*: after securing accommodation and purchasing a transport ticket, our researcher must submit a travel request for approval by her department director. The request may either be approved or returned with comments for revision. Only after obtaining approval is she authorized to attend the conference. Consider a scenario where the department director does not approve the researcher's trip because flying to Turin violates the institution's policy on reducing $CO2$ emissions. Instead, the director recommends traveling by train or car. Suppose the researcher prefers to travel by car and considers using a CarSharing service. She must choose between two ser-

vices: s_{41} provided by $CS1$, and s_{42} offered by $CS2$. CarSharing services often rely on individual or community drivers offering their own vehicles, this implies significant variability in service quality such as compliance with safety standards, and guarantees for driver punctuality. These factors introduces a new challenge for service composition: how to select a trustworthy provider offering a good service?. To make an informed choice between $CS1$ with service s_{41} and $CS2$ with service s_{42}, our researcher would probably like to consider whether the CarSharing provider is popular and well-known in her community, whether the driver is reliable, and whether past passengers reported positive experiences. She would likely prefer the CarSharing provider she perceives as more trustworthy, with reliable and well-rated driver. She might even prefer to rely on a recommendation from a (Italian) colleague who has previous passenger experience with either s_{41} or s_{42}.

1.3 Overview of Contributions

The research activities presented in this manuscript, summarized in Table 1.2 and Figure 1.3, are articulated along three complementary research lines.

Firstly, being interested in understanding precisely the foundations of service-oriented systems, we consider *service selection and composition* as a central concept in the first set of contributions. The primary goal was to investigate various quality attributes and optimization techniques in service selection and composition, addressing several research challenges outlined earlier in Section 1.2. In a first line of work, El Haddad et al. [78, 80][1], we developed heuristic methods for service selection, aiming to achieve global transactional correctness while maintaining locally (*i.e.*, at component services level) optimal QoS. These contributions responded to the need for efficient service selection where global transaction properties such as atomicity, consistency, and isolation must be preserved. After that, we extended our work to heuristic-based service composition in Blanco et al. [34], and Cardinale et al. [47, 48]. In this line of work, we used meta-heuristic search solutions that simultaneously considered functional requirements, QoS constraints, and transactional properties. Building on this foundation, and acknowledging that exact methods can guarantee globally optimal solutions, we then developed exact optimization techniques for service selection problem. In El Haddad et al. [82], we introduced fairness objective to ensure equitable QoS-based service selection among users. This shift also motivated a deeper theoretical investigation into the complexity of the service composition problem. In Abu-Khzam et al. [3], we established formal complexity results and provided corresponding proofs. Notably, all previous approaches, both heuristic-based and exact-based, treated services as black boxes, abstracting away their internal behavior specification. To address this limitation, and with a solid foundation in exact optimization techniques, we proposed an exact-based approach for behavioral service composition, incorporating both transactional properties and QoS metrics. This line of work, detailed in Gamez et al. [92, 93], allowed us to consider not only the non-functional specification but also behavioral specifications of services.

Secondly, in our pursuit to enhance the reliability of service compositions, and acknowledging that failures may occur during service execution, it becomes essential to consider effective *failure recovery strategies*. Within this line of research, our goal was to explore recovery mechanisms that ensure correct and dependable execution of composite services, particularly by leveraging the transactional properties of their component services. As a foundational step, in Cardinale et al. [50], we introduced a hybrid forward and backward recovery mechanism using Coloured Petri Nets. This approach supports forward recovery via re-execution or substitution of failed services, and backward recovery through compensation-based rollback to a previously consistent state, all while preserving global transactional semantics. To address the limitations of strict atomicity, in Cardinale et al. [51], we proposed a fuzzy atomicity model combining transactional guarantees with checkpointing. This model relaxes the traditional all-or-nothing principle to a more flexible "fuzzy all-something-or-(almost)nothing" property, better suited to partial successful executions. Finally, in Cardinale et al. et al. [52], we extended this model to introduce greater expressiveness and user-centric control. In this work, users can specify acceptable outcomes based on their utility, and the system can adapt recovery policies dynamically based on runtime state of composite service execution.

Thirdly, given that service composition often involve discovering and invoking services from unknown or unverified third-party providers, *trust* emerges as a crucial non-functional characteristic to consider.

[1] References to papers I co-authored appear in blue (*e.g.*, Gamez et al. [93] is a self-citation), while all others are in black.

Table 1.2: Contributions - lecture grid

Contributions	Quality Metrics	Optimization	Research challenges	Chapters	Fig. 1.3
[78, 80]	QoS & TP	Heuristic	RC2, RC3 (Service Selection)	Chapter 3	A
[34, 47, 48]	QoS & TP	Heuristic	RC2, RC3 (Service Composition)	Chapter 3	B
[82]	QoS	Exact	RC1 (Fair Service Selection)	Chapter 3	C
[3]	QoS	Exact	RC1 (Service Composition Complexity)	Chapter 3	D
[92, 93]	QoS & TP	Exact	RC1, RC3 (Behavioral Service Composition)	Chapter 3	E
[50]	QoS & TP	Heuristic	RC3, RC4 (Forward and Backward Recovery)	Chapter 4	F
[51, 52]	TP	Heuristic	RC3, RC4 (Semantic Recovery and Checkpointing)	Chapter 4	G
[152, 153, 155, 154, 156]	QoS & Trust	Heuristic	RC5 (Service Selection)	Chapter 5	H
[157]	QoS & Trust	Exact	RC5 (Service Composition)	Chapter 5	I

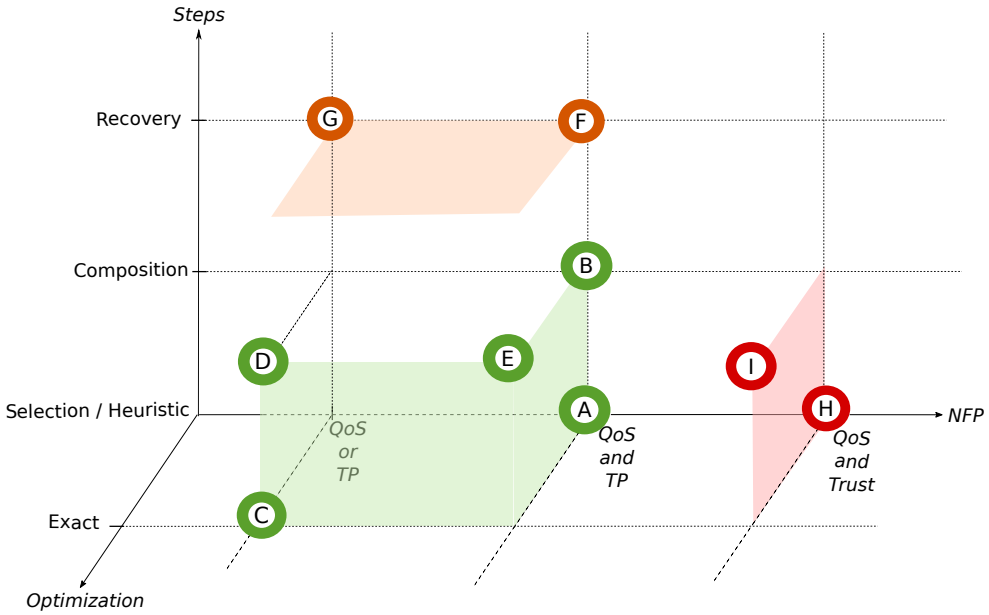


Figure 1.3: My contributions in service-oriented systems

Proposing *trust-based service selection and composition* is the main goal of our third line of research. Specifically, we propose a multi-agent approach grounded in trust and social network interactions. In a first line of work, Louati et al. [152, 153, 155, 154, 156], we enhanced service discovery and selection by introducing a trust model comprising four dimensions: societal, expertise, recommendation, and cooperation. The social dimension assesses whether a service provider is worth engaging with prior to service use. The expertise dimension evaluates the reliability and expected behavior of the service itself. The recommendation dimension determines the trustworthiness of agents offering recommendations and the credibility of those recommendations. The cooperation dimension enables agents to make informed decisions about potential partners in a service composition. Building on this foundation, in Louati et al. [157], we addressed the service composition problem through a trust-based dynamic coalition formation process. In this model, agents, each equipped with a set of services and their advertised QoS attributes, cooperate to fulfill the requester query based on a decentralized decision-making process guided by trust.

All the contributions of the research lines presented above will be detailed in Chapter 3, Chapter 4 and Chapter 5 together with a corresponding state of the art.

1.4 Outline

This manuscript is divided into six chapters, of which the remaining ones are organized as follows:

In Chapter 2, we present necessary background knowledge that form the foundation of our research work, including definitions of user requirements, component service functional and non-functional specifications, and composite service specifications used in our approaches.

In Chapter 3, we present our contributions on efficiency and consistency in service selection and composition. We propose both heuristic and exact approaches for transactional-and QoS-based selection and composition. In addition, we discuss the theoretical complexity of QoS-aware service selection and behavioral service composition, and briefly summarize additional research work we had the opportunity to explore.

In Chapter 4, we present our contributions on reliability and failure recovery in service composition. We introduce forward and backward recovery through transactional properties of services. Then we define a fuzzy atomicity model to integrate transactional properties with a checkpointing mechanism. Beside, we briefly summarize a research work on concurrency control of composite service execution that we had the opportunity to study.

In Chapter 5, we present our contributions to trust-based service discovery, selection, and composition. For service discovery and selection, we introduce a trustworthiness evaluation of providers and their services, grounded in sociability, expertise, and recommendations, while for service composition, we propose a trust-based coalition formation approach.

In Chapter 6, we conclude the manuscript by summarising our contributions and presenting the broad outlines of our future research plan on large-scale service systems, with some perspectives.

CHAPTER 2

CONCEPTS AND DEFINITIONS

In this chapter, we provide an overview of the key concepts that form the foundation for our research work. We introduce definitions related to user requirements, component service functional and non-functional specifications, and composite service specification used in our approaches.

2.1 User Requirements

In all our approaches, user functional and non-functional requirements are formalized as a *Query*. In our approaches that incorporate QoS and transactional properties (detailed in Chapter 3 and Chapter 4), functional requirements are specified either through explicit input-output pairs provided by the user or as a predefined workflow. Non-functional requirements are always captured by indicating a desired global transactional property and by assigning user-defined weights to various QoS attributes, as illustrated below.

Definition 2.1.1. Let $Onto_A$ be the integrated ontology. A Query Q is a 4-tuple (I_Q, O_Q, W_Q, T_Q), where $I_Q = \{i \mid i \in Onto_A\}$ is the set input attribute whose values are provided by the user, $O_Q = \{o \mid o \in Onto_A\}$ is the set of output attributes whose values has to be produced by the system, $W_Q = \{(w_i, q_i) \mid w_i \in [0,1]$ with $\sum_i w_i = 1$ and q_i is a QoS criterion$\}$ is the set of weights for QoS attributes, and T_Q (or R_Q) is the required global transactional property: $T_Q \in \{T_0, T_1\}$. If $T_Q = T_0$, the system guarantees that a semantic recovery can be done by the user. If $T_Q = T_1$, the system does not guarantee the result can be compensated. In both cases, if the execution is not successful, nothing is changed on the system.

When functional requirements are expressed as workflows, we adopt Yet Another Workflow Language (YAWL) (Van Der Aalst and Ter Hofstede [227]) as a modeling language. However, any other expressive workflow language could have been employed. Figure 2.1 shows the YAWL symbols, while Figure 2.2 common control-flow patterns: sequence, parallel split (AND-split), exclusive choice(XOR-split), synchronization (AND-join) and simple merge (XOR-join). For example, Figure 2.2(a) shows a sequential pattern between activities (*i.e.*tasks) A_1 and A_2. If service s_1 is assigned to A_1 and s_2 to A_2, the resulting composite service is denoted as $(s_1; s_2)$ where the semicolon (;) indicates that s_1 executes before s_2. The parallel pattern (Figure 2.2(b)), with s_1 and s_2 respectively assigned to A_1 and A_2, produces the composite service $(s_1 // s_2)$ representing concurrent execution. For exclusive choice pattern (Figure 2.2(c)), the execution corresponds to one of the possible branches. Assigning s_1 to A_1 and s_2 to A_2 yields the composite service $(s_1 \mid s_2)$, indicating that either s_1 or s_2 is executed. For more details, see Van der Aalst et al. [226] for a comprehensive characterization of workflow constructs, and Jaeger et al. [122] for an in-depth discussion of control-flow patterns relevant to service composition.

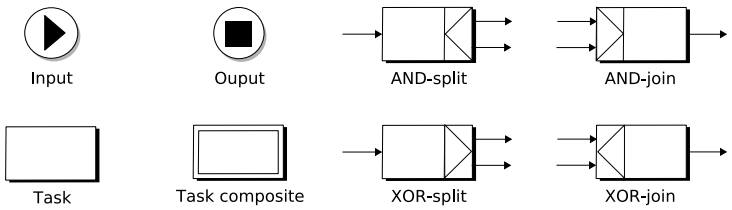


Figure 2.1: Symbols used in YAWL

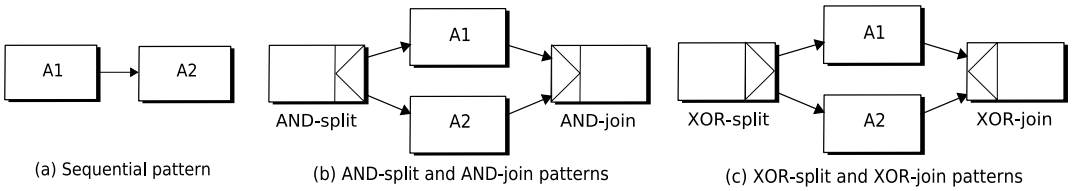


Figure 2.2: Workflow patterns

In the trust-based approaches presented in Chapter 5, users specify their functional requirements by indicating either a set of required services or a set of desired functionalities. Non-functional requirements are expressed through preferences over relationship types and the definition of trust threshold values, as illustrated below.

Definition 2.1.2. A query Q is a 5-uplet $(F, U, \alpha, \beta, \mu)$ where F is a set of required services $\{s_1, s_2, ..., s_l\}$ or functionalities $\{f_1, f_2, ..., f_l\}$, $U : R \mapsto]0, 1]$ is an utility function expressing user's preferences over relationship types in the social network, $\alpha \in [0, 1]$ is a trust in sociability threshold, $\beta \in [0, 1]$ is a trust in recommendation threshold and $\mu \in [0, 1]$ is a trust in expertise threshold.

Additionally, in these approaches, each individual user is assigned a self-interested agent and, an edge between two users represent a symmetric social relationship between them. This induce a multi-relation social network modeled by a graph defined as below.

Definition 2.1.3. Given a set $V = \{a_1, a_2, ..., a_n\}$ of agents and a set R of types of symmetric relationships with $R = \{R_1, R_2, ..., R_r\}$, a multi-relation social network (MRSN) is a connected graph $G =< V, E_1, E_2, ..., E_r >$ where $E_i \subseteq V \times V \, \forall i \in \{1, ..., r\}$ is the set of edges w.r.t the i-th relationship and $\forall i \neq i', E_i \cap E_{i'} = \emptyset$. Let $\rho : E \mapsto R$ be a function that links edges to the relationship they represent (*i.e.*, an edge $(a_k, a_j) \in E_i$ represents a social relationship of type R_i between a_k and a_j).

2.2 Component Service Specification

A comprehensive service description includes its functionality, inputs, outputs, and associated non-functional attributes. In the following, we first present our service functional description model, followed by our specifications for transactional, QoS, and trust attributes.

Functional Specification

In all our approaches, a service s_i is syntactically described by its functionality, its set of inputs needed to invoke the service, and its set of outputs that are produced by the service execution.

Definition 2.2.1. A service s is a n-uplet $(in, out, f, q^1, \ldots, q^d)$ where in is a set of inputs required to use the service, out is a set of outputs provided at the completion of the service, f is a functionality describing the provided capacity, and $q^1, \ldots, q^d$ are the advertised values of the d non-functional criteria.

Transactional Properties

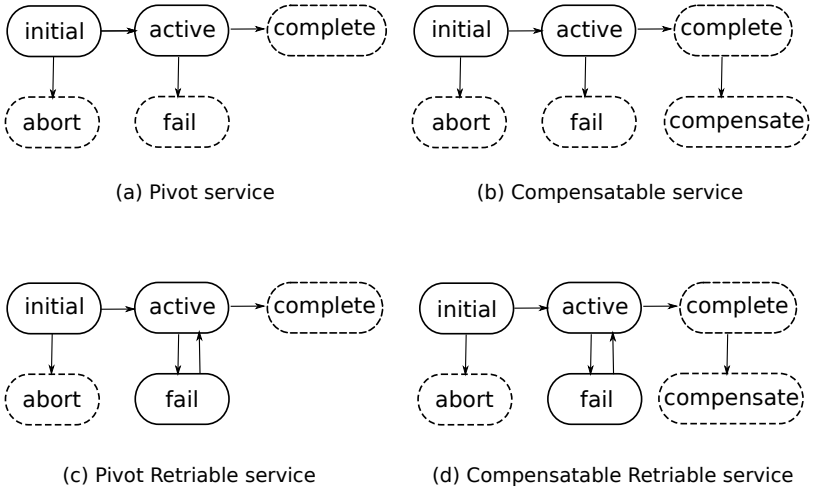
In our approaches that incorporate transactional properties (detailed in Chapter 3 and Chapter 4), we identified three key transactional properties of services, namely *pivot*, *compensatable* and *retriable*, which have also been widely adopted by other researchers, including Bhiri et al. [30], Gaaloul et al. [89], Li et al. [139], Maamar et al. [161], Montagut et al. [177], Zhao et al. [263].

Definition 2.2.2. A service is pivot (p) if once it successfully completes, its effects remains forever and cannot be semantically undone. If it fails it has no effect at all. A completed pivot service cannot be rolled back.

Definition 2.2.3. A service s is compensatable (c) if it exists another service s', or compensation policies, which can semantically undo the execution of s.

Definition 2.2.4. A service is retriable (r) if it guarantees a successfully termination after a finite number of invocations.

The retriable property is typically not used in isolation; rather, it is combined with other properties to define two transactional types: *pivot retriable* (pr) (equivalent to retriable) and *compensatable retriable* (cr). Figure 2.3 presents state diagrams illustrating these transactional properties, with final states represented by dotted lines. These properties are adaptations of the transactional properties originally identified by Mehrotra et al. [169] for MultiDatabase Systems, and they can be incorporated into the WSDL interface or the OWL-S profile of a service, as demonstrated in the work of Bhiri et al. [30] and Montagut et al. [177].



(a) Pivot service (b) Compensatable service

(c) Pivot Retriable service (d) Compensatable Retriable service

Figure 2.3: State diagrams of service transactional properties

Quality-of-Service Attributes

In our approaches that incorporate QoS properties (detailed in Chapter 3 and Chapter 4), we focused on the following five generic criteria for a component service s:

- Execution price ($q_{ep}(s)$): the fee that a requester must pay to invoke the service s,
- Execution duration ($q_{ed}(s)$): the expected time delay between invoking s and receiving the results,
- Reputation ($q_r(s)$): a measure of the trustworthiness of s, typically computed as the average ranking provided by end users,
- Successful execution rate ($q_{sr}(s)$): the probability that s will correctly respond to the user's request,
- Availability ($q_a(s)$): the probability that s is accessible and operational.

In our trust-based approaches, we considered the following three QoS attributes for a service s_{jl}:

- *Specialization* ($Sp(s_{jl})$): is the percentage of successful use of an agent's service s_{jl} compared to the other services it offers. It is defined as the ratio between the number of times that a service s_{jl} has been successfully completed (Maaradji et al. [163]) and the total number of successful executions of the agent a_j regardless of the used service.

$$Sp(s_{jl}) = \frac{Nb_{success}(s_{jl})}{\sum_{t=1}^{m_j} Nb_{success}(s_{jt})} \tag{2.1}$$

  where $Nb_{success}(s_{jl})$ is the number of successful executions of s_{jl} and m_j is the number of services offered by agent a_j. This means that the more the service s_{jl} is sought for in the social network the more a_j is recognized as an expert in this field.

- *Reliability* ($Re(s_{jl})$): is the probability that a service s_{jl} is operational at the time of invocation. It is computed as the rate between the number of successful executions $Nb_{success}(s_{jl})$ and the total number of functionality invocations $Nb_{invoc}(s_{jl})$.

$$Re(s_{jl}) = \frac{Nb_{success}(s_{jl})}{Nb_{invoc}(s_{jl})} \tag{2.2}$$

- *Experience rating* ($Eval(a_k, s_{jl})$): is the rating of the service realization quality. After the execution of a service s_{jl}, an agent a_k gives an evaluation $\nu \in [0, 1]$ of this execution reflecting its experience feedback as a customer of this service. Unlike the previous attributes, the evaluation of this attribute results from a subjective perception. Let $Eval(a_k, s_{jl})$ be the average of the experience ratings of s_{jl} for n uses by a_k.

$$Eval(a_k, s_{jl}) = \frac{\sum_{x=1}^{n} \nu_x}{n} \tag{2.3}$$

Trust Attribute

In our trust-based approaches detailed in Chapter 5, we define trust in expertise $ET(a_k, a_j, s_{jl})$ that an agent a_k has in a service s_{jl} offered by an agent a_j as the aggregation of three above QoS attributes (Lalanne et al. [134]). It is computed as follows:

$$ET(a_k, a_j, s_{jl}) = Sp(s_{jl}) \times Re(s_{jl}) \times Eval(a_k, s_{jl}) \tag{2.4}$$

2.3 Composite Service Specification

Fulfilling a user query may require combining several services, resulting into a composite service. As described in Chapter 1, we use diagram-based representation to model service composition. In some of our approaches, we use (coloured) Petri nets to model service compositions, while for others we employ graph-based representations as illustrated below.

(Coloured) Petri net-based representation

In some of our approaches in Chapter 3 and Chapter 4, we model the service registry using a coloured Petri net, referred to as a Web Service Dependency Net (WSDN), defined as follows:

Definition 2.3.1. A WSDN is a 4-tuple (A, S, F, ξ), where:

- A is a finite non-empty set of places, corresponding to input and output attributes of the services in the registry such that $A \subset Onto_A$;
- S is a finite set of transitions corresponding to the set of services in the registry;
- $F : (A \times S) \cup (S \times A) \rightarrow \{0, 1\}$ is a flow relation indicating the presence (1) or the absence (0) of arcs between places and transitions defined as follows: $\forall s \in S$, $(\exists a \in A \mid F(a, s) = 1) \Leftrightarrow (a$ is an input attribute of $s)$ and $\forall s \in S$,$(\exists a \in A \mid F(s, a) = 1) \Leftrightarrow (a$ is an output attribute of $s)$;
- ξ is a color function such that $\xi : C_A \cup C_S$ with: $C_A : A \rightarrow \Sigma_A$, a color function such that $\Sigma_A = \{I, a, ar, c, cr\}$ representing, for $a \in A$, either the transactional property of the composite service that can produce it or the user input (I), and $C_S : S \rightarrow \Sigma_S$, a color function such that $\Sigma_S = \{p, pr, a, ar, c, cr\}$ representing the transactional property of $s \in S$.

Note that, the transactional properties of a composite service $\{a, ar, c, cr\}$ are defined below.

Definition 2.3.2. A marked Web Service Dependency Net is a pair $(WSDN, M)$, where M is a function which assigns tokens (values) to places such that $\forall a \in A$, $M(a) \subseteq \{\emptyset, Bag(\Sigma_A)\}$, where Bag corresponds to a set which can contain several occurrences of the same element.

Definition 2.3.3. The initial marking M_Q depends on the user query Q and is defined as: $\forall a \in (A \cap I_Q)$, $M_Q(a) = \{I\}$ and $\forall a \in (A - I_Q)$, $M_Q(a) = \emptyset$.

According to coloured Petri net notation, for each $x \in (A \cup S)$, $({}^\bullet x) = \{y \in A \cup S : F(y, x) = 1\}$ is the set of its predecessors, and $(x^\bullet) = \{y \in A \cup S : F(x, y) = 1\}$ is the set of its successors.

Definition 2.3.4. A marking M enables a transition s iff all its input places contain a token $(\forall x \in ({}^\bullet s)$, $M(x) \neq \emptyset)$ and at least one of the following conditions is verified:
1. $(\forall a \in A, M(a) \in \{I, \emptyset\})$
2. $(C_S(s) = cr)$
3. $(C_S(s) \in \{pr, ar\}) \wedge [\forall a \in (A - {}^\bullet s), M(a) \in \{\emptyset, Bag(\{I, ar, cr\})\}]$
4. $(C_S(s) = c) \wedge [\forall a \in (A - {}^\bullet s), M(a) \in \{\emptyset, Bag(\{I, c, cr\})\}] \wedge [\forall x \in ({}^\bullet s), M(x) \in Bag(\{I, c, cr\})]$
5. $(C_S(s) \in \{p, a\}) \wedge [\forall a \in (A - {}^\bullet s), M(a) \in \{\emptyset, Bag(\{I, cr\})\}] \wedge [\forall x \in ({}^\bullet s), M(x) \in Bag(\{I, c, cr\})]$

Definition 2.3.5. Let $(WSDN,M)$ be a marked WSDN. Its color is: $\mathcal{C}_M \in \{I, a, ar, c, cr\}$. $\mathcal{C}_M = I$, when no transition has been fired (i.e., when no WS has been selected and there is no resulting CWS). Otherwise, $\mathcal{C}_M$ represents the aggregated TP of the resulting CWS and is updated each time a transition is fired.

Definition 2.3.6. The firing of a fireable transition s for a marking M defines a new marking M', denoted as $M \xrightarrow{s} M'$, such that :

1. Tokens are added to the output places of s depending on the color of s and on the color of the tokens contained by the input places of s, according to the following rules:
   if ($\exists x \in ({}^\bullet s) \mid a \in M(x)$), then $\forall y \in (s^\bullet)$, $M'(y) \leftarrow M'(y) \cup \{a\}$
   else if ($\exists x \in ({}^\bullet s) \mid ar \in M(x)$), then $\forall y \in (s^\bullet)$, $M'(y) \leftarrow M'(y) \cup \{ar\}$
   else if [$(\exists x \in ({}^\bullet s) \mid c \in M(x)) \wedge (C_S(s) \in \{p, pr, a, ar\})$],
   then $\forall y \in (s^\bullet)$, $M'(y) \leftarrow M'(y) \cup \{a\}$
   else if [$(\exists x \in ({}^\bullet s) \mid c \in M(x)) \wedge (C_S(s) \in \{c, cr\})$],
   then $\forall y \in (s^\bullet)$, $M'(y) \leftarrow M'(y) \cup \{c\}$
   else /*in this case: $\forall x \in ({}^\bullet s)$, $M(x) \in Bag(\{I, cr\})$*/
   $\forall y \in (s^\bullet)$, $M'(y) \leftarrow (M'(y) \cup C_S(s))$ if $C_S(s) \in \{a, ar, c, cr\}$,
   $M'(y) \leftarrow (M'(y) \cup \{a\})$ if $C_S(s) = p$, and $M'(y) \leftarrow M'(y) \cup \{ar\}$ if $C_S(s) = pr$

2. Tokens are deleted from input places of s, if they do not belong to O_Q:
   $\forall x \in ({}^\bullet s - O_Q)$, $M(x) \leftarrow \emptyset$,

3. Color $\mathcal{C}_{M'}$ of the resulting $(WSDN, M')$ (see Definition 2.3.5) is updated, according to the following rules:
   if $(\mathcal{C}_M \in \{I, cr\})$ and $C_S(s) = p$ then $\mathcal{C}_{M'} \leftarrow a$
   else if $(\mathcal{C}_M \in \{I, cr\})$ and $C_S(s) = pr$ then $\mathcal{C}_{M'} \leftarrow ar$
   else if $(\mathcal{C}_M \in \{I, cr\})$ and $C_S(s) \in \{a, ar, c, cr\}$ then $\mathcal{C}_{M'} \leftarrow C_S(s)$
   else if $(\mathcal{C}_M = c\})$ and $C_S(s) \in \{p, pr, a, ar\}$ then $\mathcal{C}_{M'} \leftarrow a$
   else $\mathcal{C}_{M'} \leftarrow \mathcal{C}_M$

Definition 2.3.7. A firing sequence $\sigma = \{s_1, \ldots, s_n \mid s_i \in S\}$ is a correct sequence of fired transitions starting from M_Q iff there are markings $M_1, ..., M_n$ such that $M_Q \xrightarrow{s_1} M_1 \ldots M_{n-1} \xrightarrow{s_n} M_n$; this is denoted as $M_Q \xrightarrow{\sigma} M_n$.

Definition 2.3.8. Let $\mathcal{O}_M$ be the set of the produced attributes by a firing sequence σ such that: $\mathcal{O}_{Mn} = \{a \in (\cup_{(s_i^\bullet) \mid s_i \in \sigma}) \mid M_Q \xrightarrow{\sigma} M_n\}$. A transition s is said to be a cut-off transition iff its firing does not change the set of the already produced attributes (*i.e.*, iff: $M_Q \xrightarrow{\sigma} M_n \xrightarrow{s} M_{n+1}$ and $\mathcal{O}_{M_n} = \mathcal{O}_{M_{n+1}}$).

Graph-based representation

A composite service, CS_Q, responds to a query $Q = (I_Q, O_Q)$ by combining a set of services to be executed sequentially or in parallel according to an execution flow imposed by data or control dependencies, to obtain O_Q from I_Q. In some of our approaches of Chapter 4, the execution flow of such composite service is represented by a graph as proposed in Angarita et al. [15] and described below.

Definition 2.3.9. A composite service graph, denoted as $G = (V \cup I_Q \cup O_Q, A)$, is a directed acyclic graph with the following considerations:

- Nodes in V represent services, such that $V = \{s_i, i = 1..m\}$ and s_i is a component service;
- I_Q, O_Q, are respectively input and output user parameters expressed in query Q. I_Q are the initial nodes of G and O_Q are the final nodes of G;
- Arcs (ip_j, s_i) in A, with $ip_j \in I_Q$ and $s_i \in V$, denote the input parameters for a service $s_i \in V$ such that $ip_j \in I_{s_i}$;

- Arcs (s_i, op_j) in A, with $s_i \in V$ and $op_j \in O_Q$, denote the output parameters for a service $s_i \in V$ such that $op_j \in O_{s_i}$;
- Arcs (s_i, s_j) in A, with $s_i, s_j \in V$ denote the execution flow between two services s_i and s_j such that $O_{s_i} \cap I_{s_j} \neq \emptyset$.

In composite service graph, arcs between services represent execution flow, defined by data or control flow relationships. A data flow exists when the output of one service serves as input to another. A control flow exists when execution order requires one service to wait for the completion of another. Thus, an execution flow between two services indicates either data dependency or control dependency.

Transactional Property

The transactional property of a composite service depend on the transactional properties of its component services. To address this, we extended the transactional properties of component services to composite services in El Haddad et al. [78], defining *atomic*, *compensatable*, and *retriable* composite services, as outlined in the following definitions:

Definition 2.3.10. A composite service is atomic if once all its component services complete successfully, their effect remains forever and cannot be semantically undone. On the other hand, if one component service does not complete successfully, then all previously successful component services have to be compensated.

Definition 2.3.11. A composite service is compensatable (c) if all its component services are compensatable.

Definition 2.3.12. An atomic or a compensatable composite service is retriable (r) if all its components are retriable.

Definition 2.3.13. A transactional composite service is a composite service whose transactional behavioral property is in $\{a, ar, c, cr\}$.

To ensure these transactional properties for a composite service, the transactional properties of its component services must respect several rules defined in Table 2.1. These rules impose restrictions on sequential and parallel executions of services.

Table 2.1: Rules for transactional composite services

Transactional property of a service	Sequential compatibility	Parallel compatibility
p, a	pr, ar, cr (rule 1)	cr (rule 2)
pr, ar	pr, ar, cr (rule 3)	pr, ar, cr (rule 4)
c	p, pr, a, ar, c, cr (rule 5)	c, cr (rule 6)
cr	p, pr, a, ar, c, cr (rule 7)	p, pr, a, ar, c, cr (rule 8)

Quality-of-Service

A composite service CS has the same quality properties as a component service, *i.e.*, execution price, execution duration, reputation, successful execution rate, and availability. The QoS of a composite service is evaluated by using the aggregation functions defined in Table 2.2. Activities in all execution paths between AND-split and AND-join are considered in the aggregation functions. While, activities in only one execution path between XOR-split and XOR-join constructs are considered.

Table 2.2: Aggregation functions for QoS criteria

Criteria	**Aggregation function**
Price	$q_{ep}(CS) = \sum_{i=1}^{n} q_{ep}(s_i)$
Duration	$q_{ed}(CS) = \sum_{i=1}^{n} q_{ed}(s_i)$
Reputation	$q_r(CS) = \frac{1}{n} \sum_{i=1}^{n} q_r(s_i)$
Success rate	$q_{sr}(CS) = \prod_{i=1}^{n} q_{sr}(s_i)$
Availability	$q_a(CS) = \prod_{i=1}^{n} q_a(s_i)$

Trust in Expertise

In our approach in Louati et al. [157], detailed in Chapter 5, service composition is performed by self-interested agents equipped with a set of services along with their QoS attribute values. Agents cooperate in a coalition formation to collectively provide multiple composite services that meet the complex user query. Trust in expertise, denoted $Exp_r[z]$, for a given coalition c_z is defined below as the average sum of the expertise of its members.

$$Exp_r[z] = \frac{\sum_{a_t \in c_z} ET(a_t)}{|c_z|} \tag{2.5}$$

Recall that trust in expertise of a member (*i.e.*, agent a_t) is a score established from the QoS values of its offered service(s) in the underlying coalition computed following Equation 2.4.

CHAPTER 3

Efficiency and Consistency in Service Selection and Composition

In this chapter, we address the issues of ensuring efficient and consistent execution of service compositions. Ensuring efficiency involves selecting and composing services that optimize key Quality-of-Service attributes such as execution time, cost, and availability. Achieving consistency involves utilizing transactional properties to preserve data integrity, and prevent partial or conflicting updates during concurrent execution of services. As such, we proposed both heuristic and exact approaches for service selection and composition that integrate QoS and transactional properties. Our contributions evolve from early heuristic methods for selection and composition to more advanced exact optimization techniques capable of handling both quantitative and qualitative attributes. This chapter highlights the contributions presented in El Haddad et al. [78, 80], Blanco et al. [34], Cardinale et al. [47, 48], El Haddad et al. [82], Abu-Khzam et al. [3], and Gamez et al. [92, 93]. Nevertheless, we conclude the chapter with a brief overview of additional research work that we had the opportunity to study.

3.1 Motivations

As highlighted in the Introduction (Chapter 1), service composition is a key aspect of service-oriented systems, combining services with different functional, behavioral, and non-functional properties to build more complex ones. Since multiple providers may offer functionally equivalent services, non-functional properties serve as the main criterion for selecting a single candidate per task. Bearing this in mind, service composition must address efficiency and consistency challenges. Efficiency ensures optimized performance and cost-effectiveness, while consistency guarantees data integrity and correctness of execution results across component services.

Efficiency is a fundamental concern in service composition.For instance, by minimizing latency, efficient compositions enhance the system responsiveness; and by maximizing resource utilization, they reduce infrastructure load, enabling the system to support a greater number of concurrent users. To achieve efficiency, *Quality-of-Service* (QoS) non-functional properties such as execution time, execution price, and availability are considered. This results in what is known as the QoS-aware service selection and composition problem (Zeng et al. [257]), where the goal is to choose the best services or compositions that meet user's QoS constraints. However, these constraints often conflict, requiring trade-offs among QoS attributes. As a result, this problem is recognized as a multi-criteria multi-objective optimization problem. To address it, the literature offers both *exact* and *heuristics* approaches.

Consistency is another cornerstone of successful service composition, ensuring that concurrently executing services preserve data integrity and avoid partial updates that leave the composition in an invalid state. For instance, if a composite service first charges a customer and then books a ticket, consistency mechanisms must ensure that the ticket is not booked if the payment fails. To enforce consistency,

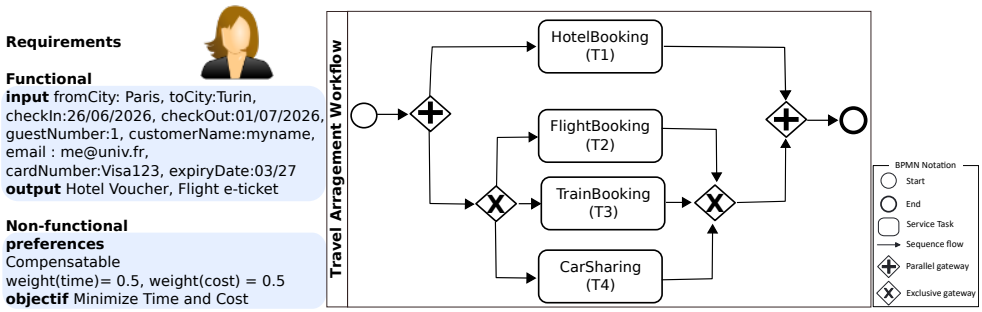


Figure 3.1: TravelArrangement booking motivating example

Table 3.1: Available candidate component services for TravelArrangement

Task	Service	Response Time	Execution Cost	Transactional Property	Provider
HotelBooking	s_{11}	$15mn$	1€	–	HB1
(T1)	s_{12}	$10mn$	3€	*compensatable*	HB2
FlightBooking	s_{21}	$20mn$	1€	–	FB1
(T2)	s_{22}	$10mn$	10€	*compensatable*	FB2
	s_{23}	$8mn$	5€	*retriable*	FB3
TrainBooking	s_{31}	$20mn$	5€	–	TB1
(T3)	s_{32}	$18mn$	8€	–	TB2
CarSharing	s_{41}	$5mn$	2€	*compensatable*	CS1
(T4)	s_{42}	$12mn$	15€	–	CS2

transactional properties of services are considered during composition to help supporting ACID guarantees (Gray [103]): Atomicity (all-or-nothing execution), Consistency (valid state transitions), Isolation (no interference from concurrent services), and Durability (persistence of committed results). Thus, choosing transactional services helps maintain valid state transitions, support concurrent execution safely, and prevent partial or conflicting updates, thus preserving data integrity.

To illustrate the need to address both challenges, let us return to our motivating scenario.

Motivating example. As shown in Figure 3.1, consider the case where the researcher specifies, as part of her functional requirements, booking a hotel room and traveling to Turin by plane. She further indicates cost optimization and compensatable transactional preference as non-functional requirements. Under these conditions, the plan (T1,T2) is selected for her travel arrangements.

✗ If only efficiency is considered ($weight(time) = 0.0$, $weight(cost) = 1.0$), services s_{11} and s_{21} form the optimal composition, with the lowest cost of 2€and an execution time of 20 minutes, since they can run concurrently. However, this may cause integrity issues if, for example, no hotel rooms are available in Turin or the flight cost is too expensive.

✗ Conversely, if only consistency is taken into account ($weight(time) = 0.0$, $weight(cost) = 0.0$), services s_{12} and s_{22} are chosen for their strong transactional guarantees, though at a higher cost of 13€and a shorter execution time of 10 minutes.

✓ When efficiency and consistency are considered together, a compromise may be selected—for instance, services s_{12} and s_{23}, which balance cost (8€), execution time (18 minutes), and some transactional guarantees.

Thus, both transactional and QoS properties should be considered during service selection and composition problem. Balancing QoS and transactional aspects is vital to creating efficient and robust service compositions (Liu et al. [144]). However, in the literature, the problem was generally addressed either from the QoS side (Jaeger et al. [122], Menasce [172], Wu et al. [237], Zeng et al. [257], Zhang et al. [259])

or from the transactional side (Bhiri et al. [30, 31], Li et al. [139], Montagut et al. [177], Portilla et al. [194] separately.

The rest of the chapter is organized as follows. In Section 3.2, we outline state-of-the-art research work for handling transactional properties and optimizing QoS associated with service selection and composition and summarizes our contributions in this research line. In Section 3.3, we propose heuristic transactional-and QoS-based approaches, while in Section 3.4, we propose exact transactional-and QoS-based approaches. In Section 3.5, we overview additional research work that we had the opportunity to study. In the last section, we conclude the chapter by summarizing our findings.

3.2 State-of-the-art and Contributions

The research on service selection and composition has undergone significant evolution over the past two decades, progressively addressing the challenges of efficiency and consistency.

QoS-based Service Selection and Composition. In the early 2000s, research on service-oriented systems focused on optimizing QoS attributes due to the growing number of functionally similar services. Early work by Menasce [172] introduced probabilistic models for QoS attributes such as cost and execution time, while Jaeger et al. [122] proposed pattern-based QoS aggregation to ensure composite services meet required QoS levels. Later, the QoS-aware service selection and composition problem was extensively studied and commonly formulated as a *combinatorial optimization* problem (Alrifai et al. [11], Ardagna et al. [16], Bonatti and Festa [35], Schuller et al. [202, 203], Trummer et al. [221], Yu et al. [251], Zeng et al. [257]), with surveys provided in El Haddad [79], Moghaddam et al. [175], and Strunk et al. [213]. Most studies acknowledged that the problem is NP-hard and equivalent to Multi-Dimension Multiple-Choice Knapsack problem (Ardagna et al. [16], Gabrel et al. [90]). Solutions in the literature broadly fall into two main categories: *exact* approaches ensuring optimality, and *heuristic* approaches delivering near-optimal results with reduced computational cost.

Exact approach foundational work by Zeng et al. [257] formalized service selection as a QoS-aware optimization problem, introducing a Mixed Integer Programming (MIP) formulation to support global planning through the decomposition of execution paths. Other influential contributions during this period include Alrifai et al. [11], Ardagna et al. [16], Bonatti and Festa [35], Schuller et al. [202, 203], Trummer et al. [221], Yu and Lin [251], who further refined exact models and problem formulations. Bonatti and Festa [35] proposed a matching-based approach that framed service selection as a compatibility and preference-driven binding problem. Subsequent work diversified modeling techniques to better capture workflow structures and constraints. Yu and Lin [251] developed two exact methods based on the Multiple Choice Knapsack Problem for sequential workflows, and Constrained Shortest Path Problem for general workflow topologies. Ardagna and Pernici [17] extended the MIP model to incorporate local and global constraints and cycles handling. Cardellini et al. [46] proposed a Linear Programming (LP) model for optimizing end-to-end QoS in composite services offered by brokers to multiple users. In parallel, Schuller et al. [202] addressed both structured and unstructured workflows through Integer Linear Programming (ILP). Additionally, Gabrel et al. [90] focused on the complexity of single-criterion scenarios, offering a refined MIP-based solution, while Trummer et al. [221] exposed computational limits via an exact exponential time algorithm.

Over time, research shifted toward heuristic and metaheuristic methods to improve scalability while maintaining solution quality. Genetic algorithms (Canfora et al. [44], Yang et al. [244]), dynamic programming heuristics (Wu et al. [237]), and ant colony optimization (Zhang et al. [258]) emerged as practical alternatives. Canfora et al. [44] proposed a genetic algorithm for QoS-aware workflow re-planning. Wu et al. [237], proposed a dual-mode approach balancing accuracy and speed, one using dynamic programming for highly accurate results at slower selection times and another for good enough results at faster selection times. Zhang et al. [258] introduced a decomposition strategy with general flow structures, framing service selection as a multi-objective optimization problem solved via ant colony algorithms. Other contributions include Zeng et al. [257] who proposed both local optimization and global planning strategies; Kokash [130] who extended the previous work by considering additional factors (*e.g.*, service failure probabilities, response times, and execution costs) along with the composite service graph

structure; and Zhang et al. [259] who integrated global user constraints into service selection. Later on, Trummer et al.[221] proposed two algorithms: a polynomial-time heuristic without guaranteed error bounds, and a polynomial-time approximation algorithm offering such guarantees.

Modern approaches increasingly leverage hybrid techniques, combining exact optimization for smaller instances with heuristic or metaheuristic methods for larger scale instances (Alrifai et al. [10], Ben Mabrouk et al. [24], Izquierdo et al. [121]). While promising, these approaches fall outside the scope of this manuscript.

Transactional-based Service Selection and Composition. Concurrently to the emphasis on efficiency, during the mid to late 2000s, researchers increasingly recognized the critical importance of consistency and reliability in service compositions. Early contributions, such as Bhiri et al. [30], began to explore how transactional properties, including compensability and retriability, could be integrated into service selection to prevent partial executions and maintain system correctness. Authors proposed an algorithm to validate composite services based on the transactional properties of their components and predefined Acceptable Termination States (ATSs). Similarly, Montagut et al. [177] used ATSs to guide service selection, incorporating user-defined workflows as functional constraints. However, Liu et al. [146] later pointed out that specifying all possible ATSs becomes increasingly complex and impractical in large-scale workflows. To address this scalability issue, alternative approaches have employed transactional rules as correctness criteria. For instance, Li et al. [139] proposed rules specifying how transactional properties, such as pivot, retriable, and compensatable services, can be composed based on workflow structure. They showed that the transactional property of a composite service can be inferred from its components and the structure of the workflow. Other researchers have extended this line of work by investigating transactional workflows (Bhiri et al. [31], Liu et al. [146]), Advanced Transactional Models (ATM) (Lakhal et al. [133], Vidyasankar et al. [229]), and graph-based algorithms that capture input-output dependencies among component services (Brogi et al. [40]).

Transactional- and QoS-based Service Seclection and Composition. The interplay between transactional consistency and QoS optimization began to receive significant attention in the 2010s, giving rise to research that jointly addressed both concerns. Studies such as those by Mei et al. [170], Wu and Yang [238], and Liu et al. [147, 150] highlighted how transactional support, especially compensation mechanisms, can significantly impact QoS metrics, potentially increasing execution costs and latencies. Mei et al. [170] proposed a method for evaluating the QoS of transactional composite services by modeling compensation costs and identifying dominant QoS metrics based on different execution patterns. Their approach used paired Petri nets to model normal control flow and reverse flow triggered in case of compensation. Wu and Yang [238] developed a model to predict the aggregated QoS of composite services, incorporating compensation costs computed based on the probability of exceptions occurring. Their algorithm estimates the average QoS for a composition by considering all component services and their respective compensation actions. Similarly addressing compensation costs, Liu et al. [147] proposed several algorithms for scheduling transactional composite services composed solely of compensatable services. These algorithms produced schedules that account for both temporal and cost constraints, including time-aware scheduling to preserve atomicity and cost-aware scheduling to minimize overall compensation costs.

For a classification of transaction-aware service selection and composition approaches, distinguishing between those based solely on transactional properties and those that integrate both transactional and QoS considerations, readers can check our survey provided in Cardinale et al. [49]. We evaluated these approaches using criteria such as the underlying transactional models, control flow strategies, verification mechanisms, composition phases addressed, and compliance with protocols or standards.

Recent research continues to address efficiency and consistency challenges in dynamic, distributed environments such as the Cloud, IoT, and Microservices architectures (Awan et al. [20], Čilić et al. [65], Khadir et al. [128], Zhao et al. [262]). Additionally, emerging trends involve incorporating machine learning techniques to optimize non-functional properties of service compositions under uncertain or rapidly changing conditions (Hammoum et al. [111], Zhang et al. [260]).

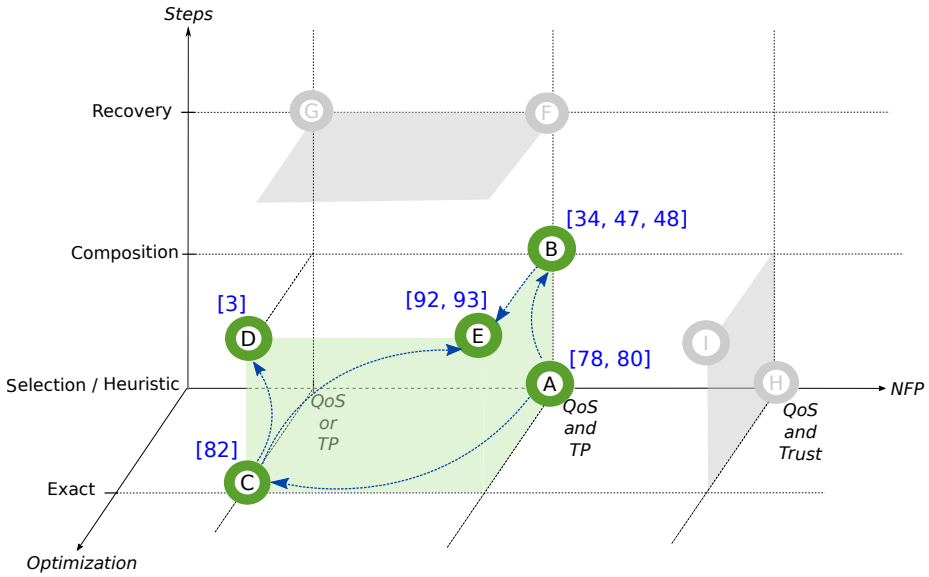


Figure 3.2: Contributions to Efficiency and Consistency in Service Selection and Composition

Positioning of contributions.

Being interested in understanding precisely the foundations of service-oriented systems, we considered service selection and composition as a central concept in this set of contributions. Our goal was to investigate various quality attributes and optimization techniques and our contributions reflects exploration of efficiency and consistency challenges, aligning closely with the major research trajectories observed in the field over the past two decades.

Our early contributions, El Haddad et al. [80], El Haddad et al. [78] (Section 3.3.1), focused on heuristic approaches for service selection that jointly considered quantitative (QoS) and qualitative (transactional) attributes, situating our work alongside other heuristic and metaheuristic efforts in the literature. In El Haddad et al. [80], we introduced an approach that integrates transactional and QoS-based service selection, deriving the transactional behavior of a composite service from its components. Users specify transactional requirements in terms of risk (acceptance or compensation of results) and QoS preferences through weights assigned to criteria. In a subsequent study, El Haddad et al. [78], we extended this approach to support compositions of both elementary and composite services. Our algorithm ensures that each selected component is locally optimal in terms of QoS while meeting global transactional requirements. Unlike methods proposed by Bhiri et al. [30], Montagut et al. [177], and Maamar et al. [161], our approach avoids component-level termination checks, thereby improving scalability.

Building on this foundation, subsequent contributions expanded in two parallel directions: heuristic approaches for composition phase considering both attribute types (Blanco et al. [34], Cardinale et al. [47, 48])(Section 3.3.2), and exact methods for selection focused on quantitative QoS attributes (El Haddad et al. [82])(Section 3.4.1).

In Cardinale et al. [47], we addressed the composition problem by combining search meta-heuristics and Coloured Petri Nets (CPN)to satisfy both functional and transactional requirements. Functional needs were defined by input/output attributes, while transactional needs were specified as risk levels. By extending CPN models to include transactional properties and adapting Petri Net unfolding for best-first search, we ensured that compositions met both functional and risk-level requirements, with the search terminating upon reaching the desired state. In Cardinale et al. [48], we further developed this approach, proposing a two-stage strategy that first identify execution paths meeting user's transactional requirements, then select the path that best satisfied QoS constraints through local optimisation. Separately, in Blanco et al. [34], we introduced a utility function that integrated functional capabilities, QoS, and transactional attributes, enabling a ranking of compositions based on user criteria. User requests specified functional requirements via input/output attributes, non-functional constraints via QoS values, and

transactional requirements as risk levels. This work differs from that of Cardinale et al. [48] in two key ways: component selection was guided by the utility function rather than transactional filtering, and it employed a global optimization approach using an A*-based heuristic. Additionally, we presented PT-SAM-Transac, a service composer that adapts the Petri net deployment algorithm to be guided by the utility function, efficiently identifying compositions while exploring only a small part of the search space. The integration of transactional properties into the selection and composition phases in all our above contributions addressed a gap in earlier QoS-centric research, contributing to the field's gradual convergence toward solutions that balance both efficiency and consistency.

In parallel, our contributions shifted towards exact optimization approaches for the selection problem, focusing initially on quantitative QoS attributes, along with efforts by Ardagna et al. [16], Bonatti and Festa [35], and Zeng et al. [257], who similarly framed the problem as a combinatorial optimization challenge. In El Haddad et al. [82], we introduced the concept of *fairness* to ensure equity among users in the execution plans. Zeng et al. [257] addressed multiple usage scenarios (*i.e.*, execution paths) by optimizing each execution path independently and merging them into a single plan, selecting for each task the service from the most frequently executed path, called the *hot path*. However, this approach can lead to globally suboptimal solutions, as we will illustrate in Section 3.4.1. To overcome this limitation, we introduced the *maximum regret* criterion to minimize the worst-case deviation from optimal solutions across all scenarios. We defined an equitable solution as one that is optimal in terms of maximum regret and formulated the problem of finding such a fair execution plan as a Linear Integer Programming model.

Having achieved effective exact approach for the selection phase, we advanced further to develop exact method for composition phase addressing solely quantitative QoS attributes in Abu-Khzam et al. [3] (Section 3.5), placing the contribution firmly within the domain of exact modeling techniques, such as those explored by Schuller et al. [202, 203], and Trummer et al. [221]. In this work, we examined the computational complexity of the QoS-aware service composition problem in complex workflow patterns. Addressing a long-standing open question, we provided a formal proof of the problem NP-hardness. Our analysis revealed that complexity depends on workflow structure, the number of tasks, available alternative services, and types of QoS criteria. We showed that the problem is solvable in polynomial time when only one QoS criterion exists per task, unless tasks are accessible via multiple transitions within the workflow, which makes it strongly NP-hard. For a fixed number of criteria, we demonstrated the existence of a pseudo-polynomial time algorithm, suggesting that the problem may be less computationally challenging in practice than previously assumed.

Drawing upon our experience with exact methods, and with handling quantitative and qualitative attributes, we subsequently proposed an exact-based composition approach capable of managing both types of quality attributes simultaneously (Gamez et al. [92, 93])(Section 3.4.2). Perhaps most significantly, this contribution represent an innovative step in the state-of-the-art by proposing an exact-based composition approach capable of simultaneously managing both quantitative and qualitative attributes. This positions the contribution as a bridge between earlier heuristic integration of transactional properties and exact approaches, filling a critical gap in methods that often consider either QoS or transactional attributes in isolation. Additionally, in all our previous contributions, services were treated as black boxes, assuming at least one service met transactional needs for each task. To overcome this limitation, we adopted a Software Product Line approach, grouping services with identical functionality into families modeled using Feature Models (Lee et al. [136]). These models represent variable operations as optional features and common operations as mandatory, with constraints capturing dependencies and non-functional properties. This enabled us to identify valid service configurations (*i.e.*, optimal combination of services) whose aggregated properties satisfy global requirements (*i.e.*, user preferences and workflow constraints). In Gamez et al. [92], we focused on transactional properties, while in Gamez et al. [93], we extended this work to include both transactional and QoS properties. The latter approach also accommodates conversation-based services, which involve multiple operations with internal transitions, unlike stateless services with atomic, independent operations. By using cross-tree constraints in feature models, we captured dependencies between individual operations' transactional properties and the aggregate behavior of the entire service, ensuring precise alignment with user preferences. A key advantage of these approaches is the ability to dynamically compose new services from existing operations at runtime, eliminating the need for pre-deployment and relaxing the initial assumption that valid services must be pre-available.

3.3 Heuristic Approaches

3.3.1 Transactional- and QoS-based Service Selection

The content of this section is adapted from El Haddad et al. [78, 80] work carried out in collaboration with colleagues from Universidad Central de Venezuela and Université Paris Dauphine-PSL

Overview

In this line of research, we focused on selecting component services to obtain a transactional composite web service that maximize user satisfaction by meeting both the QoS and transactional requirements specified by the input workflow and user preferences. As represented in Figure 3.3, the inputs of the selection process include a workflow, transactional requirements expressed in terms of risk level, and a set of weights over QoS criteria. Based on the input workflow, the `Composition Manager` searches the service registry for candidate component services for each activity (*i.e.*, task) of the workflow, taking user preferences into account. Using the retrieved candidate services, the `Planner Engine` generates an execution plan that is an assignment of one component service to each task of the input workflow.

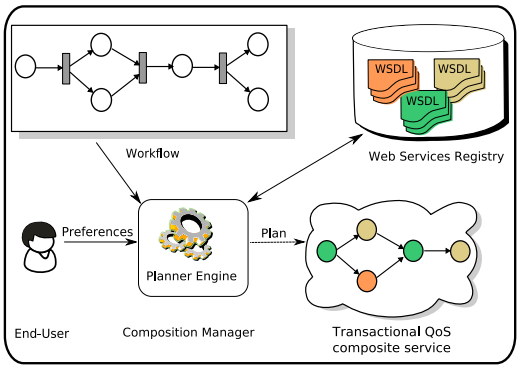


Figure 3.3: System architecture

Definition of risk

To explain the transactional service selection process, it is first necessary to define how users can express their transactional preference. The importance of the uncertainty in execution completion is captured by a criterion called *risk*. In terms of transactional properties, we consider properties a and ar to be riskier than c and cr. Therefore, we defined the following two levels of execution risk in a transactional system:

- ***R0***: the system guarantees that if the execution is successful, the obtained results can be compensated by the user (*i.e.*, the user can execute another application that can semantically undone the previous one).
- ***R1***: the system does not guarantee the result can be compensated (or semantically undone) by the user in case of successful execution.

Aggregation of transactional properties

The selection process assigns services to workflow activities in order: from left to right in sequential patterns and from top to bottom in split patterns. The automaton in Figure 3.4 models all possible transactional composite services generated through this process. State I represents the initial state, and the final states (c,cr,a, and ar) correspond to the transactional properties of a composite web service. When one of these final states is reached, a valid transactional composite service has been successfully constructed. The rules guiding sequential and parallel composition follow those outlined in Chapter 2, Table 2.1.

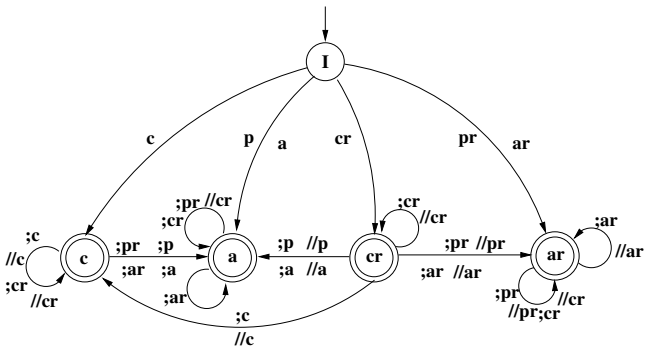


Figure 3.4: Transactional properties aggregation

Aggregation of QoS values

In this line of work, for each activity, a set of transactional services is first selected based on transactional requirements. Subsequently, a QoS-driven selection is performed. To select one service for each activity, we applied a Multiple Criteria Decision Making (MCDM) approach (Zeleny et al. [254]) based on the weights assigned by the user to each quality criterion. We used a Simple Additive Weighting technique to assign a *quality score to each service* as follows:

- Scaling phase : each criterion value $q_j(s_{ik})$ is normalized to $\bar{q}_j(s_{ik})$ that is a scaled value such that $\bar{q}_j(s_{ik}) \in [0, 1]$. Some of the criteria are negative, like execution time and execution price, are scaled negatively (the higher the value, the lower the quality) while other criteria are scaled positively (the higher the value, the higher the quality) as shown in Equation 3.1.

$$\bar{q}_j(s_{ik}) = \begin{cases} \frac{q_j(s_{ik}) - q_j^{min}}{q_j^{max} - q_j^{min}} & \text{if } q_j \text{ is positive and } q_j^{max} - q_j^{min} \neq 0, \\ \frac{q_j^{max} - q_j(s_{ik})}{q_j^{max} - q_j^{min}} & \text{if } q_j \text{ is negative and } q_j^{max} - q_j^{min} \neq 0, \\ 1 & \text{otherwise.} \end{cases} \tag{3.1}$$

- Scoring phase : $Score(s_{ik}) = \sum_j w_j \bar{q}_j(s_{ik})$, where $w_j \in [0, 1]$ is the weight assigned by the user to the quality criterion j such that $\sum_j w_j = 1$ and $\bar{q}_j(s_{ik})$ is the value of criterion j for service s_{ik}.

Based on $Score(s_{ik})$, the service with maximal score is selected to execute the activity a_i. If there are several services with maximal score, one of them is selected randomly.

Algorithm

The TQoS-driven selection algorithm takes as input a workflow WF composed of n activities and outputs a transactional composite web service (TCWS), consisting of elementary or composite web services assigned to each workflow activity. Each web service assignment affects subsequent selections through its transactional property, tracked via state variables. Two risk levels govern the algorithm's behavior: when risk chosen by the user is $R0$, than only compensatable or compensatable retriable web services are assigned to each activity, selecting the service with the best QoS score. When risk chosen by the user is $R1$, the algorithm considers initially all possible transactional properties. Services are iteratively assigned to workflow activities via the `ASSIGN_NEXT` function, which adapts the permissible service set based on the current TCWS state and transactional properties.

Function `ASSIGN_NEXT` analyzes the input workflow, WF, from the current position i. If the ith element of WF is AND-split or XOR-split pattern it calls function `ASSIGN_AND` or `ASSIGN_XOR` respectively. If the ith element of WF is a sequence pattern which is inside an AND-split or a XOR-split pattern, the

function is recursively called after the update of set WS_Set, depending on the state of the current resulting transactional composite web service . Otherwise, the ith element of WF is an activity, then function `ASSIGN_NEXT` assigns a web service to the jth activity of the workflow having the best QoS and variables $State$ and NTP are updated. We use a function `GetBestofQoS(WS_Set, j)` that returns the web service having the best QoS among the set of web services which can execute the jth activity of WF (subset of WS_Set), and a function `GetTPOf(`WS`)` that returns the transactional property of the Web service WS.

A function, `ComputeQoS(`$TCWS$`)`, is implemented to evaluate the QoS of the resulting transactional composite web service from the QoS scores of its component web services .

3.3.2 Transactional- and QoS-based Service Composition

The content of this section is adapted from Blanco et al. [34] and Cardinale et al. [47, 48] work carried out in collaboration with colleagues from Simón Bolívar University and Université Paris Dauphine-PSL

Overview

In this line of work, we proposed a hybrid approach that integrates meta-heuristic search technique to address functional requirements, QoS constraints, and risk levels defining transactional needs. These aspects are collectively considered to generate good service compositions; goodness is evaluated based on the combined degrees of satisfaction with respect to functional, QoS, and transactional criteria.

To this end, we extend the Coloured Petri Net (CPN) formalism to model both QoS and transactional properties within the composition process. Our CPN-Transactional Web Service (CPN-TWS) algorithm employs a Petri net unfolding approach to conduct a Best-First Search, which stops upon reaching a desired marking (*i.e.*, user-specified output attributes) from an initial marking (*i.e.*, user-specified input attributes) in the CPN. The unfolding process is guided by local QoS metrics and aggregated transactional properties, ensuring that the resulting composition satisfies user-defined functional objectives, QoS expectations, and risk requirements.

Our experiments indicate that considering transactional properties during service selection and composition has no significant impact on the execution time of our approach. Theoretically, the algorithm has a complexity of $\mathcal{O}(\text{card}(S)^2)$, where $\text{card}(S)$ denotes the number of services in the registry, a result that is consistent with our experimental observations.

Registry modelling

The service registry is modeled as a *Web Service Dependency Net*(WSDN), following Chapter 2, Definition 2.3.1. It is important to emphasize that the execution of a service s cannot forbid the execution of another service s' that shares the same input attributes; such services are considered independent. However, the execution of a service s may enable the execution of other services by producing the input attributes they require. We provide in Figure 3.5 an example illustrating the construction of a WSDN created from the web services s_1 to s_9.

Our approach

We proposed an automatic approach to resolve the service composition problem. The input of our algorithm is the user query Q and the WSDN, representing dependencies among the services of the registry along with their input and output attributes. The output of our algorithm is, if a solution exists, a CPN denoted $WSDN_{\sigma_Q}$ representing the set of transactional web services selected for the composition. As illustrated in Figure 3.6, our CPN-TWS selection algorithm comprises four steps. Step 1 verifies the admissibility of Q. Step 2 identifies the web services in the registry (*i.e.*, the transitions in the WSDN) that may contribute to producing the outputs of Q by considering the transactional properties (TP) that satisfy the risk level. Step 3 returns a firing sequence σ_Q, containing the component web services of a transactional composite web service selected based on optimal local QoS to satisfy Q. Step 4 returns a CPN, $WSDN_{\sigma_Q}$, built from σ_Q and allowing to execute the resulting transactional composite web service .

WS	Inputs	Outputs	TP
s1	AuthorCod, Inst	PubCod	*p*
s2	AuthorName	PubCod	*p*
s3	PubCod	Title	*p*
s4	PubCod	ConfCod	*p*
s5	PubCod	ConfCod,ConfName	*p*
s6	ConfCod	ConfName,ConfDate	*cr*
s7	Inst	AuthorCod	*p*
s8	ConfCod	ConfPlace	*pr*
s9	AuthorCod	ConfCod	*pr*

Services in the registry
(example from Blanco et al. [23])

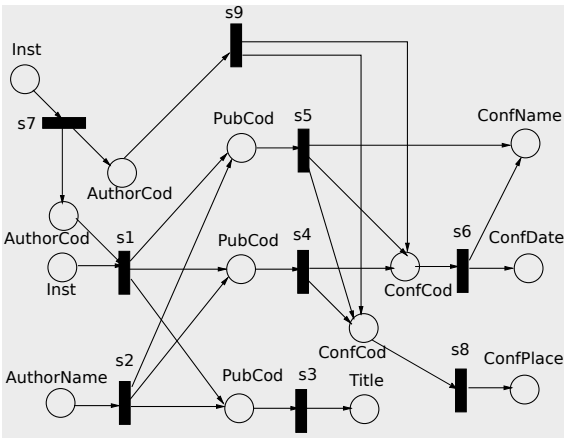


Figure 3.5: The WSDN created from the services in the registry

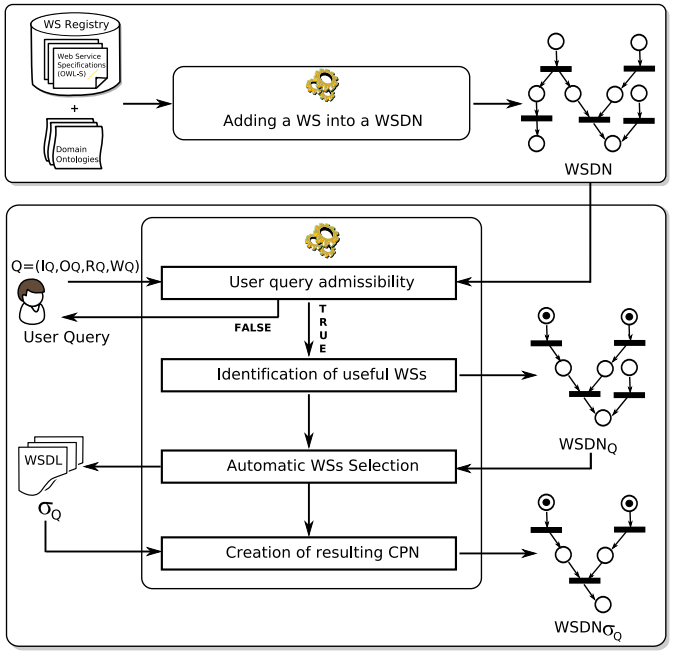


Figure 3.6: CPN-based Transactional- and QoS-based Service Composition Approach

Step1: Query admissibility. A query $Q = (I_Q, O_Q, W_Q, R_Q)$ (see Chapter 2, Definition 2.1.1) is admissible iff the web services registry contains (i) at least one web service whose TP satisfies R_Q and whose input attributes are included in I_Q, and (ii) one or more web services , whose TP satisfies R_Q, allow to produce all output attributes of O_Q. Let S_{R_Q} be the set of transitions whose color satisfies R_Q. In terms of CPN, Q is admissible iff the $WSDN$ contains (i) at least one transition of S_{R_Q} with all its predecessors in I_Q and (ii) one or several transitions of S_{R_Q} such that the union of their successors contains O_Q. If both conditions are satisfied, then the selection algorithm can proceed to the next step. Otherwise, the algorithm can not continue (there is no fireable transition, then no solution satisfying Q can be found). Note that the admissibility of Q does not guarantee that a solution satisfying Q exists. We illustrate this step with the following example.

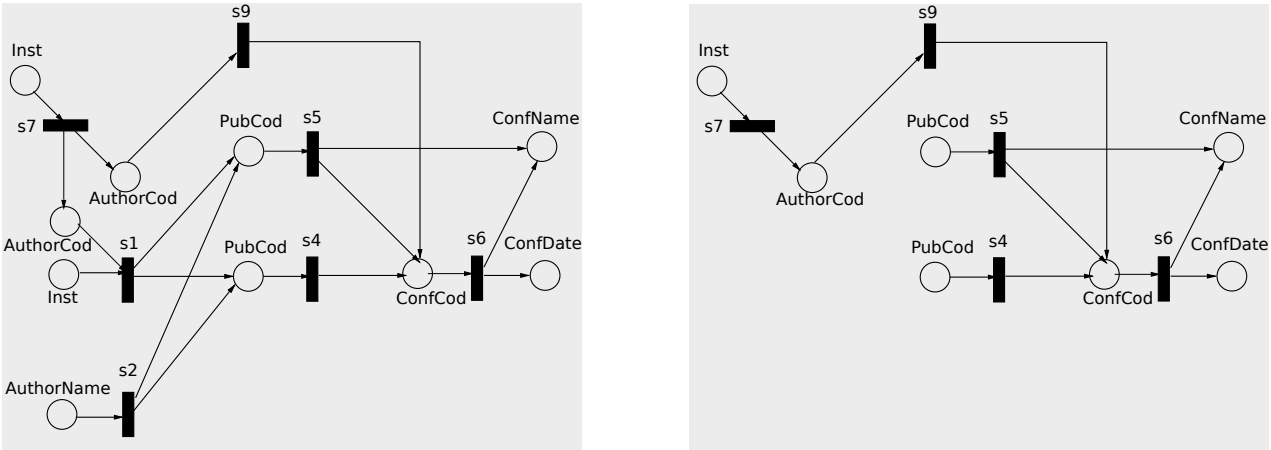
Example 3.3.1. Suppose the registry represented by the WSDN of Figure 3.5 and let $R_Q = R_1$ (*i.e.*, S_{R_Q} contains all the web services of the registry).

✗ If $O_Q = \{ConfName, ConfDate, AuthorName\}$, then Q is not admissible because place $AuthorName$ has no predecessor (*i.e.*, it represents an attribute which can not be produced by any service).

✓ Now suppose $O_Q = \{ConfName, ConfDate\}$ and if $(I_Q \cap \{Inst, AuthorCod, AuthorName, PubCod, ConfCod\}) \neq \emptyset$, then Q is admissible because all places corresponding to the attributes of O_Q has at least one predecessor (*i.e.*, they can be produced and at least one web service can be executed from I_Q).

Step2: Identification of the potentially useful services. To build the set of potentially useful web services , we analyzed the WSDN from the output places corresponding to the output attributes of O_Q and creates a new CPN, called $WSDN_Q$, by recursively adding all the predecessors of the analyzed places or transitions, using a bottom-up marking. If it does not exist at least one input attribute $\in I_Q$ among the input places of $WSDN_Q$ (*i.e.*, the places with no predecessor), then the algorithm returns an error, because no transition is fireable and then no web service can be selected, meaning that Q has no solution. Otherwise, the selection algorithm can proceed to the next step. This step is inspired from the yellow coloring step of SAM algorithm of Brogi et al. [40]. We illustrate this identification step with the following example.

Example 3.3.2. Suppose $I_Q = \{Inst\}$, $O_Q = \{ConfName, ConfDate\}$, $R_Q \in \{R_0, R_1\}$, the WSDN of the registry is represented by Figure 3.5, and all web services are either compensatable or compensatable retriable. In this scenario, the result of this step is the CPN $WSDN_Q$ represented in Figure 3.7(a). It contains predecessors of $ConfName$ and $ConfDate$ places and recursively their predecessors. Transitions s_3 and s_8 of WSDN do not appear in $WSDN_Q$ because they are not predecessors of $ConfName$ and $ConfDate$. When the transactional properties of the web services are those represented in the TP column of Figure 3.5 and $R_Q = R_1$, then only transitions s_7, s_9, s_4, s_5, and s_6 are selected as depicted in Figure 3.7(b).



(a) Considering all services are *c* or *cr*

(b) Considering TPs of Figure 3.5

Figure 3.7: The $WSDN_Q$ resulting from Step 2

Step3: Automatic selection of the component web services. The automatic selection step consists in building a path into the coverability tree of $(WSDN_Q, M_Q)$ (without building all the coverability tree), such that the firing sequence associated with this path corresponds to a transactional composite web service satisfying the user query and requirements. The inputs of this step are the $WSDN_Q$ returned by Step 2, the initial and the final markings (M_Q and M_F), and the user query Q. Its output is, if it exists, a

firing sequence σ_Q, corresponding to the transactional composite web service whose components are the web services of the registry required to evaluate Q and, satisfying all user requirements.

Example 3.3.3. Let suppose that $I_Q = \{Inst\}$ and $O_Q = \{ConfName, ConfDate\}$. Depending on the color of the transitions, the following firing sequences (see Chapter 2, Definition 2.3.7) can be returned: $\{s_7, s_1, s_4, s_6\}$; $\{s_7, s_1, s_5, s_6\}$; $\{s_7, s_1, s_4, s_5, s_6\}$; $\{s_7, s_1, s_5, s_6\}$; $\{s_7, s_9, s_6\}$; $\{s_7, s_1, s_9, s_6\}$; $\{s_7, s_9, s_1, s_6\}$. Note that if s_4 is fired, then s_9 is cut-off and if s_9 is fired, then s_4 is cut-off (because s_4 and s_9 produce the same output attributes). If s_5 is fired then s_4 and s_9 are cut-off (see Chapter 2, Definition 2.3.8) ((because $(s_4)^\bullet \subset (s_5)^\bullet$ and $(s_9)^\bullet \subset (s_5)^\bullet$). If s_4 and s_6 are fired or if s_9 and s_6 are fired then s_5 is cut-off (because $(s_5)^\bullet \subset [(s_i)^\bullet \cup (s_6)^\bullet)$ with $i \in \{4, 9\}$).

When several transitions are fireable, to select which transition has to be fired, we proposed a quality measure of a transition s which depends on the user query Q such that:

Definition 3.3.1. The quality of a transition $s_i \in S$, called $Quality_Q(s_i)$, depends on the user query Q and is defined as:

$$Quality_Q(s_i) = Score(s_i) \times g(C_S(s_i)) \times (card(O_Q \cap s_i^\bullet) + 1) \times \left(1 + \frac{card((s_i^\bullet)^\bullet)}{card(S)}\right)$$

with $Score(s_i) = \sum_j w_j \times q_j(s_i)$ with $(w_j, q_j) \in W_Q$ and $q_j(s_i)$ the value of the QoS criterion q_j for the service corresponding to s_i, and with $g : \sum_S \to \mathbb{N}$, a function such that: $g(p) = g(a) < g(pr) = g(ar) < g(c) < g(cr)$.

▷ Value $Score(s_i)$ allows to evaluate the QoS of the service corresponding to a transition s (the higher the score, the better WS QoS).

▷ Function g allows to select a transition whose transactional property is the less restrictive. An example of g could be: $g(p) = g(a) = 1$, $g(pr) = g(ar) = 2$, $g(c) = 3$, and $g(cr) = 4$.

▷ $(card(O_Q \cap s_i^\bullet) + 1)$ gives more chance to select transitions,producing more required outputs.

▷ $\left(1 + \frac{card((s_i^\bullet)^\bullet)+1)}{card(S)}\right)$ increases the quality to those transitions which will allow more transitions to be fireable.

Step4: Creation of the resulting CPN. Suppose that Step 3 returns the firing sequence $\sigma_Q = \{s_7, s_1, s_4, s_5, s_6\}$. σ_Q contains a useless transition, s_4. Indeed, $[(s_4^\bullet)^\bullet \subset (s_5^\bullet)^\bullet]$ and $M_Q \xrightarrow{\varsigma} M_F$, with $\varsigma = \sigma_Q - \{s_4\}$, however s_4 has been selected before s_5, therefore it was not considered as cut-off. As a consequence, we add another step to our algorithm, in order to eliminate potentially useless transitions of the resulting firing sequence. Moreover, if Step 3 produces the firing sequence $\sigma_Q = \{s_7, s_1, s_9, s_6\}$, this last step will eliminate transition s_1 because in σ_Q there is no sequence of transitions starting from s_1 and leading to the output places of O_Q.

In this sense, the last step of the CPN-TWS algorithm consists in cleaning the firing sequence result of Step 3 by deleting useless transitions from σ_Q. Note that the resulting Colored Petri Net, $WSDN_{\sigma'_Q}$, created from σ'_Q by Step 4, is used to execute the resulting transactional composite web service .

3.4 Exact Approaches

3.4.1 QoS-based Service Selection

The content of this section is adapted from El Haddad et al. [82] work carried out in collaboration with a colleague from Sorbonne Université

Overview

Returning to our motivating example in Figure 3.1, users may follow different executions paths: execution path A for those booking a plane ticket using (T1,T2), execution path B for those booking a train ticket

using (T1,T3), and execution path C for those booking a car seat using (T1,T4). An execution plan (*i.e.*, a composition) consists of selecting one service for each task. Since multiple execution plans are possible, to differentiate them, QoS attributes are used. For a given execution path, the QoS of the resulting composite service (corresponding to a specific execution plan) for each criterion is computed using an aggregation rule specific to that criterion (see Chapter 2, Section 2.3), and the resulting QoS vector is then transformed into a single quality indicator through Simple Additive Weighting technique (see Section 3.3.1).

However, determining the execution plan that optimizes the overall quality indicator is a combinatorial problem.Moreover, even for the same execution plan, QoS values may vary depending on the specific execution path taken by the end user. To solve this, in their work, Zeng et al. [257] proposed a method to handle the multiplicity of execution paths as follows:

- Separate solutions for each execution path: independently determining the optimal selection for each of the execution paths, then
- Merge selections: merging the selections into a single execution plan. To do this, the authors assign to each task the service chosen in the execution path corresponding to the *hot path* for that task. In other words, if a task T_i belongs to several execution paths, the service selected is that of the most frequently used path.

The *hot path* of a task is the execution path most often used when executing that task. However, this approach can lead to unsatisfactory selections overall, as we will illustrate in the following example.

Example 3.4.1. For simplicity, consider only two paths: A (T1,T2) and B (T1,T3). Suppose that 49% of users have chosen execution path A (T1,T2), while 51% of them execution path B (T1,T3). Consider user preferences expressed as weights for response time and execution cost are 0.7 and 0.3 respectively. Applying the *hot path* approach leads to the following:

- Separate solutions for each execution path: the optimal solution for each task is computed using Simple Additive Weighting technique.
  - For a user taking execution path A, $q_1^{max} = 13$, $q_1^{min} = 2$, $q_2^{max} = 20$, $q_2^{min} = 10$. Then services s_{12} and s_{23} will be selected respectively for tasks T1 and T2 of path (T1,T2) with an overall quality score of 0.83.
  - For a user taking execution path B, $q_1^{max} = 11$, $q_1^{min} = 6$, $q_2^{max} = 20$, $q_2^{min} = 18$. The optimal solution is s_{11} and s_{32} with an overall quality score of 0.82. Then services s_{11} and s_{32} will be selected respectively for tasks T1 and T3 of path (T1,T3) with an overall quality score of 0.82
- Merge selections: for the execution plan (T1,T2,T3), for tasks T2 and T3, services s_{23} and s_{32} will be selected respectively. While for task T1, the service s_{11} is the one that will be selected because the task appears in both paths and path B is the most chosen one (*hot path*).

Now let us see how the use of the hot path will lead to unsatisfactory users :

Example 3.4.2. Back to our example, we have:

- For users taking execution path A, with the use of hot path, they have to execute s_{11} and s_{23}. Such solution has an overall quality score of 0.53. This means a *regret* score for such users of 0.3.
- For users taking execution path B, with the use of hot path, they have to execute s_{11} and s_{32}. This means a *regret* score for such users of 0.

Fairness in service composition

To overcome the above drawback, we explicitly took into account the different possible usage scenarios (*i.e.*, execution paths). To do this, we drawed inspiration from work carried out in robust optimisation (Kouvelis et al. [131]), this latter notion being formally close to the notion of fairness. In particular, we aimed to minimise the maximum regret criterion, traditionally used to measure the robustness of a solution. In the context of service composition, fairness (Moulin [178]) will refer to the concern not to one end-user with respect to another in the execution plan used to perform the various tasks. In this work, we were the first as long as we know to introduce this new attribute and a notion of equity in service composition by proposing to use the fairness to minimize the max regret of a user.

Overall, the objective of our work was the simultaneous consideration of multiple execution paths. The aim was to maximise the satisfaction of the user who has suffered most. In El Haddad et al. [82], we proposed an approach for computing *fair* optimal solution in the sense of this criterion. We gave a formulation of the problem of such a fair execution plan with a Linear Integer Program as described below.

Linear Program formulation

Our linear program in mixed variables for two execution paths A, B is presented as follows :

$$
\begin{array}{rcll}
 & & \min\max\{R_A, R_B\} & \\
R_A & = & \mathrm{QoS}^*_A - \mathrm{QoS}_A & (\text{regret } A) \\
R_B & = & \mathrm{QoS}^*_B - \mathrm{QoS}_B & (\text{regret } B) \\
\sum_{i \in S_j} y_{ij} & = & 1 \quad \forall j \in A \cup B & (\text{one web service per task}) \\
\sum_{i \in S_j} p_{ij} y_{ij} & = & p_j \quad \forall j \in A \cup B & (\text{duration of a task } j) \\
x_{k,A} - (p_j + x_{j,A}) & \geq & 0 \quad \forall t_j \rightarrow t_k \ (j,k \in A) & (\text{precedences } A) \\
x_{k,B} - (p_j + x_{j,B}) & \geq & 0 \quad \forall t_j \rightarrow t_k \ (j,k \in B) & (\text{precedences } B) \\
Q_{2,A} & \geq & x_{j,A} + p_j \quad \forall j \in A & (\text{duration of } A) \\
Q_{2,B} & \geq & x_{j,B} + p_j \quad \forall j \in B & (\text{duration of } B) \\
Q_{1,A} & = & \sum_{j \in A} \sum_{i \in S_j} c_{ij} y_{ij} & (\text{cost of } A) \\
Q_{2,B} & = & \sum_{j \in B} \sum_{i \in S_j} c_{ij} y_{ij} & (\text{cost of } B) \\
 & & x_{j,A}, x_{j,B} \geq 0, \ y_{ij} \in \{0,1\} &
\end{array}
$$

3.4.2 Transactional- and QoS-based Service Composition

The content of this section is adapted from Gamez et al. [92, 93] work carried out in collaboration with colleagues from Universidad de Málaga

Overview

Our purpose in this line of work was to assist users in integrating on the fly the operations of services to realize their required tasks by further meeting their transactional and QoS preferences. Towards this purpose, we proposed a Software Product Line based approach for conversation-based service selection with transactional and QoS support. Unlike classical transactional-aware selection approaches where if a requested service with some transactional property is not deployed, then no solution is possible, our approach only need to ensure that there is at least one configuration of a service family that could be generated satisfying the required transactional property. Figure 3.8 depicts our approach that comprise three steps.

Step 1: Workflow Feature Model Generation and Specialization

We automatically generate the feature model from the requirements model by mapping workflow specifications to feature model elements (see Figure 3.8). Each workflow task becomes a mandatory feature,

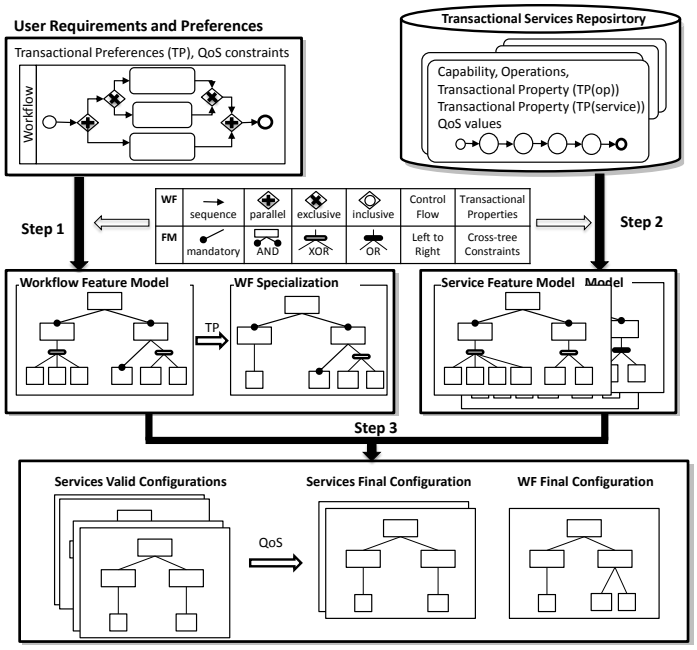


Figure 3.8: SPL-TQSSS Approach

while parallel, exclusive, and inclusive gateways correspond to AND, XOR, and OR feature relationships, respectively. The mapping also preserves the control flow of the workflow from left to right within the feature model. Additionally, transactional rules (see Chapter 3, Table 2.1) are mapped as cross-tree constraints that are used to specialize the *Workflow Feature Model* by removing the features that do not satisfy the transactional preferences. We use the transactional rules to automatically generate *Cross-tree constraints* that relate the user preferences with the transactional property of each task. These constraints together with the relationships between the features in the diagram are used by Hydra[1] to automatically generate the *Workflow Specialization*. This *WF Specialization*, shown in Figure 3.9, represents the set of valid configurations of the workflow feature model, capturing possible transactional properties for each task in accordance with the overall transactional requirements.

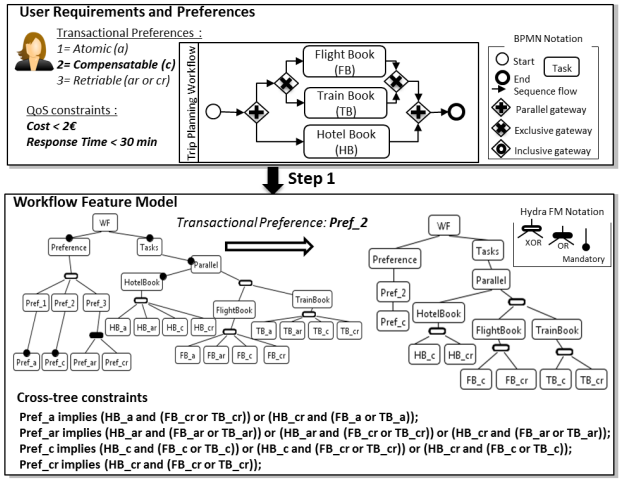


Figure 3.9: Workflow Feature Model Generation and Specialization

[1]Hydra is a tool for feature modelling developed by Caosd Research Group (https://caosd.lcc.uma.es/hydra/)

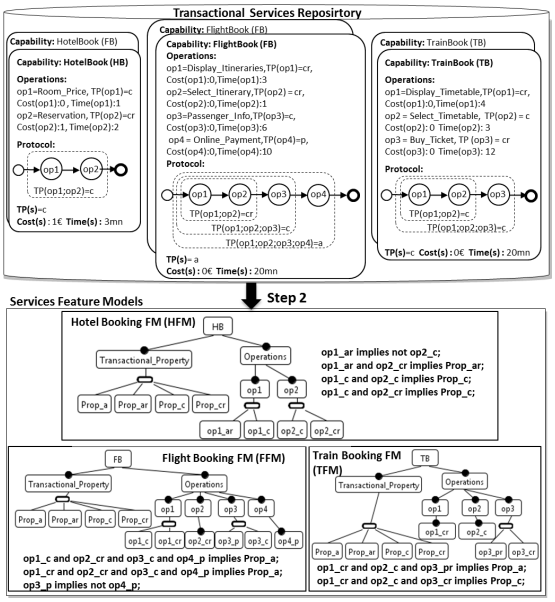


Figure 3.10: Services Feature Model Generation

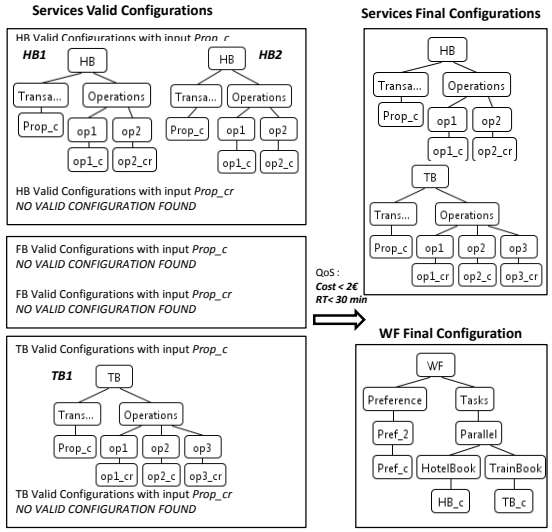


Figure 3.11: Services Valid Configurations and Final Configurations

Step 2: Services Feature Model Generation

The discovery process identifies, for each workflow task, the services in the repository that provide the required functionality. In this step, services fulfilling the same task and offering similar functionality are grouped into service families and mapped to feature models. This mapping is performed automatically using the same principles defined in the first step, but now modeling the internal operations of services instead of workflow tasks. Then, using SPL refactoring techniques (Alves et al. [12]), the feature model that contains all the service configurations belonging to a family is generated. As we did with the workflow feature model, at the beggining we assume that the transactional property of this service family could be whichever of the four possibilities. Then, following the transactional rules of Table 2.1, we automatically generate the cross-tree constraints. As a result, for each task, we obtain a *Service Feature Model*, as shown in Figure 3.10, representing the service family associated to it. This family generation is required only during the initial discovery. For next discoveries, the same feature model is reused. If new services with equivalent capabilities are deployed or removed from the repository, the service family is dynamically updated by adding or removing features corresponding to the internal operation alternatives. Note, that a service feature model represents not only the currently deployed services in the repository but also any combinations of operations that might not correspond to any deployed service at present.

Step 3: Service Valid Configuration Generation and Selection

As this stage, we have generated the *Services Feature Models* and the *WF Specialization* taking into account the user transactional preference. Then, for every task of that specialization, Hydra automatically obtains the *Service Valid Configurations* that satisfy the preferences selected in the specialization.

For each task, a *Service Valid Configuration* is selected that satisfies the required transactional behavior, services within the family whose operations comply with the transactional properties of the task. To choose among all valid configurations for each service family, we apply QoS constraints. Firstly, we consider the generated service valid configurations that have correspondence with deployed services. Among these, we select service configurations that yields the highest QoS score. If no deployed service matches the required configuration or fails to meet the QoS constraints, we dynamically generate a new service on-the-fly by composing the necessary operations from the valid service configuration. With this process, we select one *Service Final Configuration* for each workflow task, resulting in one *WF Final Configuration*.

Then, we perform optimal services selection from the service valid configurations using the Simple Additive Weighting method, which involves two phases: scaling to normalize cost and response time

values of each service s between 0 and 1, and weighting to compute a score for each service as follows: $score(s) = w_1 \times \frac{cost^{max} - cost}{cost^{max} - cost^{min}} + w_2 \times \frac{responsetime^{max} - responsetime}{reponsetime^{max} - responetime^{min}}$ where $w_j \in [0, 1]$ and $\sum_{j=1}^{2} w_j = 1$ are weights for cost and response time. If the values of these weights are not expressed by the user preferences then we consider $w_1 = w_2 = 0.5$. $cost$ and $responsetime$ refer to the scaled values for service s; $cost^{max}$ and $cost^{min}$ (resp. $reponsetime^{max}$ and $responsetime^{min}$) are the maximum and the minimum values across all valid configurations within a family.

Then, for the last *WF Specialization*, we generate all possible compositions between services associated to service valid configurations and the composition which maximizes the score function while satisfying the QoS constraints is selected. If there is more than one composition that has the same maximal score value, then a composition will be selected from them randomly. If no composition satisfies the user QoS constraints, then we generate a proper composition by generating on-the-fly suitable services from the service valid configurations.

As the last *WF Specialization* did not have more possible choices, so this is the *WF Final Configuration* of Figure 3.11 where the two tasks are compensatable, as well as the two selected services and their associated service valid configurations. In other case, if more than one choice would be still pending in the last *WF Specialization*, then we would make the selection attending to the QoS constraints as we have done with the choice between the two valid configurations *HB1* and *HB2* of the hotel booking service using the method described above. The score for each of these choices is then computed as the sum of the scores of its selected services. The choice with the best score is finally chosen.

After all these model transformations we have obtained automatically the *WF Final Configuration* and the best *Services Final Configurations* that satisfy the user functional and non-functional requirements.

3.5 A Glimpse on Further Contributions

Complexity of QoS Service Selection Problem

The content of this section is about Abu-Khzam et al. [3] work carried out in collaboration with colleagues from Lebanese American University and Université Paris Dauphine-PSL

In this work, we addressed the QoS-aware service selection problem by considering complex workflow patterns. More specifically, we focused on the computational complexity of the problem and investigates workflow structures where the problem is solvable efficiently in polynomial time. The NP-hardness of the problem, under various settings, had been an open problem for many years and had never been addressed thoroughly. We studied the problem complexity depending on the workflow structure, the number of workflow tasks, the number of alternative services per task and the types of quality of service criterion associated to services. We provided for the first time the NP-hardness proof of the problem. Additionally, we showed that the problem is solvable in polynomial time in case of only one criterion per task unless tasks can be reached via multiple transitions in a workflow in which case we showed that the problem is strongly NP-hard. Most importantly, we proved that the problem is solvable by a pseudo-polynomial time algorithm if there is a fixed number of criteria. Our findings reveal that, in most practical settings, the problem is not as hard as claimed.

Exact approach for Transactional-QoS-aware Stateful Service Selection and Composition

The content of this section is about the master thesis work of Massimoliano Drezza [73] from Politenico Di Milano, that I have supervised alone.

In this work, we proposed an algorithm to improve the efficiency of QoS-aware web service selection using Integer Programming (IP). Our approach is built upon models from Ardagna and Pernici [16] and Ben Mokhtar et al. [176], integrating them into a unified framework for solving IP problems. In our proposed model, we used Finite State Automata (FSA) to formally represent both web services and user requirements, capturing conversation-based service behavior in a structured way. Our model considered three QoS dimensions commonly used in the literature: two negative (execution price and latency) and one positive (availability). The selection process is structured into four main steps. The discovery step based on semantic distance, this step identifies web services with potentially matching capabilities. The second step is preliminary selection directly inspired by the work of Ben Mokhtar et al. [176]. This step

checked whether the service control flow (expressed via FSA) aligns with the user's request FSA, yielding a refined list of candidates. the third step is a transformation, it restructured services by aggregating capabilities along valid execution paths. Each service is then represented as a set of aggregated capabilities with associated QoS values. The last step is IP Resolution. Using the simplified service model, IP was applied to assign the best possible service capabilities to each user task, according to user-defined preferences, weights, and constraints. This result in an optimal execution plan, that we denoted EPOPT, that maps each capability in the user's request to a service invocation that maximizes QoS while satisfying all constraints. EPOPT is derived by assigning binary values to decision variables in the IP formulation. Experimental evaluation demonstrates that our approach achieves optimal solutions more efficiently than traditional methods. Tests using IBM ILOG CPLEX confirm the algorithm's speed, scalability, and effectiveness in solving the QoS-aware service selection problem.

Exact approach for QoS-aware Cloud Service Selection

The content of this section is about the master thesis work of Alice Di Luise [159] from Universitá Degli Studi Di Roma Tor Vergata, that I have supervised alone.

In this work, we addressed the QoS-aware service selection problem in a Cloud environment. Unlike traditional SOA-based systems, Cloud-based applications must account for IaaS, PaaS, and SaaS resources. As a result, a service candidate is not a monolithic unit with a single QoS profile but often a composite of multiple components, each with its own QoS attributes (Ye et al. [245]and [246]). We observed that some approaches rely on user logs (Sun et al. [214], Wang et al. [232], Zheng et al. [269]), while others use SLA or provider published data (He et al. [114], Zhang et al. [261]). Our method integrated both sources: logs that provide realistic and dynamic insights (*e.g.*, user-dependent variations in response time), and published data that offer consistent fallback information when logs are unavailable or sparse. To solve the selection problem, we proposed an exact solution via Integer Programming, preceded by a service space reduction phase. This phase incorporated entropy- and hyper-entropy-based filtering (Sun et al. [214], Wang et al. [232]), as well as a similarity computation module inspired by Zheng et al. [269] used to eliminate low-performing candidates rather than to rank them. The input was a workflow of abstract tasks, each representing a required functionality within a composite application. For each task, there exists a set of candidate services (SOA or SaaS), and only one must be selected. In the Cloud context, resource selection extends to IaaS and PaaS components such as virtual machines (VMs) and databases (DBs). Users define constraints and weights over QoS attributes, including response time, throughput, availability, price, number of vCPUs, memory, and storage. Our approach combined log-derived and provider-published QoS data. When separate selections of VM, DB, and SaaS services were infeasible, the system searched for integrated service offerings. The IP model then identified an optimal or near-optimal assignment of services that satisfies workflow structure (sequential, parallel, conditional, loop), user-defined constraints, and weighted QoS preferences. We implemented and evaluated our approach on an EC2 instance. Experimental results show that the proposed space reduction techniques significantly improve the IP solver's response time.

3.6 Summary

In this chapter, we addressed service selection and composition by integrating quantitative QoS criteria and qualitative transactional requirements. After reviewing related work, we proposed heuristic and exact methods to ensure consistent and efficient composite services. The heuristic approaches formalize transactional properties and user preferences, enabling local selection strategies that satisfy both QoS and transactional constraints. For end-to-end composition, we used metaheuristic techniques based on coloured Petri nets and developed utility-based models to rank candidate compositions. On the exact methods side, we modeled service selection as a combinatorial optimization problem, introducing a fairness objective through a maximum regret criterion. Finally, we leveraged Software Product Line engineering and Feature Models to support dynamic, conversation-based service composition. The proposed contributions help bridge the gap between performance optimization and transactional consistency in service composition.

CHAPTER 4

Reliability in Service Composition Execution

In this chapter, we tackle the issue of ensuring reliable execution of service compositions in the presence of failures. Achieving reliability involves utilizing transactional mechanisms that support failure recovery either by restoring the composition to its pre-failure state, or by transitioning it to a state from which normal execution can continue. The recovery behaviour of a composite service is determined by the transactional properties of its component services, which must undertake recovery actions when failures occur. As such, we first proposed an approach based on forward recovery that replaces faulty services, and if replacement is not feasible, backward recovery is applied using compensation. Next, we proposed an approach based on semantic and checkpointing recovery that relaxes the notion of atomicity. In this approach, a failure results in the return of a snapshot of the most advanced partial state, allowing users to accept or reject intermediate results. This chapter highlights the contributions presented in Cardinale et al. [50, 51, 52]. We conclude the chapter with a brief overview of additional research work on concurrency control in service composition execution, presented in El Haddad et al. [81], that we had the opportunity to study.

4.1 Motivations

Reliability is another cornerstone of successful service composition. Composite services are typically constructed from distributed components that run on heterogeneous systems, communicate over the Internet, and are managed by independent and autonomous providers. Due to this distribution and the inherently unreliable nature of the Internet, failures are expected during the execution of component services (Liu et al. [146]). These failures may arise from service unavailability, execution errors, corrupted outputs, or violations of predefined Service Level Agreements (SLAs). The failure of a single component can potentially compromise the correctness of the entire composition, making failure recovery an important issue of reliable service-oriented systems, and the central focus of this chapter. The goal behind the recovery is to enable the composite execution continuity in case a component service becomes unavailable at runtime.

In service compositions, failure recovery mechanisms are used to guide the resulting faulty execution towards a recovery state that is either an acceptable termination state from an end-user viewpoint, or a state from which the execution can be continued normally. Research on failure recovery for composite web services has been an active area focused on two main recovery strategies (Tartanoglu et al. [218]): *backward error recovery*, and *forward error recovery*. The former involves undoing the effects of previously completed services, often through compensation, when a failure occurs later in the composition. Backward recovery restores consistency by undoing the effects of previously completed services, typically through compensation. It aligns with transactional models, particularly those following ACID principles, where rollback ensures atomicity and integrity. Forward recovery, in contrast, handles faults by guiding the system to a new valid state without reversing past actions. This approach relies on exception handling techniques such as retries, services substitution, or proposing alternatives to allow continued execution.

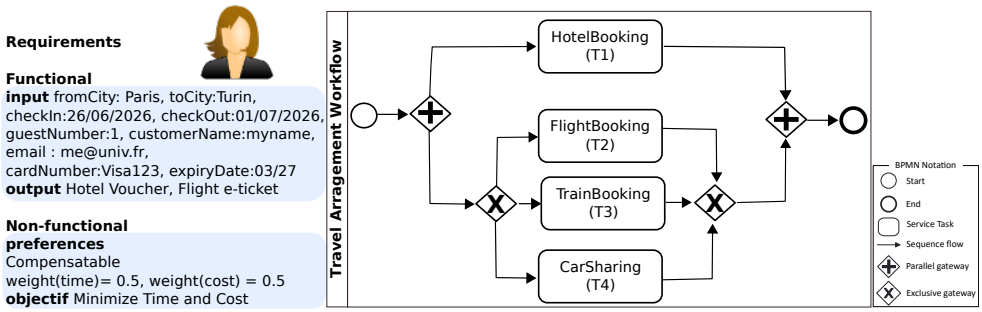


Figure 4.1: TravelArrangement booking motivating example

Table 4.1: Available candidate component services for TravelArrangement

Task	Service	Response Time	Execution Cost	Transactional Property	Provider
HotelBooking	s_{11}	$15mn$	1€	–	HB1
(T1)	s_{12}	$10mn$	3€	*compensatable*	HB2
FlightBooking	s_{21}	$20mn$	1€	–	FB1
(T2)	s_{22}	$10mn$	10€	*compensatable*	FB2
	s_{23}	$8mn$	5€	*retriable*	FB3
TrainBooking	s_{31}	$20mn$	5€	–	TB1
(T3)	s_{32}	$18mn$	8€	–	TB2
CarSharing	s_{41}	$5mn$	2€	*compensatable*	CS1
(T4)	s_{42}	$12mn$	15€	–	CS2

To illustrate the need for failure recovery mechanisms, consider the following scenario based on the previously introduced TravelArrangement composite service. Note that in this chapter, we consider only services with transactional properties.

Motivating example. Suppose that our researcher requests the cheapest combination of hotel and flight reservation services for a business trip to Turin. TravelArrangement then initiates the booking with services s_{12} and s_{23} as the optimal composite service as they offer the lowest cost of 8€ and an execution time of 10 minutes. Suppose that the hotel booking service s_{12} completes successfully, but the flight booking s_{23} fails. The composite service cannot be fulfilled entirely, and two main recovery strategies can be applied:

- Backward recovery by restoring the state back: The hotel booking s_{12} is cancelled using compensation. The researcher is then free to restart the reservation from the beginning as no booking has been kept.
- Forward recovery: TravelArrangement may retry with either the flight booking service s_{23} or with a substitute service such as service s_{22}. If the second booking attempt succeeds, then the booking proceeds as planned. If not, TravelArrangement can propose alternative transportation service such as the car sharing service s_{42} to maintain a valid and useful result.

In the previous Chapter 3, we presented our contributions to the selection and composition of transactional services, enabling compositions to adhere to strict atomicity, adhering to the classical "all-or-nothing" semantics, where either the entire composite service completes or none of its effects are visible. This model is particularly effective when rollback is feasible and consistent state restoration is guaranteed (*i.e.*, via backward recovery). However, backward recovery prevents users from receiving any partial results, while forward recovery may lead to long delays due to failure repair. In many cases, partial responses may still hold value, and strict atomicity can be impractical such as in irreversible actions (*e.g.*,

sending confirmation emails) or when users have varying requirements and can accept different termination states of component services. To address these limitations, this chapter goes beyond "all-or-nothing" semantics and introduces the concept of *relaxed atomicity*, which softens the "all-or-nothing" principle by allowing partial execution outcomes, provided they preserve semantic correctness and meet user-defined acceptance criteria. This is supported by advanced recovery strategies that adapt transactional principles to more flexible and fault-tolerant execution models.

We now introduce the recovery concepts underlying composite service execution, which form the basis of the contributions presented in this chapter. Following Liu et al. [145], the lifecycle of a component service is divided into two phases: an active phase, during execution, and a completed phase, which begins after execution ends. A failure of the service can occur during its active phase, but recovery may be required in either phase. Depending on the service's phase, different recovery strategies can be applied to preserve *relaxed atomicity*:

- *Backward recovery*: it consists in restoring the state that the composition had at the beginning of the composite service execution (*i.e.*, all the effects produced by the failed service are semantically undone by roll-back, and the effects of previously executed services before the failure are semantically undone by compensation). All transactional properties (p, a, c, pr, ar, and cr) allow backward recovery. Backward recovery for a composite service implies relax atomicity.

- *Forward recovery*: it consists in repairing the failure to allow the failed service to continue its execution. Retriable properties (pr, ar, and cr) allow forward recovery. Retry and substitution are some other techniques used to provide forward recovery. Forward recovery relies on exception handling mechanism.

- *Semantic recovery*: after the successful end of a service composition execution, a semantic recovery consists in reaching a state, which is semantically closed to the state the system had before the composition execution. Only compensatable transactional properties (c and cr) allow semantic recovery, relaxing therefore the atomicity. This is also a backward recovery.

- *Checkpointing*: it consists of continuing the execution of the part of the composite service not affected by a failure, while delaying the execution of the affected part. When a service fails, if there is no possibility to repair the failure by forward recovery, instead of executing backward recovery, the current execution state (snapshot) of the composite service is saved and the execution flow of the composite can be pursued as much as possible.

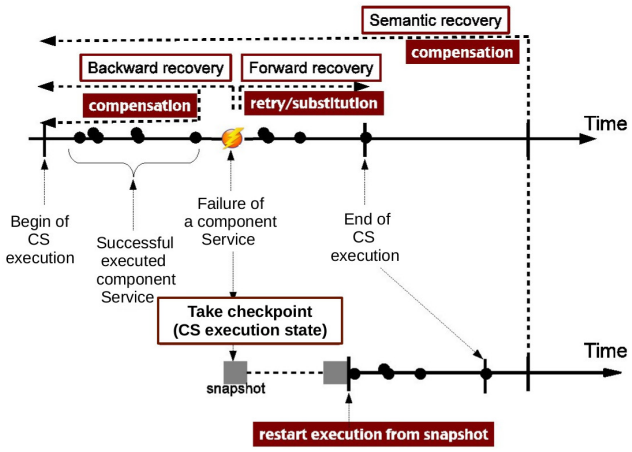


Figure 4.2: Failure recovery strategies for composite service CS

All recovery techniques discussed above support a relaxed "all-or-nothing" model to maintain system consistency. As shown in Figure 4.2, this model is better described as "all-or-(almost)nothing" for

composite services, since users may observe partial results from successfully executed services before compensation is applied. We propose *checkpointing* as an alternative that further relaxes atomicity while preserving fault tolerance (Cardinale et al. [55], Rukoz et al. [197]). This enables an "all-something-or-nothing" behavior: upon failure, a snapshot of the most advanced partial state is returned, allowing users to accept or reject intermediate results. Unlike compensation, where users receive "(almost)nothing", checkpointing gives them partial control. The checkpointed composite service, comprising outputs from completed services and unexecuted services due to failure, can later be resumed from the snapshot to complete the execution and produce the full result.

In composite services, recovery may combine several failure recovery mechanims. By relaxing isolation and atomicity, such mechanisms help maintain integrity and resilience under partial failures. These strategies enable service compositions to function effectively in real-world scenarios, where strict rollback is neither always feasible nor desirable. The rest of the chapter is organized as follows. In Section 4.2, we outline state-of-the-art research work for failure recovery techniques in service compositions and summarizes our contributions in this area. In Section 4.3, we propose an Petri net-based failure recovery approach that combine both backward and forward recovery, while in Section 4.4, we propose an graph-based failure recovery approach that combine both semantic recovery and checkpointing. In Section 4.5, we overview additional research work about concurrency control during composite service execution that we had the opportunity to study. In the last section, we conclude the chapter by summarizing our findings.

4.2 State-of-the-art and Contributions

Over the past two decades, numerous studies have addressed reliability and failure recovery strategies (Gao et al. [94], Immonen et al. [119], Vargas-Santiago et al. [228]) in service composition. These efforts have evolved from rule-based static approaches to modern predictive and self-healing systems. Next, we outline the evolution of these approaches with particular emphasis on transactional-based ones.

Early mechanisms relied on rule-based and exception handling mechanisms to achieve forward error recovery and ensure fault tolerance in service compositions (Brambilla et al. [39], Casati and Cugola [57], Friedrich et al. [88], Hagen and Alonso [109]). These approaches rely on explicitly defined recovery rules, often embedded within orchestration languages such as BPEL, through try-catch-finally blocks and compensation handlers, to manage failures during execution. Service substitution was limited to predefined alternatives, and compensation was employed to undo previously executed operations. However, these strategies were inherently reactive, requiring prior anticipation of potential faults, and thus unable to handle runtime failures or dynamic execution contexts.

To overcome the limitations of early failure recovery methods, a new wave of approaches introduced transactional models that support both forward and backward recovery. By leveraging the transactional properties of component services, these approaches enable fault-tolerant service compositions by defining service behavior in case of failure and specifying the corresponding recovery actions. Among them, Portilla et al. [195] proposed atomicity contracts, grouping services to ensure collective success or failure, along with recovery strategies based on service transactional behavior. Zhao et al. [263] introduced relaxed atomicity, where a composition is transactionally correct if all its components reach a unanimous outcome or a "completed accepted" state, softening the strict "all-or-nothing" principle. Additional correctness criteria have been formulated through transactional coordination protocols (Pires et al. [192]), or explicit fault-handling mechanisms (Liu et al. [145]). While some approaches rely on exception handling or on transactional properties, some others combine both. Among these, Liu et al. [144, 145, 146] developed a series of taxonomies to classify services by their transactional capabilities. They first distinguished between atomic, quasi-atomic (compensatable), and non-atomic (irreversible) services, later refining this into atomic, semantic-atomic, weak-atomic, and pivot services based on lifecycle phases and support for compensation or two-phase commit (2PC). Lastly, they proposed a two-dimensional model combining compensatable and cancellable properties, resulting in the FACTS framework. FACTS enables hybrid fault tolerance by integrating exception handling with relaxed atomicity, using declarative Event-Condition-Action rules and a scalable commit protocol inspired by 2PC. Several complementary approaches followed. Maamar et al. [161] introduced a forward recovery approach based on context-aware policies encapsulating service transactional behavior. Schäfer et al. [200] extended the Web Services Coordination protocol to support flexible compensation via service replacement and additional invocations. Ben Lakhal

et al. [133] proposed the FENECIA framework, classifying services by vitality and compensation properties (*e.g.*, vital-compensatable, non-vital non-compensatable). FENECIA is based on WS-SAGAS transactions and supports forward recovery via retry or replacement, as well as backward recovery via compensation. If a vital service fails and cannot be recovered, the entire composition is aborted; otherwise, execution may proceed. Later, Cardinale and Rukoz [54] proposed a framework for the reliable execution of transactional composite services using Coloured Petri Nets (CPN). Their approach handles service failures based on transactional properties modeled within the CPN, supporting backward recovery (via compensation), forward recovery (via retry or replacement), and semantic recovery. Failed services are retried when possible; if retry fails, they are replaced with transactional equivalents; if replacement also fails, compensation is applied. This work was later extended by Angarita et al. [14], who introduced the FaCETa framework, combining forward recovery through service replacement and backward recovery using a CPN-based unrolling algorithm.

For certain queries, partial results can still be valuable to users, underscoring the need for recovery strategies that provide usable output even in the event of failure. Checkpointing offers such a strategy by enabling partial result delivery and supporting replication as a proactive mechanism.Using checkpointing, Rukoz et al. [197] proposed an approach where, upon failure, the unrolling process of the CPN controlling the execution of a transactional composite service is checkpointed, allowing execution to continue as far as possible. This enables users to receive partial results as they become available and, if needed, to re-submit the checkpointed Petri net to resume execution from the saved state.

Recent recovery approaches in service compositions leverage machine learning for proactive fault tolerance through failure prediction (Singh et al. [208]). Notably, Alhosban et al. [8] proposed the SFSS framework, which predicts failures based on service fault history, execution time relative to the overall composition, and service importance in the compositon. SFSS enables preemptive planning when high failure probability is detected, by selecting the optimal execution plan using a scoring function. El Ghondakly et al. [83] introduced a deep learning-based method for defect localization and prediction, using convolution neural network (CNN), recurrent neural network (RNN), and a hybrid CNN-RNN model. Their approach identifies faulty services in terms of number, location, and timing. While RNNs offered faster predictions, the method did not reduce computational complexity.

Positioning of contributions.

This section presents my contributions to correct and dependable execution of service composition, with further details in the following sections. It focuses on failure recovery strategies that ensure reliability by leveraging transactional properties of component services, along with additional work on concurrency control.

For failure recovery, as a foundational step in Cardinale et al. [50] (Section 4.3), we proposed a framework based on coloured Petri nets to model both the execution and compensation flows of transactional composite web services whose components locally optimize QoS. The framework supports forward and backward recovery: in case of failure, forward recovery is attempted via re-execution or service substitution; if that fails, backward recovery is applied through compensation to preserve consistency. This work aligns with prior efforts by Angarita et al. [14] and Cardinale and Rukoz [54], contributing to the class of CPN-based recovery techniques.

Expanding on this, in Cardinale et al. [51, 52] (Section 4.4), we focused on transactional aspect and introduced the notion of fuzzy atomicity, which relaxes the classical "all-or-nothing" execution model. In Cardinale et al. [51], by integrating transactional properties with checkpointing, we proposed a model supporting partial execution results in the presence of failures by replacing the classical "all-or-nothing"with a more flexible and realistic fuzzy "all-something-or-(almost)nothing" execution model. In case of failure of a component service, compensation allows to semantically undone the effects of component services that have been executed before the failure. Checkpointing mechanism allows to provide partial responses to a query, in case of failure, along with a snapshot of the composite service execution, in order to be able to execute later the part of the composite service that cannot be executed after the failure. This approach enhances user satisfaction by allowing partial but meaningful responses when total recovery is infeasible, complementing checkpointing-based recovery of Rukoz et al. [197]. Building upon this foundation, in Cardinale et al. [52], we further enhanced our fuzzy atomicity model to introduce greater expressive-

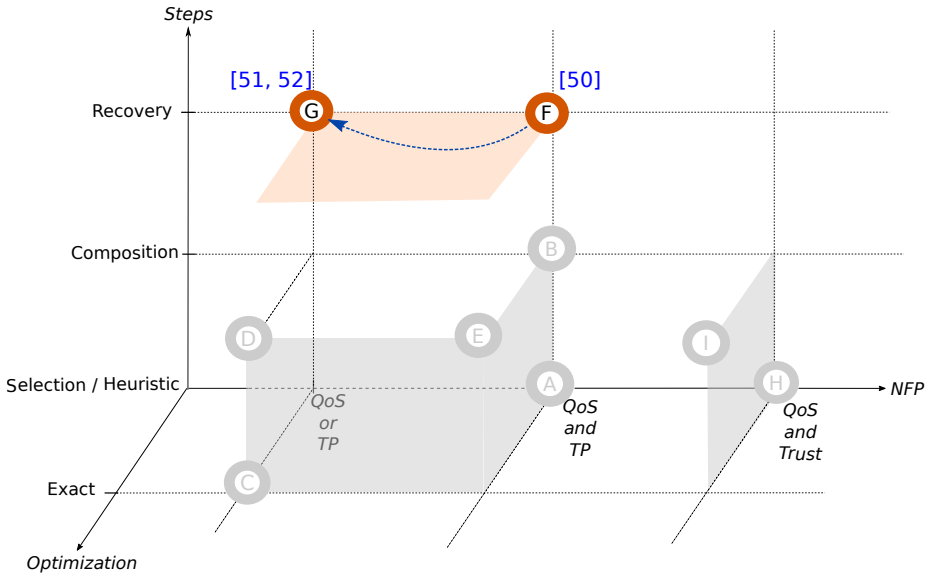


Figure 4.3: Contributions to Failure Recovery in Service Composition Execution

ness and user-centric control. Our refined model allow users to better specify their desired outputs and to consider the state of the composite service execution. It also allows to relax the retriable transactional property, which is difficult to ensure in high dynamic environments. This refined framework allows for adaptive recovery strategy selection including forward, backward, semantic, and checkpointing depending on the minimum user requirement, the composition state, and the failed service, echoing trend in self-healing recovery.

For concurrency control, our work in El Haddad et al. [81](Section 4.5) deal with concurrency between composite services while respecting relaxed isolation. A composite service, considered as an open nested transaction, may reveal partial results that can be used by other services. We proposed a decentralized serialization graph mechanism built on an optimistic protocol and hierarchical composition structure to ensure globally correct execution, aligning with the needs of scalable and adaptive service orchestration.

4.3 Forward and Backward Recovery

The content of this section is adapted from Cardinale et al. [50] work carried out in collaboration with colleagues from Simón Bolívar University and Université Paris Dauphine-PSL

Overview

This work [50] builds on our previous work by Cardinale et al. [48] (see Chapter 3, Section 3.3.2) where we extended the Colored Petri Net (CPN) formalism to incorporate non-functional component service properties and to model composition phases.

In the first phase, the `COMPOSER` module automatically selects services and generates their execution flow as a CPN. This selection is guided by an unrolling algorithm that uses QoS and transactional parameters to prune the search space and ensure optimal and reliable compositions. Two CPNs are generated: one for the transactional composite service that meets functional, QoS, and transactional requirements, and another CPN for the compensation flow in case of failures. In the first CPN, colors represent transactional properties, while in the second, they track execution states for failure handling.

In the second phase, the `EXECUTER` module carries out the service execution based on the generated CPNs. It ensures fault-tolerant execution of a transactional composite web service by: (i) following the specified sequential/parallel flow, (ii) applying forward recovery by substituting faulty services when possible, and (iii) performing backward recovery via compensation if substitution fails.

Next, we describe our combined forward and backward recovery mechanism through the `COMPOSER` and the `EXECUTER` modules.

The COMPOSER

The firing of a transition of a WSDN corresponds to the selection of a service, which will participate in the composite service allowing to answer the user query Q. We define the *marking* of a WSDN, the *fireable* property of a transition, and the *firing rules* in such way we obtain, at the end, a transactional composite web service (see Chapter 2, Section 2.3). Thus, given a user query Q and a WSDN, the selection process will create a CPN, called $WSDN_Q$, sub-part of WSDN, which satisfies Q and its corresponding compensation CPN, called BR_WSDN_Q, representing the backward recovery process as follows:

Definition 4.3.1. *A $WSDN_Q$ is a 4-tuple (A_Q, S_Q, F_Q, ξ_Q), where:*

- $A_Q \subseteq A \mid I_Q \subseteq A_Q \wedge O_Q \subseteq A_Q$;
- $S_Q \subseteq S$;
- $F_Q : (A_Q \times S_Q) \cup (S_Q \times A_Q) \rightarrow \{0,1\}$ is a flow relation indicating the presence (1) or the absence (0) of arcs between places and transitions defined as follows:
  $\forall s \in S_Q, (\exists a \in A_Q \mid F_Q(a,s) = 1$ if $F(a,s) = 1)$ and $\forall s \in S_Q, (\exists a \in A_Q \mid F_Q(s,a) = 1$ if $F(s,a) = 1)$;
- ξ_Q is a color function such that $\xi_Q : S_Q \rightarrow \Sigma_S$ and $\Sigma_S = \{p, pr, a, ar, c, cr\}$ represents the TP of $s \in S$.

The global transactional property of $WSDN_Q$ ensures that if a component service, whose transactional property does not allow forward recovery fails, then all previous executed services can be semantically recovered by a backward recovery. For modeling the compensation flow of a transactional composite web service , we formally define BR_WSDN_Q as follows:

Definition 4.3.2. A BR_WSDN_Q, associated to a given $WSDN_Q$=(A_Q, S_Q, F_Q, ξ_Q), is a 4-tuple (A', S', F^{-1}, ζ), where:

- A' is a finite set of places corresponding to the $WSDN_Q$ places such that: $\forall a' \in A'\ \exists a \in A_Q$ associated to a' and a' has the same semantic of a.
- S' is a finite set of transitions corresponding to the set of compensation services in $WSDN_Q$ such that: $\forall s \in S_Q, \xi_Q(s) \in \{c, cr\}, \exists s' \in S'$ which compensate s.
- F^{-1}:$(A_Q \times S_Q) \cup (S_Q \times A_Q) \rightarrow \{0,1\}$ is a flow relation establishing the restoring order in a backward recovery defined as: $\forall s' \in S'$ associated to $s \in S_Q, \exists\ a' \in A'$ associated to $a \in A_Q \mid F^{-1}(a', s') = 1 \Leftrightarrow F(s,a) = 1$ and $\forall s' \in S', \exists a' \in A' \mid F^{-1}(s', a') = 1 \Leftrightarrow F(a,s) = 1$.
- ζ is a color function such that $\zeta : S' \rightarrow \Sigma'_S$ and $\Sigma'_S = \{In, Ru, Ex, Co, Fa, Ab\}$ represents the execution state of $s \in S_Q$, and $s' \in S'$ is its compensation service (In: initial, Ru: running, Ex: executed, Co: compensated, Fa: Failed, and Ab: abandoned).

We consider that a service should be selected at most once, therefore the corresponding transition in $WSDN$ should be fired only one time. As a consequence, when a transition s is fired, tokens are added to its output places and all tokens are deleted from its input places (except from places that belong to O_Q). If s is compensatable $(C_S(s) \in \{c, cr\})$, its corresponding compensation service, s', has to be added in the compensation CPN. Each service s includes in its semantic description the corresponding reference to its compensatable service s'; s and s' are registered in the same registry. More formally, we defined the firing rules as follows:

Definition 4.3.3. The firing of a fireable transition s for a marking M defines a new marking M', denoted as $M \xrightarrow{s} M'$, such that :

1. Tokens are added to the output places of s depending on the color of s and on the color of the tokens contained by the input places of s, according to the following rules:
   if ($\exists x \in ({}^\bullet s) \mid a \in M(x)$), then $\forall y \in (s^\bullet)$, $M'(y) \leftarrow M'(y) \cup \{a\}$
   else if ($\exists x \in ({}^\bullet s) \mid ar \in M(x)$), then $\forall y \in (s^\bullet)$, $M'(y) \leftarrow M'(y) \cup \{ar\}$
   else if [$(\exists x \in ({}^\bullet s) \mid c \in M(x)) \wedge (C_S(s) \in \{p, pr, a, ar\})$],
   then $\forall y \in (s^\bullet)$, $M'(y) \leftarrow M'(y) \cup \{a\}$
   else if [$(\exists x \in ({}^\bullet s) \mid c \in M(x)) \wedge (C_S(s) \in \{c, cr\})$],
   then $\forall y \in (s^\bullet)$, $M'(y) \leftarrow M'(y) \cup \{c\}$
   else /*in this case: $\forall x \in ({}^\bullet s)$, $M(x) \in Bag(\{I, cr\})$*/
   $\forall y \in (s^\bullet)$, $M'(y) \leftarrow (M'(y) \cup C_S(s))$ if $C_S(s) \in \{a, ar, c, cr\}$,
   $M'(y) \leftarrow (M'(y) \cup \{a\})$ if $C_S(s) = p$, and $M'(y) \leftarrow M'(y) \cup \{ar\}$ if $C_S(s) = pr$

2. Tokens are deleted from input places of s, if they do not belong to O_Q:
   $\forall x \in ({}^\bullet s - O_Q)$, $M(x) \leftarrow \emptyset$,

3. Color $\mathcal{C}_{M'}$ of the resulting $(WSDN, M')$ is updated, according to the following rules:
   if $(\mathcal{C}_M \in \{I, cr\})$ and $C_S(s) = p$ then $\mathcal{C}_{M'} \leftarrow a$
   else if $(\mathcal{C}_M \in \{I, cr\})$ and $C_S(s) = pr$ then $\mathcal{C}_{M'} \leftarrow ar$
   else if $(\mathcal{C}_M \in \{I, cr\})$ and $C_S(s) \in \{a, ar, c, cr\}$ then $\mathcal{C}_{M'} \leftarrow C_S(s)$
   else if $(\mathcal{C}_M = c)$ and $C_S(s) \in \{p, pr, a, ar\}$ then $\mathcal{C}_{M'} \leftarrow a$
   else $\mathcal{C}_{M'} \leftarrow \mathcal{C}_M$

4. *s and its input and output places are added to $WSDN_Q$=(A_Q, S_Q, F_Q, ξ_Q) as:*
   $A_Q \leftarrow A_Q \cup {}^\bullet s \cup s^\bullet$; $S_Q \leftarrow S_Q \cup \{s\}$; $F_Q(a, s) \leftarrow 1, \forall a \in {}^\bullet s$; $F_Q(s, a) \leftarrow 1, \forall a \in s^\bullet$

5. *If $C_S(s) \in \{c, cr\}$, its compensation WS s' and its input and output places are added to $BR\_WSDN_Q = (A', S', F^{-1}, \zeta)$, according to the following rules:*
   $A' \leftarrow A' \cup \{a' \mid \exists$ a $\in A_Q \wedge$ a $\in {}^\bullet s \vee$ a $\in s^\bullet\}$; $S' \leftarrow S' \cup \{s'\}$; $F^{-1}(a', s') = 1 \Leftrightarrow F_Q(s, a) = 1$;
   $F^{-1}(s', a') = 1 \Leftrightarrow F_Q(a, s) = 1$

Associated to $WSDN_Q$, there exists a firing sequence $\sigma = \{s_1, \ldots, s_n \mid s_i \in S_Q\}$, such that: $M_Q \xrightarrow{\sigma} M_F$, where M_Q is the initial marking and M_F denotes the desired marking in which $\forall o \in O_Q, M_F(o) \neq \emptyset$. When several transitions are fireable, to select which transition has to be fired, we use the quality measure presented in Chapter 3, Section 3.3.2, Definition 3.3.1.

Based on the presented formalism, we have implemented the unrolling algorithm to automatically generate $WSDN_Q$ and BR_WSDN_Q.

The EXECUTER

Once $WSDN_Q$ and BR_WSDN_Q are obtained, the transactional composite web service have to be executed ensuring a consistent state of the system in presence of failures. The execution process is managed by algorithms that execute the CPNs. We formally describe the execution process in following definitions.

The marking of a $WSDN_Q$ or BR_WSDN_Q represents the current values of attributes that can be used for some component services to be invoked or control values indicating the compensation flow, respectively. A Marked CPN denotes which transitions can be fired.

Definition 4.3.4. A marked executable WSDN =(A, S, F, ξ) is a pair (CPN,M), where M is a function which assigns tokens (values) to places such that $\forall a \in A$, $M(a) \in \mathbb{N}$.

Definition 4.3.5. A marking M enables a transition s iff all its input places contain tokens such that: $\forall x \in (^\bullet s), M(x) \geq card(^\bullet x)$

During execution in $WSDN_Q$ and BR_WSDN_Q, a transition (*i.e.*, service invocation) is fireable only if all its predecessor transitions have fired. Execution depends solely on the number of tokens in places, not their colors. While colors are used in transitions, only token counts matter in places; sufficient to enforce correct sequential and parallel execution, preserving the global transactional property.

Example 4.3.1. As shown in Figure 4.4, ws_3 needs two tokens in a_3 to be invoked. Since a_3 is produced by both ws_1 and ws_2, ws_3 must wait for both to complete. Although ws_1 has already placed a token in a_3, and ws_2 is still running, firing ws_3 prematurely would result in a parallel execution with ws_2, which may violate their intended transactional order. Therefore, ws_3 must wait until both predecessors finish, after ws_3 and ws_4 can be executed in parallel.

This example illustrates how data flow constrains execution flow to maintain transactional consistency, though in some cases, both flows may naturally align.

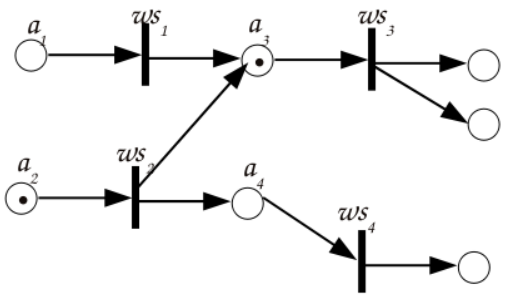


Figure 4.4: Example of Fireable Transitions

The execution of a transactional composite web service is driven by an algorithm executing its corresponding $WSDN_Q$=(A_Q, S_Q, F_Q, ξ_Q). To enable backward recovery, execution traces are recorded on BR_WSDN_Q=(A', S', F^{-1}, ζ). Execution begins with an *Initial Marking*: a token is added to each place representing an input of Q ($\forall a \in (A_Q \cap I_Q), M(a) = 1, \forall a \in (A_Q - I_Q), M(a) = 0$), and the state of all transitions in BR_WSDN_Q is set to *initial* ($\forall s' \in S', \zeta(s') \leftarrow In$). Firing a transition in $WSDN_Q$ corresponds to the execution of a service s which participates in the composition. While s runs, the state of its corresponding s' in BR_WSDN_Q is marked *running* ($\zeta(s') \leftarrow Ru$). Once s completed, s' is marked *executed* ($\zeta(s') \leftarrow Ex$), others transitions become fireable, and the following *firing rules* are applied.

Definition 4.3.6. The firing of a fireable transition s for a marking M defines a new marking M', such that: all tokens are deleted from its input places ($\forall x \in {}^\bullet s$, $M(x) = 0$), if the $\xi_Q(s) \in \{c, cr\}$, the state of its corresponding s' in BR_WSDN_Q is set to *running*($\zeta(s') \leftarrow Ru$), and the service s is invoked. After s finishes, tokens are added to its output places ($\forall x \in (s^\bullet)$, $M(x) = M(x) + card(x^\bullet)$), and the state of its corresponding s' in BR_WSDN_Q (if it exists) is set to *executed*($\zeta(s') \leftarrow Ex$).

If a service s fails and $\xi_Q(s)$∈{pr, ar, cr}, it is re-invoked until it successfully finishes (forward recovery). Otherwise, its corresponding s' in BR_WSDN_Q (if present) is marked as *failed* ($\zeta(s') \leftarrow Fa$), triggering backward recovery. This requires compensating all previously executed services in reverse order; parallel execution services can be compensated in any order. Backward recovery halts the execution

of $WSDN_Q$ and the compensation process is initiated over BR_WSDN_Q with its *Initial Marking* (*i.e.*, tokens are added only in input places of BR_WSDN_Q). The execution of BR_WSDN_Q then follows Definition 4.3.7 and Definition 4.3.8.

Definition 4.3.7. A marking M enables a transition s' iff all its input places contain tokens such that $\forall a' \in ({}^\bullet s'), M(a') \neq 0, \wedge\, \zeta(s') \notin \{Co, Ab\}$.

Definition 4.3.8. The firing of a fireable transition (see Defintion 4.3.7) s' for a marking M defines a new marking M', such that:

- if $\zeta(s') = In$, $\zeta(s') \leftarrow Ab$ (i.e., the corresponding s is abandoned before its execution),
- if $\zeta(s') = Fa$, $\zeta(s') \leftarrow Ab$ (i.e., the corresponding s is abandoned, it has failed),
- if $\zeta(s') = Ru$, $\zeta(s') \leftarrow Co$ (in this case s' is executed after s finishes, then s is compensated),
- if $\zeta(s') = Ex$, $\zeta(s') \leftarrow Co$ (in this case s' is executed, i.e., s is compensated),
- tokens are deleted from its input places ($\forall x \in {}^\bullet s'$, $M(x) = M(x) - 1$) and tokens are added to its output places as many successors it has ($\forall x \in (s'^\bullet)$, $M(x) = card(x^\bullet)$).

Figure 4.5 illustrates a backward recovery scenario. The marked $WSDN_Q$ in Figure 4.5(a) and the corresponding BR_WSDN_Q in Figure 4.5(b) represent the execution state at the moment ws_4 fails. The execution of $WSDN_Q$ is halted, and the initial marking on BR_WSDN_Q is set to start its execution process (Figure 4.5(c)). After firing ws'_3 and ws'_5 to compensate ws_3 and ws_5, and abandoning ws_4 and ws_7 (the former failed; the latter wasn't yet invoked), a new marking is produced (Figure 4.5(d)) where ws'_1 and ws'_2 become fireable and can be executed in parallel. Only compensatable services have corresponding transitions in BR_WSDN_Q; for instance, ws_6 and ws_8 (and their outputs a_{10} and a_{12}) are excluded as they are not compensatable.

Failures occurring late in execution can lead to high resource waste due to compensation. However, ensuring full retriability for forward recovery is challenging. To address this, we proposed a forward recovery approach based on *service substitution* (Cardinale and Rukoz [53]. This leverages *service classes* (Azevedo et al. [21]), which group semantically equivalent services (*i.e.*, services with the same functionality but differing in input and output attributes, transactional properties, and QoS attributes). These classes are assumed to be predefined and described semantically in the service registry. Upon failure, if a service is not retriable, a substitute is automatically selected and invoked.

To this end, we defined *functional substitute* and *exact functional substitute* as follows:

Definition 4.3.9. Let SC be a service class, if s, $s^* \in SC$, we say that s is a functional substitute of s^* (denoted as $s \equiv_F s^*$), if $({}^\bullet s^*) \subseteq ({}^\bullet s) \wedge (s^*)^\bullet \supseteq (s^\bullet)$.

Definition 4.3.10. Let SC be a service class, if s, $s^* \in SC$, we say that s is an exactly functional substitute of s^* (denoted as $s \equiv_{EF} s^*$), if $({}^\bullet s^*) = ({}^\bullet s) \wedge (s^*)^\bullet = (s^\bullet)$.

In a service class, the functional equivalence is defined according to the services input and output attributes. A service s is a *functional substitute* of another service s^*, if s^* can be invoked with at most the input attributes of s and s^* produces at least the same output attributes produced by s. They are exactly functional substitutes if they have the same input and output attributes.

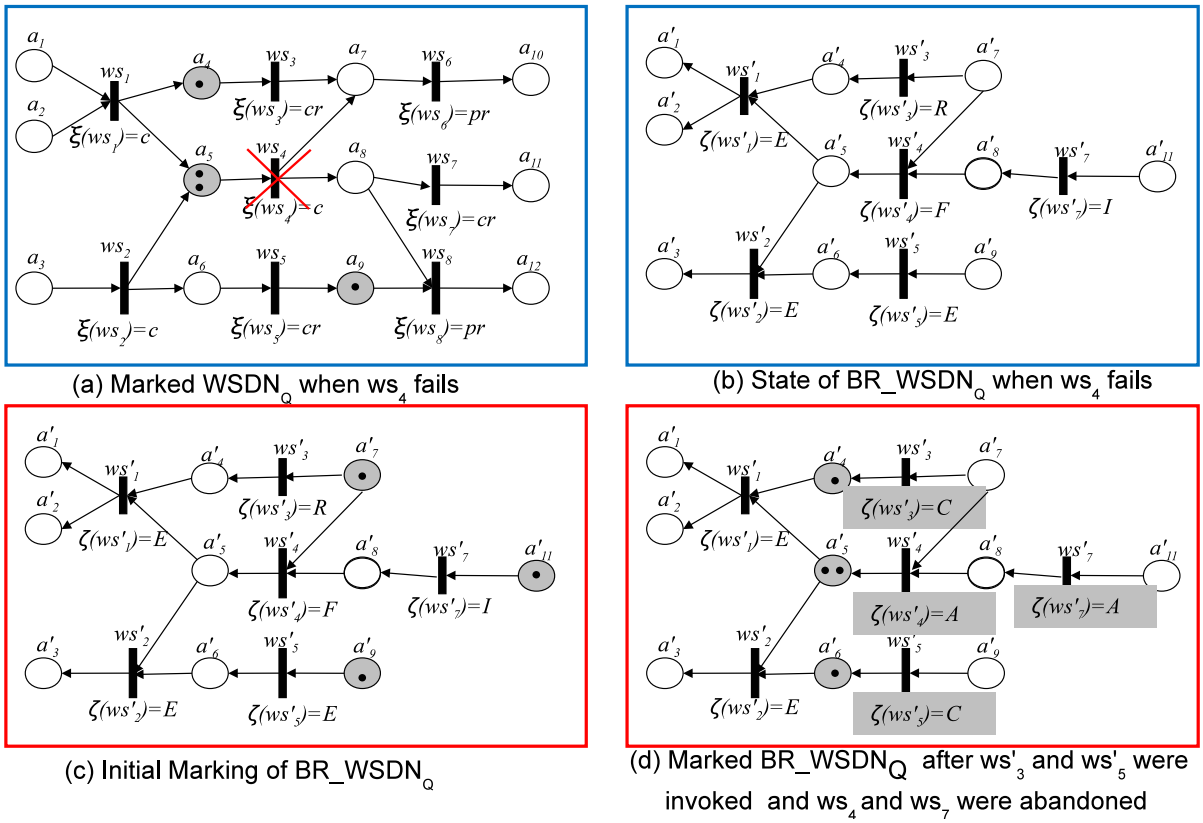


Figure 4.5: Example of backward recovery

Example 4.3.2. Several functional equivalence examples are illustrated in Figure 4.6. For instance, $ws_1 \equiv_F ws_2$, however $ws_2 \not\equiv_F ws_1$, because ws_1 does not produce output a_5 as ws_2 does. $ws_1 \equiv_F ws_3$, $ws_3 \equiv_F ws_1$, and also $ws_1 \equiv_{EF} ws_3$.

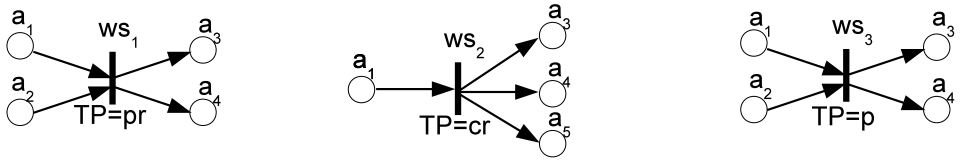


Figure 4.6: Example of equivalent services

To guarantee the global transactional property of the transactional composite web service , a service s can be replaced by another s^* if the latter can behave as s in the recovery process. Hence, if $\xi_Q(s)$=p (s only allows backward recovery), it can be replaced by any other service since all transactional properties allow backward recovery. If $\xi_Q(s) = pr$, it can be replaced by any retriable service (pr,ar,cr), as all of them allow forward recovery. An a service allows only backward recovery, then it can be replaced by another service with backward recovery capability. A c service can be replaced by a service that provides semantic recovery as c and cr services. While a cr service can be only replaced by another cr service as it supports

forward, backward, and semantic recovery. We therefore define the notion of a *transactional substitute* as follows:

Definition 4.3.11. Let SC be a service class, if s, $s^* \in SC$, we say that s is a transactional substitute of s^* (denoted as $s \equiv_T s^*$):

- if $(\xi_Q(s) = p, \xi_Q(s^*) \in \{p, pr, a, ar, c, cr\}) \wedge (s \equiv_F s^*)$,
- if $(\xi_Q(s) = pr, \xi_Q(s^*) \in \{pr, ar, cr\}) \wedge (s \equiv_F s^*)$,
- if $(\xi_Q(s) = a, \xi_Q(s^*) \in \{a, ar, c, cr\}) \wedge (s \equiv_F s^*)$,
- if $(\xi_Q(s) = ar, \xi_Q(s^*) \in \{ar, cr\}) \wedge (s \equiv_F s^*)$,
- if $(\xi_Q(s) = c, \xi_Q(s^*) \in \{c, cr\}) \wedge (s \equiv_{EF} s^*)$,
- if $(\xi_Q(s) = cr, \xi_Q(s^*) = cr) \wedge (s \equiv_{EF} s^*)$.

Example 4.3.3. Several transactional substitute examples are given in Figure 4.6. For instance, $ws_1 \equiv_T ws_2$, because $ws_1 \equiv_F ws_2$ and $\xi_Q(ws_2) = cr$, then ws_2 can behave as a pr WS; however $ws_1 \not\equiv_T ws_3$, even $ws_1 \equiv_F ws_3$, because as $\xi_Q(ws_3) = p$, w_3 cannot behave as a pr web service.

Transactional substitute definition allows *services substitution* in case of failures as defined below:

Definition 4.3.12. Let $WSDN_Q$=(A_Q, S_Q, F_Q, ξ_Q) be the CPN allowing the execution of a transactional composite service that satisfies the Query $Q = (I_Q, O_Q, W_Q, T_Q)$, and BR_WSDN_Q=(A', S', F^{-1}, ζ) its corresponding backward recovery CPN. In case of a service $s \in S_Q$ fails, it can be replaced by another s^*, if $s \equiv_T s^*$, and the following actions proceed:

1. $S_Q \leftarrow S_Q \cup \{s^*\}$;
2. $\forall a \in {}^\bullet(s^*), F(a, s^*) = 1 \wedge \forall a \in {}^\bullet s, F(a, s) = 0$;
3. $\forall a \in s^\bullet, F(s^*, a) = 1, F(s, a) = 0$;
4. $S_Q \leftarrow S_Q - \{s\}$;
5. if $\xi_Q(s) \in \{c, cr\}$, $s' \in S'$ is replaced by s'^* (it compensates s^*) applying 1, 2, 3, and 4 on BR_WSDN_Q.

When a substitution occurs, the faulty service s is removed from $WSDN_Q$ and replaced by s^* while preserving the original sequential relation defined by the input and output attributes of s. For compensatable services, exact functional substitutes is a must to avoid altering the compensation flow in BR_WSDN_Q. The goal is to complete the transactional composite web service execution with the same properties as the original. Figure 4.7(a) shows a CPN registry; Figure 4.7(b) a $WSDN_Q$; and Figure 4.7(c) the updated $WSDN_Q$ after replacing ws_1 with ws_2. Note that a_5, produced by ws_2 but not originally by ws_1, is excluded to preserve the original execution flow.

When in a service class there exist several service candidates for replacing a faulty s, it is selected the one with the best quality measure. The quality of a transition depends on the user query Q and on its QoS values. Services substitution is done such that the substitute service locally optimize the QoS. A similar quality measure used by the COMPOSER is used during the execution, in order to keep the same heuristic to select substitutes.

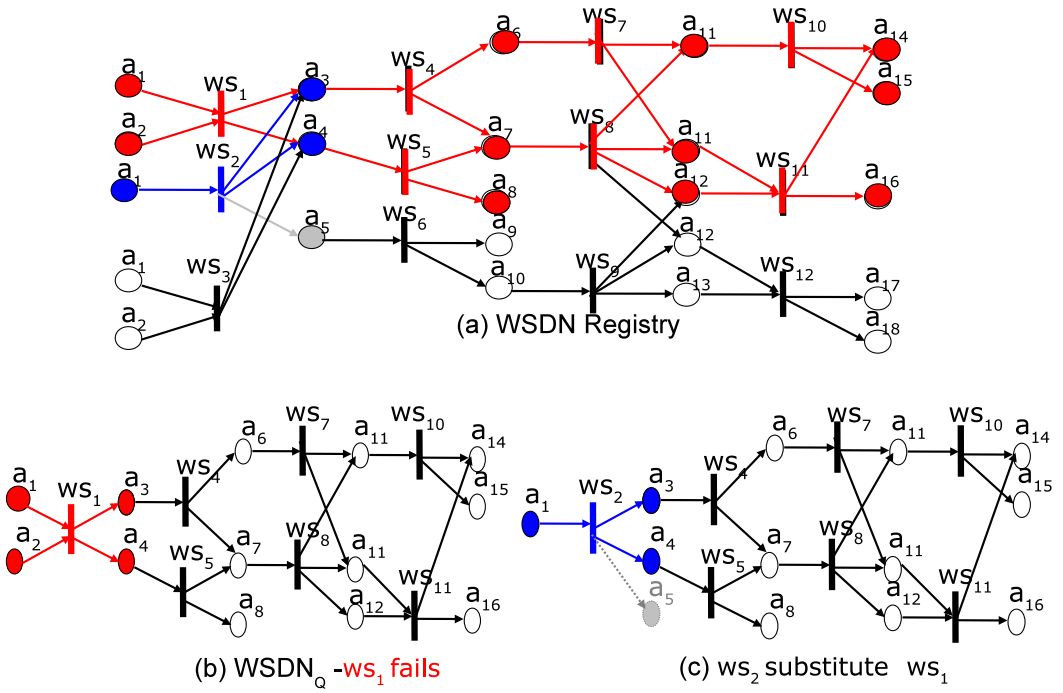


Figure 4.7: Example of services substitution

In summary, when a service s fails, the recovery action depends on its transition color ($\xi_Q(s)$): (i) if $\xi_Q(s)$ is retriable, s is re-invoked until successful (forward recovery); (ii) if not retriable, a transactional substitute s^* is selected and execution continues (forward recovery); (iii) if no substitute exists, the corresponding s' in BR_WSDN_Q (if it exists) is marked *failed* ($\zeta(s') \leftarrow F$), triggering backward recovery.

4.4 Semantic and Checkpointing Recovery

The content of this section is adapted from Cardinale et al. [51] and [52] work carried out in collaboration with colleagues from Simón Bolívar University and Université Paris Dauphine-PSL

Overview

In this work, we proposed a *fuzzy* model to measure relaxed atomicity in composite service execution. By leveraging transactional properties of services and checkpointing mechanism, the model softens the strict "all-or-nothing" guarantee into an "all-something-or-(almost)nothing" (called *fuzzy* "all-or-nothing") behavior. Given a user query defining available inputs (*e.g.*, name, date), expected outputs (*e.g.*, flight ticket, climate prediction), and the *acceptable fuzzy atomicity* (*i.e.*, minimum acceptable results), our model computes a *fuzzy* atomicity score based on the outputs produced and services successfully executed. Depending on the score value, different recovery techniques could be envisaged in case of failures to ensure a *fuzzy* atomic (*i.e.*, *fuzzy* "all-or-nothing") execution of the composite service.

In Cardinale et al. [51], we introduced an atomicity model that quantifies the relaxation of the "all-or-nothing" property based on user-defined output acceptability thresholds. In Cardinale et al. [52], we extended the model to i) improve user expressiveness in output preferences and ii) relax the retriable property, which is challenging to guarantee in dynamic environments such as Cloud platforms.

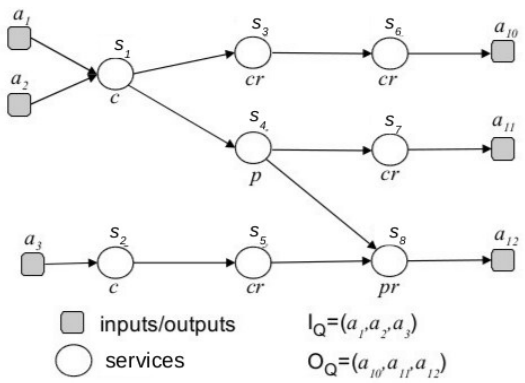


Figure 4.8: Atomic CS_Q: at least a p service in the composition

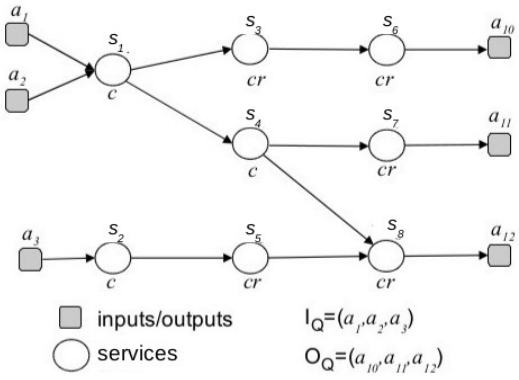


Figure 4.9: Compensatable CS_Q: all component services are c

Relaxing atomicity by compensation

Transactional properties are useful to guarantee reliable composite service execution and to ensure the whole system consistent state even in presence of failures.

In the event of failures, backward recovery provides "nothing" or "(almost)nothing" in case of compensation. Figure 4.8 shows an atomic composite service CS_Q for a query with $I_Q = \{a_1, a_2, a_3\}$ and $O_Q = \{a_{10}, a_{11}, a_{12}\}$. Retriable services are assumed to succeed, enabling forward recovery. Only c and p services (s_1, s_2, and s_4) may fail without repair. If a parallel c service fails (*e.g.*, s_1 or s_2), the other can be compensated. If the p service (s_4) fails, its preceding sequential services (s_1, s_2) and parallel services (s_3, s_5, s_6) must be compensated. In a compensatable composite service, any unrecoverable failure leads to compensation of all related sequential and parallel services, yielding semantic recovery with limited output "(almost) nothing"). Figure 4.9 illustrates such a compensatable CS_Q.

In both compensatable retriable and atomic retriable composite services, the retriable condition ensures successful execution, guaranteeing the user receives all desired outputs even in the presence of failures. Once all services complete successfully, users obtain 100% of their intended results. However, if a failure occurs, execution halts completely even if unaffected services could still produce partial outputs. This motivates the idea of *relaxed atomicity*, where successfully executed components (with outputs) are considered alongside failed ones. Unexecuted services may still be invoked later as we will present below.

Relaxing atomicity by checkpointing

Suppose the atomic composite service in Figure 4.8 reaches the execution state shown in Figure 4.10 when s_4 fails. Although s_4 cannot be repaired, execution can still proceed with unaffected services. Only s_7 and s_8 depend on s_4, so they are the only ones impacted. Assuming no further failures, the output a_{10} can still be produced.

Figure 4.11 shows the resulting state after executing all unaffected services and successfully obtaining a_{10}. The red subgraph highlights the remaining part of the composite service (*i.e.*, services that were not executed because they depend on the failed s_4 or its outputs). This subgraph can later be re-executed to recover the missing outputs.

To formalize our approach, we define key concepts of the checkpointing process in Definitions 4.4.1 and 4.4.2, corresponding to *local* and *global* snapshots.

Definition 4.4.1. A Local Snapshot is a 2-tuple representing the state of a service s_i in CS_Q, denoted as LS_{s_i}=<In_{s_i}, Out_{s_i}>, where:

- In_{s_i} represents the values of inputs of I_{s_i} that can be used to invoke s_i. Then: $In_{s_i} = \{(i, v) \mid i \in I_{s_i}$ and v is the value of input $i\}$, where $v \neq$NULL, if (i$\in I_Q$) or ($\exists s_j \in CS \mid i \in O_{s_j}$ and s_j

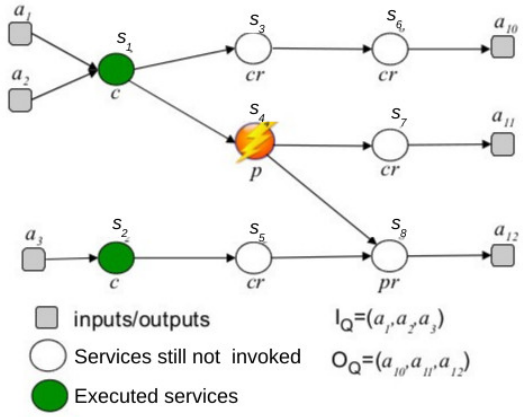


Figure 4.10: CS_Q when s_4 fails

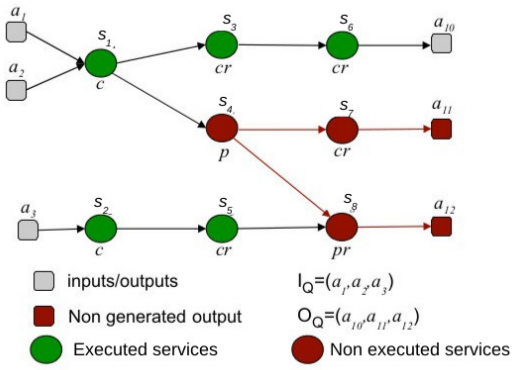


Figure 4.11: Checkpointed CS_Q

was successfully executed) (i.e., the value is part of I_Q or it was produced by a service s_j executed before s_i); or $v = NULL$ otherwise (i.e., the value is not in I_Q and it is not yet produced by a previous service).

- Out_{s_i} represents the values of outputs of O_{s_i} produced by the execution of s_i. Then: $Out_{s_i} = \{(o, v) \mid o \in O_{s_i}$ and v is the value of output $o\}$, where $\forall (o, v) \in Out_{s_i}$, $v \neq NULL$, if s_i has been successfully executed; or $Out_{s_i} = \emptyset$, if s_i has not been executed.

To illustrate this definition, note that for service s_8 in Figure 4.11, one input has NULL value (that one produced by s_4) and the other has a value produced by s_5, while $Out_{s_8} = \emptyset$.

Definition 4.4.2. A Global Snapshot (GS) is a 3-tuple representing the set of data necessary to restart the execution of a CS_Q, denoted as GS_{CS_Q}=<LS_{CS_Q},G_{ck},$In_{G_{ck}}$>, where:

- $LS_{CS_Q} = \{L_{s_i} \mid s_i \in CS_Q\}$ is the union of all local snapshots of the CS;
- G_{ck} is the subgraph of the CS_Q that could not be executed; and
- $In_{G_{ck}}$ is the set of inputs needed to restart this subgraph G_{ck}.

For the CS_Q shown in Figure 4.11, the subgraph G_{ck} that could not be executed is depicted in red. $In_{G_{ck}}$, the information needed to restart this subgraph is In_{s_4}, which is equal to Out_{s_1}, and In_{s_8}, with $In_{s_8} = Out_{s_5} \cup \{(j, NULL) \mid j \in O_{s_4}\}$. In short, $In_{G_{ck}} = \{Out_{s_1} \mid Out_{s_1} = In_{s_4}, Out_{s_5} \cup \{(j, NULL) \mid j \in O_{s_4}\}\}$.

Fuzzy atomicity and recovery strategy

From the user's perspective, submitting a query $Q = (I_Q, O_Q)$ results in one of three outcomes: (i) full output (O_Q) is received, (ii) no or minimal output if failures are unrecoverable ("nothing" or "(almost) nothing"), or (iii) partial output ("something") when checkpointing is used. Our approach incorporates user-defined *acceptable fuzzy atomicity*, specified either as a minimum percentage of required outputs (*e.g.*, 66% of the total outputs) or by marking outputs as essential (*e.g.*, an e-book) or optional (*e.g.*, a recommended list). Based on this preference, the system can adaptively select an appropriate recovery strategy, enabling a flexible, self-adaptive *fuzzy*"all-or-nothing" execution.

We next introduce our model for measuring fuzzy atomicity and the algorithm that selects the most suitable recovery method based on the system state and user preferences (*acceptable fuzzy atomicity*).

Fuzzy atomicity model. To consider user preferences, the definition of a query is redefined as follows.

Definition 4.4.3. A Query Q can be expressed as $Q = (I_Q, O_Q, f)$ or $Q = (I_Q, O_Q = Ost_Q \cup Oop_Q)$, where I_Q is the set of input attributes (whose values are provided by the user); O_Q is the set of output attributes needed by the user (whose values have to be produced); f is the acceptable fuzzy atomicity expressed as a percentage of the minimum desired outputs; Ost_Q is the set of output attributes strictly needed by the user; and Oop_Q is the set of optional output attributes that the user expects. Note that $O_Q = Ost_Q \cup Oop_Q$ and $Ost_Q \bigcap Oop_Q = \emptyset$.

Note that, if users do not want to relax atomicity, they can express $f = 1$ or indicate all their desired outputs as strict.

Before presenting our model, we outline key assumptions that support the definition of the proposed *fuzzy atomicity* property:

1. The system S consists of data managed by the services composing CS_Q, where each component service ensures the ACID properties over its local data,
2. Before executing CS_Q, the system is in a consistent state, denoted $S_{initial}$, where all component services maintain local consistency,
3. Composite service CS_Q is free from deadlocks and compliant with transactional rules (Table 2.1),
4. Execution of each component service, as well as CS_Q as a whole, complies with ACID properties,
5. Each component service s_i is transactional. If s_i has a compensation service s'_i, it also ensures consistency even under failure. Let O'_{s_i} denote the outputs produced by s'_i, and let CS'_Q be the set of compensating services for the compensatable components of CS_Q.

Definition 4.4.4 specifies what constitutes a *correct execution* required to formally define the fuzzy atomicity property.

Definition 4.4.4. Given a system S in a consistent initial state ($S_{initial}$), the execution of a composite service, CS_Q, is successful iff after its execution the final state of S (S_{final}) is also consistent (meaning that all components of CS_Q have been successfully executed or compensated).

When a composite service CS_Q is executed in response to a query $Q = (I_Q, O_Q, f)$ or $Q = (I_Q, O_Q = Ost_Q \cup Oop_Q)$, three execution states can be identified: *Internal Execution State* that refers to the state of CS_Q based on the execution status of its component services (Definition 4.4.5), *System State* that represents changes made to the system S through CS_Q (where each successfully executed service or its compensation may modify S, Definition 4.4.6), and *User Execution State* that corresponds to the user's perspective, reflecting the portion of desired outputs (O_Q) received (Definition 4.4.7).

Definition 4.4.5. A composite service CS_Q that satisfies a query $Q = (I_Q, O_Q, f)$ or $Q = (I_Q, O_Q = Ost_Q \cup Oop_Q)$ can be in one of the following internal execution states:

1. CS_{state}=I: initial, when CS_Q starts its execution.
2. CS_{state}=E: executing, when at least one of its component services is still running.
3. CS_{state}=SF: successfully finished, when all its components services are successfully finished.
4. CS_{state}=C: compensating, when a compensation of at least one of its components is started.
5. CS_{state}=FC: finished compensated, when all successfully finished services were compensated.

6. CS_{state}=SB: stand-by, when some services are successfully finished, while others were not executed because of a failure. In this case, a Global Snapshot is obtained.

The state of the system depends on the internal execution state of the CS_Q and on the produced outputs. Below, we present state changes defined using the following standard rule form, where state R is reached iff conditions C_1 to C_n are verified :

$$\frac{C_1, \ldots C_n}{\text{Reached state } R}$$

Definition 4.4.6. The system S can be in one of the following consistent states ($System_{state}$) during the execution of a CS_Q that satisfies a query $Q = (I_Q, O_Q, f)$ or $Q = (I_Q, O_Q = Ost_Q \cup Oop_Q)$:

1. *Initial state:*

$$\frac{CS_{state} = I}{S_{initial}}$$

2. *Partial state:*

$$\frac{CS_{state} = E \text{ and } \left| \bigcup_{s_i \in CS_Q} O_{s_i} \right| \geq 0}{S_{partial}}$$

3. *Final with success:*

$$\frac{CS_{state} = SF \text{ and } \left(\bigcup_{s_i \in CS_Q} O_{s_i} \supseteq O_Q \right)}{S_{final}}$$

4. *Compensating state:*

$$\frac{CS_{state} = C \text{ and } \left| \bigcup_{s_i \in CS_Q} O_{s_i} \right| \geq 0 \text{ and } \left| \bigcup_{s'_i \in CS'_Q} O_{s'_i} \right| \geq 0}{S_{partial}}$$

5. *Finished compensated state:*

$$\frac{CS_{state} = FC \text{ and } \left| \bigcup_{s_i \in CS_Q} O_{s_i} \right| \geq 0 \text{ and } \left| \bigcup_{s'_i \in CS'_Q} O_{s'_i} \right| > 0}{S_{final}}$$

6. *Stand-by state:*

$$\frac{CS_{state} = SB \text{ and } \left(\left(\bigcup_{s_i \in CS_Q} O_{s_i} \cap O_Q \neq \emptyset \right) \text{ or } \left(\bigcup_{s_i \in CS_Q} O_{s_i} \cap Ost_Q = Ost_Q \right) \right)}{S_{standby}}$$

Definition 4.4.7. For a query $Q = (I_Q, O_Q, f)$ or $Q = (I_Q, O_Q = Ost_Q \cup Oop_Q)$, its corresponding CS_Q can be in one of the following execution states from the point of view of the user:

1. $userCS_{state}$=I: initial, when the CS_Q is submitted for execution.
2. $userCS_{state}$=E: executing, when user might have partial responses. Let O_P be the set of these partial responses obtained by the user:
   a) if CS_{state}=E, then $O_P = \left(\bigcup_{s_i \in CS_Q} O_{s_i} \cap O_Q \right)$ and $|O_p| \geq 0$;
   b) if CS_{state}=C, then $O_P = \left(\left(\bigcup_{s_i \in CS_Q} O_{s_i} \cap O_Q \right) \cup \bigcup_{s'_i \in CS'_Q} O_{s'_i} \right)$ with $\left| \bigcup_{s_i \in CS_Q} O_{s_i} \right| \geq 0$ and $\left| \bigcup_{s'_i \in CS'_Q} O_{s'_i} \right| \geq 0$.
3. $userCS_{state}$=F: finished, when user has final response. Let O_F be the set of responses obtained by the user after the completed execution of CS_Q:
   a) if CS_{state}=SF, $O_F = O_Q$; the user has 100% of his desired outputs;
   b) if CS_{state}=FC, $O_F \neq O_Q$; the user has x% of his desired outputs $(0 \leq x < 100)$;
   c) if CS_{state}=SB, $O_F \subset O_Q$ or $O_F = Ost_Q$; the user has y% of his desired outputs or at least its mandatory outputs $(0 < y < 100)$, and a Global Snapshot exits that can be resubmitted later.

From the point of view of users, fuzzy atomicity is important in terms of the number of its desired outputs. Following definition formally describes the degree of outputs, related with O_Q, generated by the execution of a composite service.

Definition 4.4.8. Given CS_Q that satisfies a query $Q = (I_Q, O_Q, f)$ or $Q = (I_Q, O_Q = Ost_Q \cup Oop_Q)$ and the set of outputs generated by the execution of CS_Q that are in O_Q, denoted as O_F $(O_F \subseteq O_Q)$, the Degree of Outputs Dependency generated by a CS ($DegreeOutputG_{CS}$) is the percentage of O_Q generated by the execution of CS_Q, expressed as $DegreeOutputG_{CS_Q} = |O_F|/|O_Q|$.

This degree is equal 1 (representing 100% of O_Q), if the internal execution state of CS_Q is successfully finished (*i.e.*, CS_{state}=SF), in which case $O_F = O_Q$. This degree represents a value less than 1, if the internal execution state of CS_Q is finished compensated (*i.e.*, CS_{state}=FC), or executing (*i.e.*, CS_{state}=E), or stand-by (*i.e.*, CS_{state}=SB), in which cases $O_F = \emptyset$ or $O_F \subset O_Q$ or $O_F = Ost_Q$.

Recovery strategy algorithm. Our approach enables fault-tolerant execution by selecting recovery strategies based on fuzzy atomicity. A dedicated algorithm, in cooperation with the *execution engine* (see Figure 4.12), monitors service and system states, applies recovery when needed, and calculates fuzzy atomicity. The algorithm monitors the composite service state (CS_{state}), updates the system state, decides recovery strategies upon failures, and calculates both system and user fuzzy atomicity $System_{fatomicity}$ and $User_{fatomicity}$. It takes as input the query Q, the composite service graph CS_Q, and the retry limit N_{tries}. During normal execution, it tracks progress and computes atomicity metrics. When all services succeed, the user receives 100% of O_Q.

If a service fails, function `Get_Possible_Final_Outputs()` estimates the attainable outputs O'_F from unaffected services. The ratio $DegreeOutputG_{CS_Q} = O'_F/O_Q$ is compared to the user's fuzzy threshold to guide recovery. If the acceptable fuzzy measure is not met, then either the failed service is retriable then it is retried until success, or the failed service is not retriable then backward recovery via compensation is

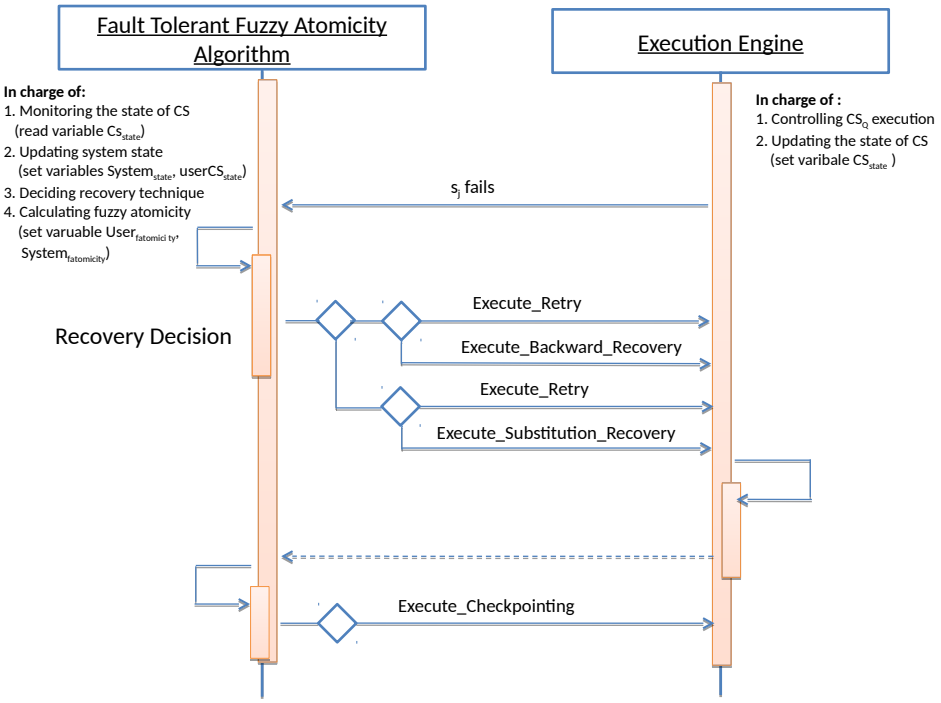


Figure 4.12: Execution engine and fuzzy atomicitybased recovery strategy cooperation protocol

applied. All executed services are rolled back, resulting in a state where "(almost)nothing" is preserved. If the acceptable fuzzy measure is met, then if the failed service is retriable then it is retried up to N_{tries}, otherwise, if a substitute exists, it is invoked. In case of unsuccess (*i.e.*, when failed service has not been successfully executed after N_{tries} times or when its substitute has not been successfully executed or when it not has a substitute), the algorithm executes checkpointing. Execution continues with unaffected services, and the user may still receive some desired outputs.

The algorithm also supports relaxing the strict retriable condition. In real-world systems, failures may not be fixed immediately. Therefore, after a limited number of retries, checkpointing allows deferred execution. The user can resume from the checkpoint to eventually obtain all desired outputs. At the end of CS execution, successful or not, the algorithm returns the final $System_{fatomicity}$, $User_{fatomicity}$, and the checkpointed CS_Q, if applicable.

4.5 A Glimpse on Further Contributions

Concurrency Control of Composite Service Execution in Peer-to-Peer Networks

The content of this section is about El Haddad et al. [81] work carried out in collaboration with colleagues from Université Paris Dauphine-PSL

In this work, we addressed concurrency management during the execution of component services in peer-to-peer networks to enhance reliability. In such networks, services running on independent peers are orchestrated according to predefined transactional requirements. We introduced a transactional execution model that builds on the transactional properties of these services. It is a hierarchical model based on open nested transactions, where a root transaction can initiate multiple sub-transactions. Leaf sub-transactions are treated as standard flat transactions and correspond to service invocations, while non-leaf sub-transactions manage control flow and determine the execution of their children. To manage concurrency, we used an optimistic control strategy through a decentralized serialization graph. A leaf transaction constructs its graph from log files sent by the peers whose services were invoked. A non-leaf transaction derives its graph from those of its children. In both cases, the graph is updated by merging

received data and replacing transaction identifiers with those of sibling transactions within the same hierarchical level. Serialization graphs are propagated bottom-up—from peers to leaves, and then through intermediate transactions up to the root. If a cycle is detected in the graph, two scenarios may arise: (i) if the transaction logic is a conjunction, the failure of any child leads to the failure of the transaction; (ii) if the logic is a disjunction, a victim child is selected to abort in order to break the cycle, following an `Error!` message from the parent. This decentralized mechanism ensures globally correct execution of composite web services through coordination among dependent sub-transactions and their associated peers.

4.6 Summary

In this chapter, we addressed composite services execution consistency and reliability in the presence of failures. After reviewing related work, we proposed a coloured Petri net based execution framework that supports both forward and backward recovery. Forward recovery through retry or substitution of failed services, and backward recovery via compensation when forward strategies are not applicable. To enable more flexible and user-centric recovery strategies, we introduced the notion of fuzzy atomicity, which relaxes the classical "all-or-nothing" model of service execution. Then, we proposed a recovery approach combining semantic compensation and checkpointing to allow partial results to be accepted by users when full recovery is infeasible. We concluded this chapter with a brief overview of work on concurrency control during service composition execution.

CHAPTER 5

TRUSTWORTHINESS IN SERVICE SELECTION AND COMPOSITION

In this chapter, we tackle the challenge of trustworthiness in service compositions. We provided approaches to trust-based service management in distributed, social service environments. First, we defined a trust model as a compositional concept that includes social, expert, recommender and cooperation-based component. Then, we proposed a distributed trust computation mechanism for service discovery and selection, and an adaptive coalition formation approach for service composition. This chapter highlights contributions carried out during the doctoral thesis of Amine Louati, that I have co-supervised with a colleague from Université Paris Dauphine-PSL, and presented in Louati et al. [152, 153, 155, 154, 156] *and Louati et al.* [157].

5.1 Motivations

In Chapters 3 and 4, we explored approaches that ensure both efficiency and reliability in service compositions through objective quality attributes. Quantitative QoS metrics were used to optimize performance, while qualitative transactional properties aimed to guarantee consistency and reliability. However, these objective attributes overlook an important dimension of service evaluation: subjective user perceptions. In practice, users rely heavily on reviews, ratings, and personal experiences when selecting services. Subjective attributes reflect individual satisfaction and trust factors that have become increasingly relevant in the Web 2.0 era. A service might perform well technically but still receive poor ratings due to weak customer support, for instance. Such discrepancies underscore the importance of integrating user-centric, *subjective attributes* into service selection and composition.

Furthermore, the approaches in the earlier chapters (Chapter 3 and Chapter 4) assess services in isolation, ignoring the characteristics of the service providers themselves. In real-world scenarios where compositions often involve third-party or unverified providers, it is difficult to determine a priori whether a service, and its provider, can be *trusted* to carry out the required functionality.

Finally, the rise of social networks and peer-to-peer platforms has introduced a *social dimension* to service provisioning. Users now act as both consumers and providers in decentralized, socially embedded ecosystems (Dustdar et al. [75]). Service selection increasingly depends on social indicators such as peer recommendations, community ratings, and previous experiences (Chard et al. [58]). In this context, information including mutual relationships, and historical behaviour, plays a central role in shaping user preferences and decision-making.

This chapter goes beyond the previous chapters by evaluating not just services but also their providers, and by integrating social trust factors into service discovery, selection and composition. In service-oriented computing, trust is typically associated with a service's non-functional description, often expressed through QoS attributes (Li et al. [142], Vu et al. [230], Xu et al. [242]). However, numerous studies (Al-Sharawneh [6], Bansal et al. [22], Billhardt et al. [32], da Silva and Zisman [67], Li et al. [141], Maaradji et al. [163], Wang and Vassileva [235]) have shown that relying solely on QoS metrics is insufficient for distinguish-

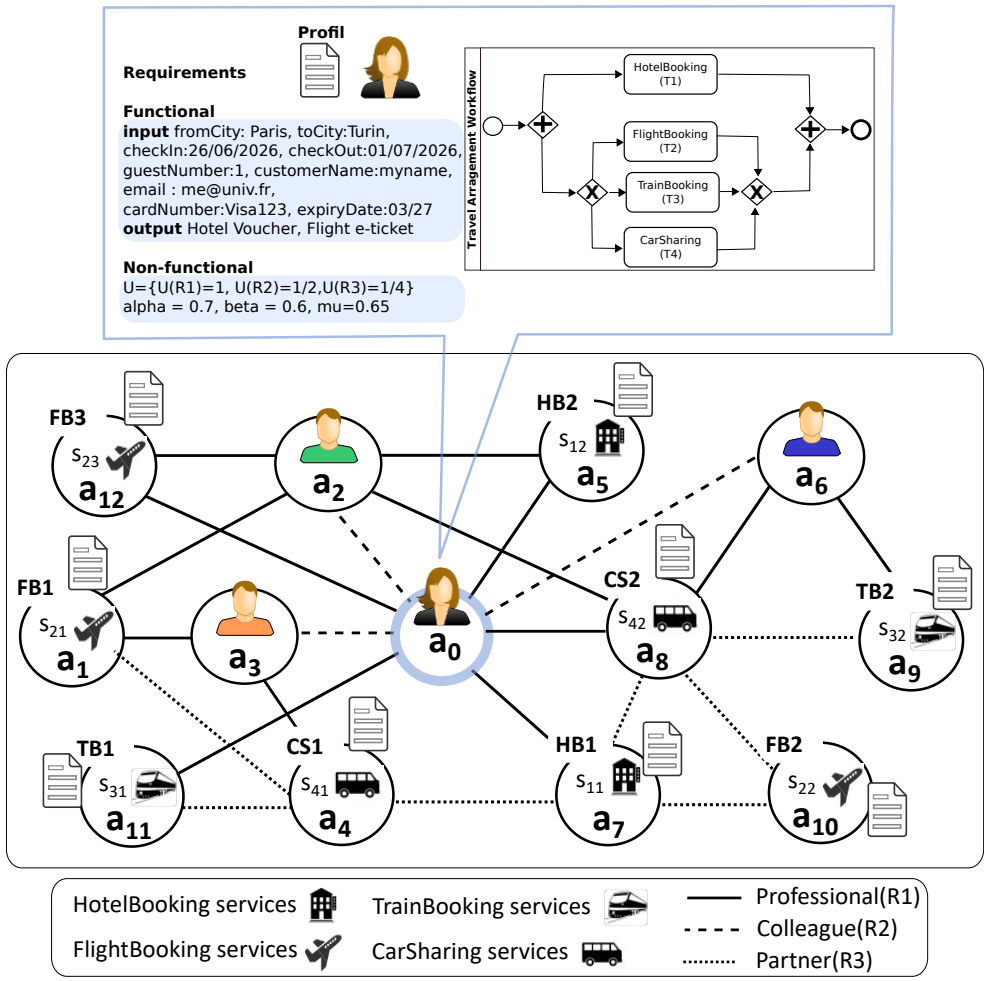


Figure 5.1: Multi-relational social network and TravelArrangement booking motivating example

Table 5.1: Available candidate component services for TravelArrangement

Task	Service	Response Time	Execution Cost	Transactional Property	Provider
HotelBooking	s_{11}	15mn	1€	–	HB1
(T1)	s_{12}	10mn	3€	*compensatable*	HB2
FlightBooking	s_{21}	20mn	1€	–	FB1
(T2)	s_{22}	10mn	10€	*compensatable*	FB2
	s_{23}	8mn	5€	*retriable*	FB3
TrainBooking	s_{31}	20mn	5€	–	TB1
(T3)	s_{32}	18mn	8€	–	TB2
CarSharing	s_{41}	5mn	2€	*compensatable*	CS1
(T4)	s_{42}	12mn	15€	–	CS2

ing between trustworthy and untrustworthy service providers or for assessing the reliability of services. A robust computational model of trust must instead mirror how trust is built in human societies. For example, a service provider with whom a requester shares a history of successful interactions, mutual social connections, or strong recommendations is more likely to be trusted, even in the absence of optimal QoS guarantees, as the following example illustrates.

Motivating example. After securing accommodation and purchasing a flight ticket to Turin, the researcher submits her travel request for department approval. Suppose the department director rejects the request, citing a policy to reduce CO_2 emissions and recommending train or car travel instead. Suppose the researcher prefers to travel by car and considers using a CarSharing service. As illustrated in Figure 5.1 and Table 5.1, she have to choose between two services: s_{41} provided by $CS1$ and s_{42} provided by $CS2$.

✗ If the researcher relies solely on QoS and transactional attributes, she would select service s_{41} provided by $CS1$, as it is faster, cheaper, and compensatable.

✓ However, if she takes into account her past experience with both providers $CS1$ and $CS2$, and if she prefers a well-known provider by consulting her social network illustrated in Figure 5.1, and if she values the recommendation of an Italian colleague with prior experience as a passenger (a_3 with s_{41} compared to a_2 or a_6 with s_{42}), she could opt for s_{42} offered by $CS2$.

Multiple studies (Li et al. [141], Liu and Wang [148], Maaradji et al. [163], Wang and Vassileva [235]) have shown that trustworthiness is a key factor in effective service selection, ensuring alignment with requester expectations. Liu and Wang [148] define trust as "the belief of one participant in another, based on their interactions, in the extent to which the future action to be performed by the latter will lead to an expected outcome". Several authors have proposed incorporating a societal perspective into traditional service discovery (Al-Sharawneh [6], Billhardt et al. [32], Li et al. [141], Liu and Wang [148]). This perspective draws on trust information from the trustor's social network and prior interactions. Sabater-Mir and Vercouter [199] identified three key sources of trust: (i) direct experience from past interactions between the trustor and trustee, (ii) communicated experience from third parties, and (iii) social information, which includes semantic or structural knowledge useful for evaluating trust. Beyond computer science, trust has been studied in philosophy (Perlman and Fehr[191]), socio-psychology (Berscheid and Ries [26], Luhmann [158]), and economics (Adler [4]). These fields showed that trust is multi-faceted (Abdul-Rahman and Hailes [2], Li and Wang [140]), and each component contributes uniquely to assessing a service's trustworthiness.

This chapter introduces trust-based multi-agent approaches that leverage social network structures and user interactions for service discovery, selection, and composition. Trust is modeled using three components: social credibility, expertise, and recommendation, derived from social network analysis and service quality metrics. High social credibility implies strong awareness of reliable providers and recommenders; expertise reflects high service performance; and recommendation indicates the trustworthiness of third-party endorsements. We also propose a decentralized request propagation mechanism and a dynamic coalition formation process for service composition, introducing a new trust dimension: trust in cooperation.

The rest of the chapter is organized as follows. In Section 5.2, we outline state-of-the-art research work for trustworthy service discovery, selection and composition, and summarizes our contributions in this research line. In Section 5.3, we present our trust model and our trustworthy-based multi-agent approach for service discovery and selection. In Section 5.4, we present our approach for service composition based on trust and coalition formation. In the last section, we conclude the chapter by summarizing our findings.

5.2 State-of-the-art and Contributions

In this section, we review trust-based approaches for service discovery, selection, and composition.

Trust-driven Service Discovery and Selection. To identify trustworthy service providers, numerous approaches have explored trust in expertise through reputation, typically by analyzing past user-service interactions (Billhardt et al. [32], Lalanne et al. [134], Li et al. [142], Vu et al. [230], Xu et al. [242]). Vu et al. [230] and Xu et al. [242] proposed QoS-aware frameworks that incorporate user feedback and reputation management into distributed service discovery. Billhardt et al. [32] introduced experience-based selection, estimating trust from similar services in the absence of direct history, while Lalanne et al. [134]

emphasized perceived quality and end-user satisfaction. Li et al. [142] relied on user feedback (*e.g.*, ratings and comments) to compute service reputation. Decentralized reputation models such as Fire (Huynh et al. [117]) and Regret (Sabater and Sierra [198]) allow agents to assess reputation autonomously. Regret uses three trust dimensions: individual (direct experience), social (group opinion), and ontological (semantic context), while Fire considers direct trust, role-based trust, witness reports, and certified reputation. Yu and Singh [248] formalized trust propagation to prevent interactions with unreliable services. Other works (Day and Deters [68], Kalepu et al. [127], Yu and Lin [252]) used third-party monitoring of QoS metrics (availability, accuracy, execution time, cost, and bandwidth) to assess service behavior. Billionniere et al. [33] expanded this with context-specific attributes.

As noted in Huynh et al. [117]) and Sabater and Sierra [198], trust evaluation should also incorporate social relationships and agent roles. This includes metrics like degree centrality (Bansal et al. [22]), social proximity (Maaradji et al. [163]), combined prestige-centrality (Sierra and Debenham [206], and multi-constraint social models (Liu and Wang [148]). However, many of these approaches often ignored semantic information and typically consider social networks with only a single relationship type.

When direct interactions are lacking, referral systems enable agents to cooperate by giving, pursuing, and evaluating recommendations (Yu and Singh [249]). Trust inference methods in social networks (Golbeck [101], Hang et al. [113], Liu et al [149], Wang et al. [233], Wang and Singh [234]) aim to generate personalized recommendations by aggregating opinions from the trust network. Golbeck [101] showed that trust-aware recommendation techniques, such as the FilmTrust system, yield more accurate and personalized results than classical collaborative filtering. Other techniques include leader-follower strategies (Al-Sharawneh and Williams [7]), and trust propagation (Massa and Avesani [167], Neville et al. [180]). While effective, these methods rely primarily on subjective user input, potentially limiting reliability.

Trust-driven Service Composition. Early work framed service composition as a planning problem, relying on centralized orchestration engines to match user needs with service capabilities (Paik et al. [188], Ponnekanti et al. [193], Sirin et al. [209], Tong et al. [220], Xu et al. [240]). These approaches modeled composition as a graph search task, where inputs and outputs define initial and goal states, and services act as operators enabling state transitions. While structured, these methods depend on centralized registries and semantic matching, limiting scalability and expressiveness regarding user preferences. Subsequent approaches shifted toward agent-based coordination for dynamic and distributed composition. Choreography models enabled agents to reason about capabilities and coordinate tasks (Charif et al. [59]), though they prioritized user constraints over agent autonomy. Case-based reasoning (Siala et al. [205]) and context-aware agents (Maamar et al. [160]) introduced more flexibility, but often overlooked social factors like trust, cooperation, and compatibility. In most of these models, agents acted more as coordinators than as collaborative participants in an organizational structure. Parallel research explored coalition formation to support cooperation among self-interested agents in distributed environments (Asl et al. [18], Ermolayev et al. [85], Muller et al. [179], Tong et al. [219], Bourdon et al. [38], Griffiths and Luck [104]). Asl et al. [18] demonstrated that coalitions can yield stable communities optimizing both individual and collective utility. However, many approaches like Ermolayev et al. [85], Muller et al. [179], and Tong et al. [219], neglected trust in partner evaluation. Griffiths and Luck [104] incorporated trust in admission decisions but lacked mechanisms for members to accept or reject newcomers. While Bourdon et al. [38] proposed a trust-based, provider-centric coalition formation but lack mechanisms for agents to leave or switch when dissatisfied.

Positioning of contributions.

Our primary objective in this line of research was to develop comprehensive trust-based mechanisms for service discovery, selection, and composition in service-oriented environments. While many existing approaches focus narrowly on reputation-based trust derived from past interactions (*e.g.*, Billhardt et al. [32], Lalanne et al. [134], Li et al. [142], Vu et al. [230], Xu et al. [242]), our work extends trust modeling to a multi-dimensional perspective that incorporates social relationships and recommendation credibility.

Initially, in a line of work Louati et al. [152, 153, 155, 154, 156] (Section 5.3), we introduced a trust model encompassing three dimensions: societal, expertise, and recommendation. This multi-dimensional approach addresses the limitation noted by Huynh et al. [117] and Sabater and Sierra [198] that trust eval-

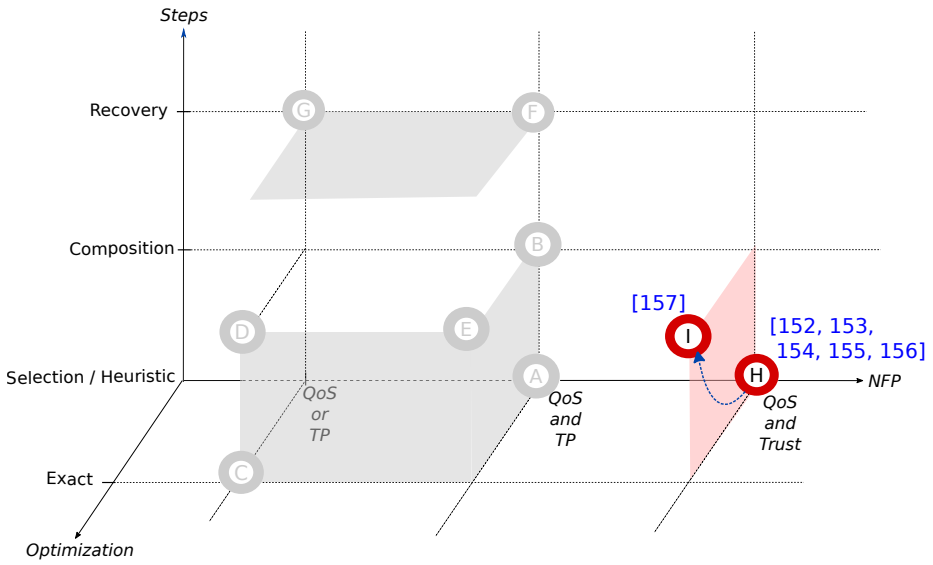


Figure 5.2: Contributions to Trustworthiness in Service Selection and Composition

uation should incorporate social relationships and agent roles beyond mere service reputation. Societal dimension assesses whether a service provider merits engagement prior to service utilization, extending beyond the fixed QoS attributes commonly used in approaches like Day and Deters [68], and Kalepu et al. [127]. Expertise dimension evaluates the reliability and expected performance of the service itself, while recommendation dimension determines the trustworthiness of agents providing recommendations and the credibility of those recommendations. In our earlier work Louati et al. [152], we concentrated on discovering trustworthy service providers within a Multi-Relation Social Network (MRSN) centered on a service requester. Unlike existing social trust approaches that rely on single relationship types or limited social metrics (*e.g.*, Bansal et al. [22], Maaradji et al. [163]), we computed social trust by aggregating social position, social proximity, and social similarity measures that incorporate both semantic and structural information extracted from the requester's MRSN. Semantic information encompasses service requester and provider profiles along with their interactions, while structural information includes the positioning of service providers within the MRSN graph. The centrality measure, fundamental in social network analysis, contributes significantly to social trust computation, as demonstrated by Bansal et al. [22]. Our approach generates a Trust-Relation Social Network (TRSN) that is requester-centered and based on a single relation, the social trust relation, which effectively filters and ranks service providers. We later proposed, in Louati et al. [153], a distributed approach for trust-based service discovery in social networks enabling agents to propagate service queries and evaluate providers based on sociability and expertise. Drawing from referral systems (Golbeck [101], and Hang et al. [113]), our model identifies not only trustworthy providers but also effective recommenders—agents who, while not offering services themselves, contribute valuable referrals. After, in Louati et al. [155], we distributed the search process by designing a decentralized query propagation mechanism aligned with the social network topology, eliminating the need for central coordinators as in Yu and Singh [249]. Agents collaborate to locate trustworthy recommenders and providers offering quality services based on distributed knowledge of their network acquaintances' expertise. Later, in Louati et al. [154], we proposed a multilevel agent-based framework that integrates the three trust dimensions (sociability, expertise, and recommendation) into the selection process. Unlike prior ranking or trust propagation methods (*e.g.*, Massa and Avesani [167], Hang and Singh [112]), our model supports multiple trust dimensions simultaneously. Lastly, in Louati et al. [156], we developed a distributed algorithm for service discovery and selection using a probabilistic trust propagation model via referral systems. Unlike prior trust inference methods that focused mainly on aggregated opinions for personalized recommendations (*e.g.*, Golbeck [101], Liu et al. [149], Wang et al. [233]), our approach distinguishes between the roles of intermediate agents (recommenders and providers) along referral chains, enabling more nuanced trust evaluation than traditional collaborative filtering.

Building upon these foundations, in Louati et al.[157] (Section 5.4), we proposed a broker-based multi-agent framework for dynamic, trust-aware service composition through coalition formation. While previous coalition-based models explored cooperative behavior among agents (Asl et al. [18], Ermolayev et al. [85], Muller et al. [179]), they often lacked robust partner evaluation or mechanisms for adaptation. Our model integrates trust assessment (trust in cooperation) prior to collaboration and preserves agent autonomy in partner selection, addressing gaps in works like Griffiths and Luck [104]. Furthermore, unlike static coalition models, our framework allows dissatisfied members to leave and supports dynamic restructuring, overcoming the rigidity identified in Bourdon et al. [38].

5.3 Trust-driven Service Discovery and Selection

The content of this section is adapted from Louati et al. [152, 153, 155, 154, 156] work carried out during the Ph.D. thesis of Amine Louati that I have co-supervised with a collegue from Université Paris Dauphine-PSL

Overview

In this work, we modeled service discovery and selection using a multi-agent system, where agents represent requesters, providers, and intermediaries within a requester's social network. Agent interactions occur through message exchanges.

We designed a trust model to (1) help agents identify trustworthy providers, and (2) support distributed decision-making. The model consists of three components: *trust in sociability* (relevance of providers and recommenders), *trust in expertise* (service quality), and *trust in recommendation* (reliability of opinions). Agents use these trust metrics during service discovery to guide interactions. In cases without prior interactions, a decentralized referral system where agents cooperate by sharing and evaluating referrals is used. Trust between non-adjacent agents is inferred using a probabilistic model, adapted from Wang et al. [233], which considers the roles (recommender or provider) of intermediaries in the trust path. Semantic information is integrated by incorporating agent profiles and relationship types into the trust model, improving its expressiveness. We also proposed a distributed algorithm for service discovery and trust propagation through referrals.

Our assumptions include: (i) agents are cooperative and share experiences, (ii) each agent only knows its direct acquaintances, and (iii) decisions are decentralized. Agents have a finite set of services and act on behalf of their users. During discovery, agents can assume four roles: *Requester* that initiates discovery and forwards queries to trusted contacts; *Provider* that offers required services and may propagate queries; *Recommender* that identifies and forwards queries without offering required services; *Stopper* that halts propagation when no relevant knowledge is available, reducing overhead.

Model

We considered a multi-relation social network (MRSN) (Szell et al. [216]) modeled by a graph where nodes represent agents and, an edge between two agents indicates a symmetric social relationship between them (see Chapter 3, Definition 2.1.3). The neighborhood of an agent is defined as follows:

Definition 5.3.1. Given a $MRSN$ graph $G = (V, E)$, the neighborhood of an agent $a_k \in V$ w.r.t. a type of relationship $R_i \in \{R_1, R_2, ..., R_r\}$, denoted $N_{R_i}(a_k)$, is defined as $N_{R_i}(a_k) = \{a_j \in V \mid (a_k, a_j) \in E_i\}$.

In the MRSN, each agent a_k interacts with a subset of agents, called the social acquaintances SA_k. This set represents a_k's local view in the MRSN such as $SA_k = \bigcup_{R_i \in R} N_{R_i}(a_k)$.

A service is described in terms of functionality, inputs, outputs, and non-functional attribute values as in Chapter 3, Definition 2.2.1. A user communicates his needs by expressing a set of required services and his preferences over relationship types and trust threshold values as in Chapter 3, Definition 2.1.2. We used a deliberative architecture (Bryson et al. [41], Georgeff and Lansky [96]) that enables agents to evaluate trust before engaging in interactions. As defined below, the architecture comprises four modules

(Reasoning $\mathcal{RM}$, Trust $\mathcal{TM}$, Control $\mathcal{CM}$, and Interaction $\mathcal{IM}$) along with two repositories (Beliefs $\mathcal{BR}$ and Goals $\mathcal{GR}$).

Definition 5.3.2. An agent a_k is defined as a 6-components structure $< \mathcal{BR}, \mathcal{GR}, \mathcal{RM}, \mathcal{TM}, \mathcal{CM}, \mathcal{IM} >$ with:

- $\mathcal{BR} =< Pr_k, S_k, PIT_k >$, the belief repository with Pr_k a profile consisting of a set of items structured into a set of fields each containing one or several values, $S_k = \{s_{k1}, \ldots, s_{km_k}\}$ a set of offered services and PIT_k a Personal Interaction Table. Each record in PIT_k contains the following elements: an acquaintance agent $a_j \in SA_k$, the profile Pr_j of a_j, the social acquaintances set SA_j of a_j and the set of services S_j provided by a_j. This information is acquired through interactions among agents.
- $\mathcal{GR}$, the goal repository which encompasses the required services needed to solve user's query.
- $\mathcal{RM}$, the reasoning module representing the matching function. A matching function between a service $s_{kl} \in S_k$ and a service $s \in F$ is defined as follows:
$matching(s, s_{kl}) = True \Leftrightarrow (s_{kl}.in \subseteq s.in) \wedge (s.out \subseteq s_{kl}.out) \wedge (s_{kl}.f \equiv s.f)$.
- $\mathcal{TM}$, the trust module which computes all trust measures that an agent a_k has with its social acquaintances before interacting with them.
- $\mathcal{CM}$, the control module which manages agent's behavior and guides its decision-making in the discovery and selection process.
- $\mathcal{IM}$, the interaction module which structures the messages built by the agent a_k and handles the received ones.

Trust Model

Trustworthiness in providers and their offered services is built upon three measures: trust in sociability (ST), trust in expertise (ET), and trust in recommendation (RT).

Trust in Sociability (ST). It measures the social trust that agent a_k places in agent a_j, based on information from the MRSN including the graph structure, agent profiles (*e.g.*, personal information and interests), and relationship types. From this data, three metrics are derived: *social position* (*SPo*), *social proximity* (*SPr*), and *social similarity* (*SSi*). We first describe these metrics and then explain how they are combined to compute trust in sociability.

1. *Social Position Measure* (*SPo*). It is computed using the centrality degree of agent a_j, reflecting its social power within the network (Bansal et al. [22]). It considers not only the number of relationships but also their types $R_i, 1 \leq i \leq n$, as defined below:

$$SPo(a_j) = \sum_{i=1}^{r} \sum_{a_l \in SA_j} U(\rho((a_j, a_l))) \times b^i(a_j, a_l) \tag{5.1}$$

   where $b^i(a_j, a_l) = 1$ iff a_j and a_l are directly connected with an edge of relationship R_i, 0 otherwise.

2. *Social Proximity Measure* (*SPr*). It is defined as the average cost of a path between two agents in the graph. Let $path = (a_k, \ldots, a_j)$ be a path of length d between an agent a_k and an agent a_j, and $U(\rho((a_{l-1}, a_l)))$ be the cost of the edge $(a_{l-1}, a_l) \in path$, we defined SPr as follows:

$$SPr(a_k, a_j) = \frac{\sum_{l=1}^{d} U(\rho((a_{l-1}, a_l)))}{d} \tag{5.2}$$

3. *Social Similarity Measure* (*SSi*). It is computed based on the comparison between two agents of their profiles and their social acquaintance sets. $SSi(a_k, a_j)$ is an aggregation of two measures, namely, *Neighborhood Similarity* (*NS*) and *Profile Similarity* (*PS*).

   - *Neighborhood Similarity Measure* (*NS*). We defined neighborhood similarity to find links between agents based on their social acquaintances as follows:

   $$NS(a_k, a_j) = \sum_{i=1}^{|R|} U(\rho((a_{l-1}, a_l))) \times \delta^i(a_k, a_j) \tag{5.3}$$

   with $\delta^i(a_k, a_j) = \frac{1}{1+jac^i}$ where $jac^i = \frac{y_i+z_i}{x_i+y_i+z_i}$ is the *Jaccard distance* between a_k and a_j according to the relationship R_i such as $x_i = |N_{R_i}(a_k) \cap N_{R_i}(a_j)|$, $y_i = |N_{R_i}(a_k)| - x_i$, $z_i = |N_{R_i}(a_j)| - x_i$.
   - *Profile Similarity* (*PS*). An agent's profile[1] is not only characterized by its acquaintances, but also by a set of personal information and interests. We defined profile similarity as follows:

   $$PS(a_k, a_j) = \frac{1}{|I|} \times \sum_{i \in I} \beta_i \times S_i(a_k, a_j) \tag{5.4}$$

   where $S_i(a_k, a_j)$ is the similarity between the ith items of a_k and a_j using Burnaby measure [42], I is the set of items in profiles and β_i is the weight attributed to the item i with $\sum_{i \in I} \beta_i = 1$.

   The overall measure of social similarity, $SSi(a_k, a_j)$, between a requester agent a_k and an agent a_j is computed as the product of the two above measures:

   $$SSi(a_k, a_j) = NS(a_k, a_j) \times PS(a_k, a_j) \tag{5.5}$$

The overall trust in sociability, $ST(a_k, a_j)$, that an agent a_k has in agent a_j is then computed using a Simple Additive Weighting technique as follows:

$$ST(a_k, a_j) = \lambda_1 \times SPo'(a_j) + \lambda_2 \times SPr'(a_k, a_j) + \lambda_3 \times SSi'(a_k, a_j) \tag{5.6}$$

where $\lambda_t \in [0, 1]$ is the weight of the t-th social measure with $\sum_{t=1}^{3} \lambda_t = 1$. and $SPo'(a_j)$, $SPr'(a_k, a_j)$, and $SSi'(a_k, a_j)$) are the normalized values of $SPo'(a_j)$, $SPr'(a_k, a_j)$, and $SSi'(a_k, a_j)$) so they lies between 0 and 1.

Trust in Expertise (ET). It evaluates the QoS of a service. A reliable agent should be both socially trustworthy and sufficiently expert. Following Lalanne et al. [134], we have defined $ET(a_k, a_j, s_{jl})$, the trust agent a_k has in service s_{jl} provided by agent a_j, as the aggregation of three following metrics:

1. *Specialization* ($Sp(s_{jl})$): the percentage of successful uses of service s_{jl} compared to all services it offers. It is computed as presented in Chapter 2, Equation 2.1.

2. *Reliability* ($Re(s_{jl})$): the probability that a service s_{jl} is operational at the time of invocation. It is computed as presented in Chapter 2, Equation 2.2.

3. *Experience rating* ($Eval(a_k, s_{jl})$): the rating of the quality of service execution. After using s_{jl}, agent a_k gives an evaluation $\nu \in [0, 1]$ reflecting its experience. $Eval(a_k, s_{jl})$ is the average of the experience ratings of s_{jl} for n uses by a_k computed as presented in Chapter 2, Equation 2.3.

The overall trust in expertise, $ET(a_k, a_j, s_{jl})$, that an agent a_k have in a service s_{jl} offered by an agent a_j is computed as follows in Chapter 2, Equation 2.4.

[1] A profile consists of a set of items structured into a set of fields, each field containing one or several values (*e.g.*, gender=[female], music-likes=[folk, jazz, pop])

Trust in Recommendation (RT). According to Golbeck [101] and Berscheid and Reis [26].

we defined trust in recommendation (RT) as the trust value a_k assigns to a recommendation from a_j for service s_{pl} offered by provider a_p. It is estimated from past experiences with a_j during previous compositions and has two parts: objective part $[r_{kj}|s_{pl}] \in [0,1]$ which is the proportion of good recommendations made by a_j for s_{pl}), subjective part $[q_{kj}|s_{pl}] \in [0,1]$ which reflect a_k's satisfaction with a_j's recommendations for s_{pl}. Following Chen and Singh [61], we computed RT as:

$$RT(a_k, a_j, s_{pl}) = \begin{cases} 1 & \text{if } [r_{kj}|s_{pl}] = 0 \, or \, [q_{kj}|s_{pl}] = 0 \\ ([r_{kj}|s_{pl}] + 1)^{[q_{kj}|s_{pl}]} - 1 & \text{otherwise} \end{cases} \tag{5.7}$$

As noted above, $[r_{kj}|s_{pl}]$ is the ratio of services s_{pl} recommended by a_j and actually selected by a_k, among all a_j's recommendations for s_{pl} offered by provider a_p. Following Maamar et al. [162], it is defined as:

$$[r_{kj}|s_{pl}] = \begin{cases} 1 & \text{if} \quad Nbrec_{jk|s_{pl}} = 0 \\ \frac{Nbsel_{kj|s_{pl}}}{Nbrec_{jk|s_{pl}}} & \text{otherwise} \end{cases} \tag{5.8}$$

where $Nbrec_{jk|s_{pl}}$ is the number of times a_j recommended service s_{pl} (offered by a_p) to a_k, and $Nbsel_{kj|s_{pl}}$ is the number of times a_k selected s_{pl} in the composition.

However, $[q_{kj}|s_{pl}]$ measures a_k's satisfaction with a_j's recommendations for service s_{pl} offered by a_p. Given $Eval(a_k, s_{pl})$ (Chapter 2, Equation 2.3), the rating a_k assigns after executing s_{pl}—$[q_{kj}|s_{pl}]$ is the average of all ratings a_k has given to s_{pl} after successful executions:

$$[q_{kj}|s_{pl}] = \begin{cases} 1 & \text{if} \quad Nbsel_{kj|s_{pl}} = 0 \\ \frac{\sum_1^{Nbsel_{kj|s_{pl}}} Eval(a_k, s_{pl})}{Nbsel_{kj|s_{pl}}} & \text{otherwise} \end{cases} \tag{5.9}$$

Our approach

Our trust-based service discovery and selection approach, illustrated in Figure 5.3, involves three steps: service discovery, trust inference, and service selection. In the first step, our algorithm computes trust in sociability and recommendation, matches service functionality, and builds Trust-Relation Social Network (TRSN) a tree linking providers and recommenders, weighted by trust values. In the second step, trust values are propagated through the TRSN to build a Requester-Centered Social Network (RCSN), where each provider is evaluated based on inferred trust. In the third step, services are ranked by expertise trust, and only those exceeding a defined threshold are selected.

In our approach, sociability and recommendation trust guide discovery, while expertise trust ensures high-quality selection without excluding useful recommendations.

Step 1: Service Discovery. This step identifies trustworthy providers using a distributed trust-based breadth-first search over the MRSN graph, given a query $Q = (F, U, \alpha, \beta, \mu)$. The requester agent a_r initiates the process by setting its role, its distance $(= 0)$, and preparing an empty provider set $PSet_r$ to store discovered providers, services, and associated trust values. To begin, a_r collects updated information from its social acquaintances SA_r (*i.e.*, services, acquaintances, and profiles) through REQUEST/INFORM messages. It then computes *trust in sociability* and retains only those acquaintances exceeding threshold α in LTA_r. Trust thresholds α and β are dynamically scaled with the chain length to prioritize highly trusted distant agents. Queries are propagated only to trusted acquaintances who are either potential providers or reliable recommenders. *Recommendation trust* $f_\beta(dist_k)$ further reduces propagation to high-quality recommenders. To avoid cycles, agents update their distance only if the new value is shorter, set their parent, and records their children in the tree. This step then proceeds by checking for service matches. If the agent offers a required service (*i.e.*, a provider), it records it and propagates the query to trusted neighbors. If not, but it can recommend trusted providers (*i.e.*, a recommender), then it forwards the query; otherwise, propagation stops.

The result of this step is a Trust-Relation Social Network (TRSN) tree rooted at a_r, linking providers and recommenders based on service functionality and sociability.

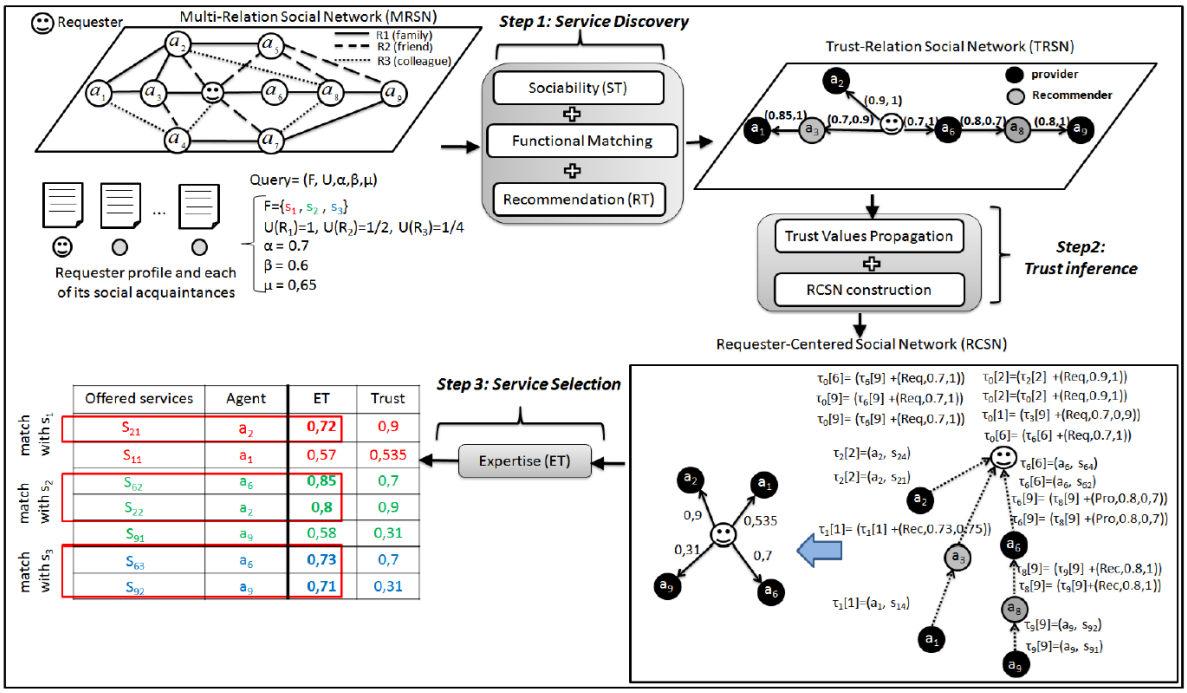


Figure 5.3: Trust-Based Service Discovery and Selection Approach

Step 2: Trust Inference. In social networks, an agent evaluates another's trustworthiness either through direct interactions or, in their absence, through a *trust inference mechanism* that aggregates trust values along paths between agents to produce a single evaluation.

Trust Values Propagation. The previous step produces a TRSN tree linking providers and recommenders, with edges weighted by *trust in sociability* and *trust in recommendation*. The requester a_r initiates the trust inference algorithm by sending INFORM messages (tagged with `infer`) to its children. Each agent forwards the message to its own children and, if it is a provider, returns a stack containing its identifier and matching service. When a parent receives a stack, it appends its role and trust values: $\{role_k, ST(a_k, a_j), RT(a_k, a_j)\}$, where $RT = 1$ if the child is a provider. The stack is then passed upward until it reaches a_r, which stores it in $PSet_r$. Figure 5.4 illustrates trust propagation from provider a_9 to requester a_0 within the TRSN in Figure 5.3.

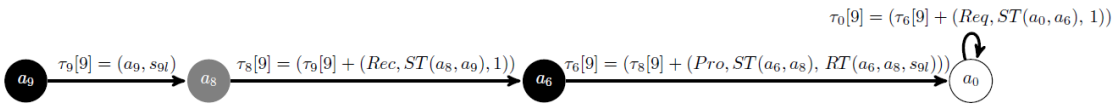


Figure 5.4: A provider-recommender chain

After propagation, $PSet_r$ contains, for each provider, the trust values (*i.e.*, sociability and recommendation) along its provider–recommender chain $chain = (a_r, a_k, a_{k+1}, \ldots, a_p)$. Trust between non-adjacent agents is inferred using the probabilistic model of Wang et al. [233], extended to include recommendation trust. Based on $PSet_r$ and the law of total probability, the inferred trust of requester a_r in provider a_p is given by:

$$Trust(a_r, a_p) = P(a_r) \times P(a_k|a_r) \times P(a_{k+1}|a_k) \times \ldots \times P(a_p|a_{p-1}) \tag{5.10}$$

where $P(a_r) = 1$ denotes the trustworthiness of a_r to itself. $P(a_k|a_r)$ denotes the trustworthiness of a_k from a_r's point of view. This is an aggregation of trust in sociability and trust in recommendation such as $P(a_k|a_r) = ST(a_r, a_k) \times RT(a_r, a_k)$ where, $RT(a_r, a_k) = 1$ if $role_k = Pro$.

Using $PSet_r$ set and Equation 5.10, a_r computes its trust in each discovered provider in TRSN, even in the absence of direct interaction.

RCSN Construction. This phase builds a network from the requester's perspective. Using the inferred trust values $Trust(a_r, a_p)$ for each discovered provider a_p, we construct the Requester-Centered Social Network (RCSN), a directed, weighted tree $G' = (V', E')$ where V' includes the requester and all providers, and E' consists of trust-weighted edges from a_r to each a_p. The computed values are stored in the structure $PSet_r^*$, as defined below:

$$\begin{aligned} \mathcal{F}_1 : \quad & PSet_r \rightarrow PSet_r^* \\ & \tau_r[p] \mapsto (\tau_r[p], Trust(a_r, a_p)) \end{aligned}$$

where $\mathcal{F}_1$ is a function that processes each stack $\tau_r[p]$ of a discovered provider a_p by appending its inferred trust value $Trust(a_r, a_p)$.

Step 3: Service Selection. In the previous steps, requester a_r identified a set of trustworthy providers. This step focuses on ranking them and selecting the most suitable ones for service execution. After computing *trust in expertise* (ET), a_r generates an ordered set $PSel_r$ using the following function:

$$\begin{aligned} \mathcal{F}_2 : \quad & PSet_r^* \rightarrow PSel_r \\ & (\tau_r[p], Trust(a_r, a_p)) \mapsto (\tau_r[p], Trust(a_r, a_p), ET(a_r, a_p, s_{pl})) \end{aligned}$$

where $\mathcal{F}_2$ processes each record in $PSet_r^*$ by adding the trust in expertise that a_r has in a_p. At the end of this step, for each required service, a_r selects providers whose expertise trust exceeds the threshold μ.

5.4 Trust-driven Service Composition

The content of this section is adapted from Louati et al. [157] work carried out during the Ph.D. thesis of Amine Louati that I have co-supervised with a collegue from Université Paris Dauphine-PSL

Overview

In this work, we proposed a broker-based multi-agent model for dynamic service composition, where self-interested agents, each offering services with associated QoS values, cooperate through coalition formation to fulfill complex user queries. To integrate the social dimension, a trust mechanism allow agents to evaluate potential partners before cooperating. To preserve providers autonomy, agents select coalition partners based on mutual trust, ensuring that only stable and agreeable collaborations are formed. If dissatisfied, agents may leave a coalition through a defined mechanism.

Our Coalition Formation Process (CFP) builds coalitions of trustworthy agents and is characterized by the following properties: (i) *incremental*, recruiting providers layer by layer from the requeste's trusted-relation social network; (ii) *dynamic*, allowing agents to autonomously join or leave coalitions based on trust and message exchange; and (iii) *overlapping*, enabling agents offering multiple services to participate in multiple coalitions.

Following Klusch and Sycara [129], our broker-based model defines three agent roles: (i) *candidate* that is a potential provider likely to join a coalition; (ii) *member* that is a provider already assigned to one or more coalitions; and (iii) *broker* that is a service requester who introduces candidates, manages interactions, and selects the coalition with the highest expertise trust when multiple options exist.

Model

Let F be the description domain of available functionalities. A user query $Q = \{f_1, f_2, \ldots, f_n \,|\, \forall 1 \leq i \leq n, f_i \in F\}$ is defined as a finite set of functionalities. Let $A = \{a_1, a_2, ..., a_s\}$ be the set of agents, each agent $a_k \in A$ is an autonomous entity defined as follows:

Definition 5.4.1. An agent $a_k =< S_k, Trust, CT, ET, \lambda\text{Inf}_k, \lambda\text{Sup}_k, \beta_k, Blist_k >$ where:

- $S_k = \{s_1, s_2, \ldots, s_{m_k}\}$ is the set of m_k offered services,
- Trust(a_k, a_j) is the trust that a_k has in an agent a_j in the social network which is defined as the aggregation of trust in sociability and trust in recommendation (for details see Section 5.3),
- $CT(a_k, a_j)$ is the trust in cooperation that a_k has in an agent a_j,
- $ET(a_k, a_j, s)$ is the trust in expertise that an agent a_k has in a service s offered by an agent a_j which is defined as the aggregation of three quality of service criteria namely: specialization, reliability and quality rating (for details see Section 5.3);
- λInf_k and $\lambda\text{Sup}_k \in [0, 1]$ are respectively, the trust in cooperation lower and upper thresholds,
- $\beta_k \in [0, 1]$ is the trust in coalition threshold.
- $Blist_k$ is a blacklist containing a set of not cooperative agents that do not comply with its terms of use.

Let us denote $A_p \subseteq A$ the set of trustworthy provider agents, $A_i = \{a_k \,|\, a_k \in A_p$ and $\exists s \in S_k$ such as $s.f \equiv f_i\}$ the set of providers offering a service with the functionality f_i, and $A_Q = \bigcup_{i=1}^{n} A_i$ is the set of providers offering services for all required functionalities in Q. As several agents could offer services for required functionalities, *coalition formation* between them can address the composition problem as follows.

Definition 5.4.2. Let Q be a user query. A coalition $c = \{x_1, x_2, \ldots, x_n \,|\forall i \in [\![1, n]\!], \exists k \in [\![1, s]\!]$ such as $x_i = a_k$ and $a_k \in A_i\}$ is a set of agents that instantiate Q.

During the coalition formation process providers are organized in coalitions where each of them is able to provide one or many required services. A coalition that does not contain a set of agents required to instantiate all functionalities in the query is called *intermediate coalition*. An intermediate coalition, denoted c_z, is a partial instantiation of a query such as $c_z = \{x_1, x_2, \ldots, x_n \,|\, x_i \in \{a_k, f_i\}$ and $a_k \in A_i\}$. The content of an intermediate coalition evolves. The transition from one intermediate coalition c_z to another c_{z+1} is done through *proposal*.

Definition 5.4.3. A proposal $\phi = \{y_1, y_2, \ldots, y_n \,|\, y_i \in \{a_k, \emptyset\}$ and $a_k \in A_i\}$ represents either a membership request or a membership offer. In case of a membership request, $\phi = \{y_1, y_2, \ldots, y_n \,|\, \exists! y_i$ such as $y_i = a_k \in A_i$ and $\forall j \neq i, y_j = \emptyset\}$. In case of a membership offer, $\phi = \{y_1, y_2, \ldots, y_n \,|\, y_i = x_i$ if $x_i \equiv a_k, y_i = \emptyset$ otherwise$\}$.

Coalition formation process

Our CFP unfolds in three sequential steps: initial coalition generation, member selection, and best coalition choice. Each phase produces intermediate results that feed into the next, enabling a structured and trust-aware approach to dynamic service composition.

Step 1: Initial Coalition Generation. The goal of this step is to generate an initial set of coalitions $\mathcal{C}$ within the trust-relation social network (TRSN). Given the user query Q, the provider set A_p, and the TRSN. the requester agent a_r (acting as broker) initializes $\mathcal{C} = \emptyset$ and identifies first-layer providers (*i.e.*, those at distance 1 from a_r). Each identified provider a_k is assigned to a *new initial coalition* c_z (*e.g.*, as illustrated in Figure 5.5, $\mathcal{C} = \{c_1 = \{a_2\},\ c_2 = \{a_6\}\}$), and receives an INFORM message indicating its coalition. Agents are assumed initially cooperative, but retain full autonomy to accept, leave, or reject

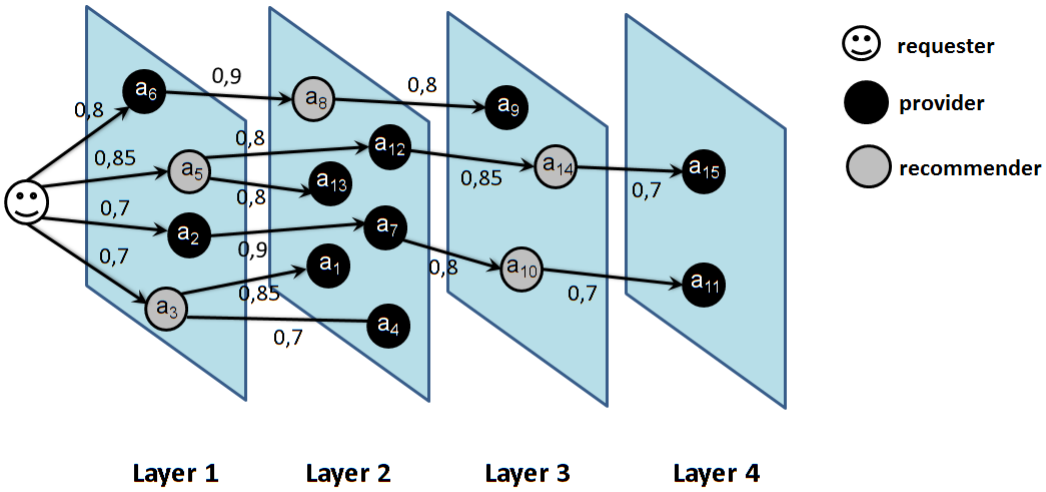


Figure 5.5: Multilayered representation of Trust-Relation Social Network

memberships. Upon receiving an INFORM message (see Step 2 below), an agent sets its role to *member*. Each newly formed coalition is then added to $\mathcal{C}$. By the end of this step, the broker has constructed the initial coalition set $\mathcal{C}$, used as input for the upcoming *member selection* step.

Step 2: Member Selection. The goal of this step is to complete each initial coalition $c_z \in \mathcal{C}$ by recruiting agents that provide services required by the user query. For clarity, we describe the process for a single coalition c_z. This member selection step is managed by the broker agent a_r, who incrementally expands c_z by adding one member per iteration.

The selection strategy gives priority to providers located closer to a_r in the TRSN, as they are considered more trustworthy. Given the dynamic nature of the coalition formation process, where agents may join or leave coalitions at runtime, a timeout mechanism $timer_r[z]$ is applied to prevent infinite loops.

At each layer TRSN(l), a_r identifies candidate providers $candP_r[z]$ offering the needed functionalities $func_r[z]$. If no suitable candidates exist, the search proceeds to layer TRSN($l+1$). Agent a_r selects the most trustworthy candidate a_j based on previously computed trust values during service discovery (see Section 5.3), and initiates a membership negotiation protocol represents in the Figure 5.6 using Agent Unified Modeling Language (Odell et al. [182]) formalism.

A PROPOSE message is sent to all current members of c_z to evaluate the candidate a_j. Each member responds based on two criteria: (i) whether a_j is in its blacklist $Blist_k$, and (ii)its *trust in cooperation* $CT(a_k, a_j)$, computed as follows:

Definition 5.4.4. Let $NbSoll_k[j]$ be the number of solicitations made by a_k to a_j, and $NbMem_k[j]$ the number of times a_j joined a coalition with a_k. Then,

$$CT(a_k, a_j) = \begin{cases} 1, & \text{if } NbSoll_k[j] = 0 \\ \frac{NbMem_k[j]}{NbSoll_k[j]}, & \text{otherwise} \end{cases} \tag{5.11}$$

If $CT(a_k, a_j)$ exceeds the threshold λSup_k, member a_k accepts a_j by sending an ACCEPT_PROPOSAL and increments $NbSoll_k[j]$; otherwise, it replies with a REJECT_PROPOSAL. The broker a_r collects all responses and applies a majority rule. If accepted, a_r sends a PROPOSE to a_j with a membership offer; if rejected, a_j is removed from $candP_r[z]$, and synchronization variables are reset. Upon receiving the offer, a_j checks for blacklisted members in c_z. If none are found, it evaluates the coalition's trust level $evalC(a_k, c_z)$ as follows:

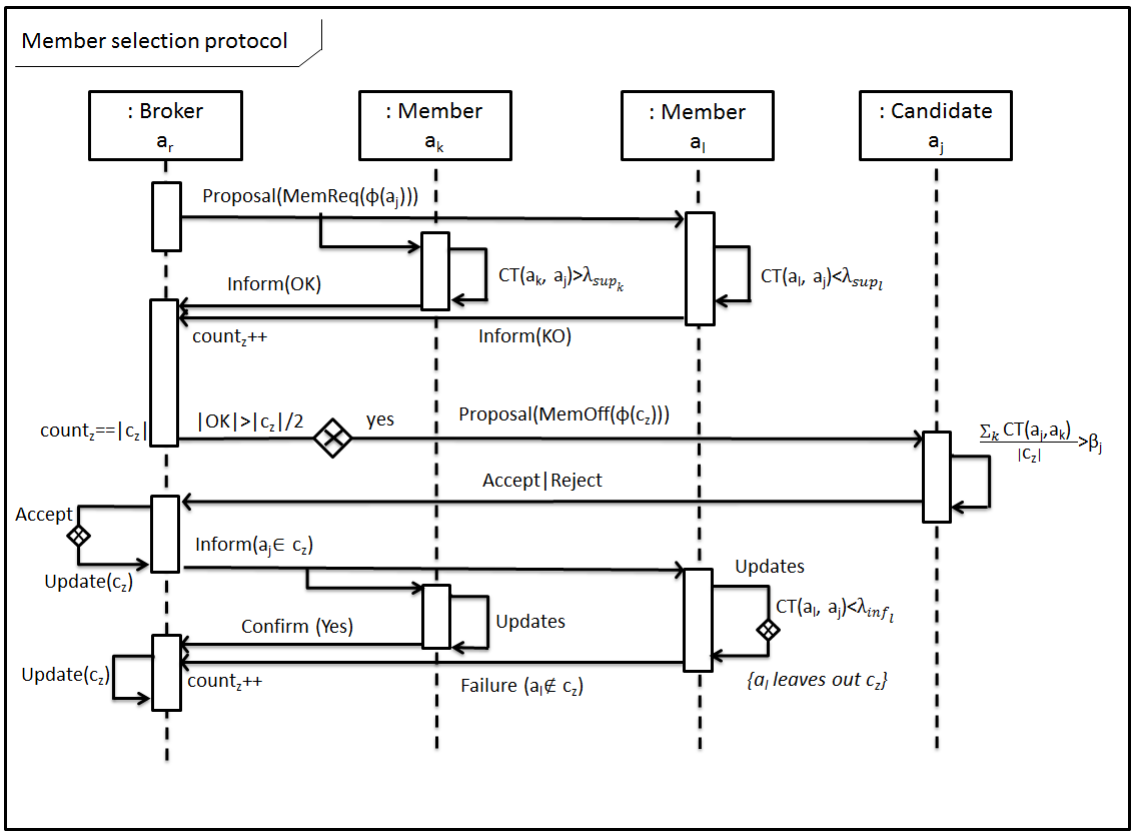


Figure 5.6: Member Selection Protocol

Definition 5.4.5. Let c_z be a coalition and a_k a candidate. Then,

$$evalC(a_k, c_z) = \frac{1}{|c_z|} \sum_{a_t \in c_z} CT(a_k, a_t) \tag{5.12}$$

If $evalC(a_k, c_z)$ exceeds the threshold β_k, a_k accepts the offer, joins c_z, and sends an ACCEPT to the broker. Otherwise, it sends a REJECT. In case of rejection, a_r removes a_k from $candP_r[z]$ and updates $counter_r[z]$. If accepted, a_r notifies coalition members, updates $func_r[z]$, and excludes satisfied functionalities from further search. If a member disagrees with the addition of a_k (*i.e.*, $CT(a_k, a_j) < \lambda \text{Inf}_k$), it may leave the coalition, blacklist a_k, and notify a_r via a FAILURE message. This dynamic blacklist prevents ping-pong effects and preserves coalition stability. On receiving CONFIRM or FAILURE, a_r updates $counter_r[z]$. If a member leaves, it is removed from c_z, its functionalities are restored to $func_r[z]$, and the candidate set is updated. The member selection step for c_z continues until one of the following termination conditions is met: (i) all required functionalities are covered, (ii) timeout $timer_r[z]$ is reached, or (iii) the maximum TRSN layer is explored. A refined coalition set $\mathcal{C}$ is then passed to the next step.

Step 3: Best Coalition Choice. Our approach aim to address the challenge of identifying multiple candidate coalitions and selecting the most suitable one for the requester. To achieve this, the broker begins by discarding incomplete coalitions, then ranks the remaining ones based on their *trust in expertise* values. This value is defined as the average of the expertise scores of all coalition members (see Chapter 2, Equation 2.5). Each member's expertise score is computed from the QoS values of the services it contributes (see Definition 5.4.1). The coalition with the highest trust in expertise is selected as the final composite service and returned to the user.

5.5 Summary

In this chapter, we presented the evolving landscape of service-oriented computing and how integrating social trust and multi-agent systems can enhance service discovery, selection, and composition. After reviewing related work, we introduced a trust-based, user-centric perspective that incorporates subjective perceptions, social context, and historical interactions to evaluate both services and their providers. The proposed multi-agent system framework models trust through three dimensions, namely social, expertise, and recommendation, leveraging social network analysis and referral systems. After, we proposed a broker-based coalition formation model for dynamic service composition that adapts to trust dynamics and agent satisfaction.

CHAPTER 6

Conclusion and Future work

This chapter gives a summary of my contributions, followed by a presentation of some perspectives towards modern large-scale service-oriented systems that face two critical challenges: managing the increasing complexity of service dependencies, and extracting meaningful insights from service-generated logs. My research project examines how supply chain principles can address the dependency problem in complex service architectures, and how process mining can transform service logs into actionable business intelligence.

6.1 Summary of contributions

This manuscript summarized my research work in service-oriented computing, focusing on the integration of optimization theory and distributed systems to develop solutions for service discovery, selection, composition, and failure recovery.

First, we tackled service selection and composition by maximizing quality-of-service while preserving transactional consistency (see Chapter 3). We proposed two main approaches. Heuristic-based methods that maintain global transactional correctness while optimizing local QoS at the component level. Exact optimization methods that integrate behavioral specifications with QoS and transactional constraints. We further enriched these contributions by integrating fairness considerations to ensure equitable service selection, and by formally characterizing the computational complexity of QoS-aware composition problem. Second, my research work addressed fault tolerance through failure recovery mechanisms to ensure robust execution in composite services (see Chapter 4). Two primary recovery strategies were introduced. The first one was a coloured Petri net based modeling approach that supports both forward recovery (retrying or substituting failed components) and backward recovery (compensation-based rollback to a consistent state). The second one, a relaxed transactional model based on fuzzy atomicity, allowing partial successes when acceptable to the user, and including checkpointing for intermediate states preservation, enabling selective rollback and allowing user-driven control over recovery outcomes based on utility expectations. Lastly, acknowledging that quality-of-service alone is insufficient to capture the nuances of trust, we incorporated social, experiential, and relational trust dimensions and proposed two approaches (see Chapter 5). We proposed a multi-dimensional trust model, for improving service discovery and selection, defining three dimensions: societal (community credibility), expertise (service competence and reliability), and recommendation (credibility of recommenders). Then, a trust-driven composition approach within a multi-agent system, in which agents maintain both trust scores (as defined previously) and trust in cooperation (reflecting the likelihood of successful collaboration). Composition is guided by dynamic coalition formation, optimizing for both trust thresholds and service quality.

Throughout our contributions, we have assumed that services expose stable functional and non-functional specifications through their interfaces, with these remaining consistent over time. We have also assumed a moderate number of services, resulting in dependency graphs of manageable complexity.

6.2 Towards Large-scale Service-oriented Systems

Modern software systems rely on services, microservices, and open source libraries to create rapidly deployable applications by leveraging reusable components, thereby reducing development time and promoting continuous integration and delivery (Lewis and Fowler [138]). Although this practice offers unprecedented scalability and modularity, it poses two significant challenges that have a direct impact on the maintainability of these systems.

The first challenge stems from the exponential *growth of dependencies between services* as systems evolve (Chen [60]). What starts as a manageable set of interconnected services quickly evolves into a complex dependency graph, prone to the "dependency hell" problem, where managing and updating service dependencies becomes increasingly difficult and error-prone (Dragoni et al. [72]).

The second challenge concerns the *vast amounts of log data* generated by distributed services. Each service interaction, transaction, and system event produces log entries that, when properly analyzed, can reveal critical insights about business processes, performance bottlenecks, and operational patterns. However, the volume and complexity of these logs make manual analysis impractical, requiring automated approaches to extract meaningful information.

My future research work, detailed below, will address these two challenges by proposing the application of supply chain principles to deal with the complexity of dependencies (Xia et al. [239]), as well as process mining as a solution for log analysis (Van der Aalst [224]).

Managing Complex and Evolving Service Dependency Graphs

Context. Modern software increasingly rely on the composition of services, microservices, and shared libraries creating large-scale systems where the number of components and their dependencies grows exponentially in both size and complexity (Newman [181]). Such systems may include hundreds or even thousands of components, each with its own set of dependencies, forming an intricate graph of dependencies that is hard to manage or visualize. This phenomenon, commonly referred to as "dependency hell", mirrors similar issues faced in software package management but at an architectural level (Abate et al. [1]). As systems scale, their dependency graphs become increasingly complex, posing significant challenges to their *quality maintenance*. Maintaining quality is both a goal and a challenge within this context. Relying on outdated components creates maintenance debt, as the dependency graph becomes harder to manage over time.

Moreover, while modularity and independent deployment are strengths of large-scale systems, they also lead to uncoordinated components evolution. Changes to APIs, contracts, or performance in one component may unintentionally affect many others. This *evolution* problem calls for mechanisms that support continuous compatibility checking, proactive detection of potential breakages, and automated updates. Managing evolving dependencies is thus essential for reducing cost, ensuring quality, and maintaining secure and up-to-date systems.

As part of Damien Jaime's doctoral thesis, that I have co-supervised with a collegue from Sorbonne Université, we have established a strong foundation for this research challenge through several contributions. In Jaime et al. [123], we proposed a tool-based solutions for analyzing dependencies of a project, capable of scaling up to complete ecosystems of dependencies (such as Java/Maven with its millions of libraries and hundreds of millions of dependency relationships). This work demonstrated the feasibility of large-scale dependency analysis and established baseline performance metrics for ecosystem-wide analysis. After that, in Jaime et al. [125], we developed a solution based on multi-objective, multi-criteria optimization for constructing dependency evolution plans. This solution integrates quality and security criteria while accounting for possible disruptive changes, providing an approach to dependency management that balances multiple competing objectives. The datasets and tools developed as part of this initial work were successfully utilized in a major data mining challenge, demonstrating their practical applicability and robustness at scale (Jaime et al. [124]).

Objective. Building on our previous contributions, our future research aims to address several issues in managing dependencies, particularly in large-scale systems:

- Vulnerability assessment: dependencies are part of the broader software supply chain and can be a significant entry point for cyberattacks if left unpatched. When vulnerabilities are disclosed through CVEs (Common Vulnerabilities and Exposures), their severity is rated, helping both maintainers and users prioritize updates. However, due to the complexity of dependency graphs, especially transitive dependencies, developers may be unaware of vulnerable dependencies within their codebase.

- Dependency updates and compatibility: updating dependencies can introduce breaking changes due to API incompatibilities. Even with semantic versioning, upgrades, particularly across major versions, remain risky when driven by security or performance requirements. In large-scale systems, version conflicts are common, especially in ecosystems like Java/Maven where only one version of a library can be used, making dependency resolution complex.

Methodology. Inspired by traditional supply chain management, we propose applying its principles to software dependency management, using the Software Bill of Materials (SBOM) as a central organizing concept (Mirakhorli et al. [173]). An SBOM offers a machine-readable inventory of all components and dependencies within a software system and is increasingly mandated by regulations such as the EU's Cyber Resilience Act (2024/2847). SBOMs help assess whether a system is theoretically affected by known vulnerabilities and support informed risk mitigation. However, current SBOM standards and tools such as SPDX and CycloneDX face limitations in automation, expressiveness, and actionable analysis (Stalnaker et al. [210]). Our future research aims to address these shortcomings as well as the above identified issues through the following directions:

- Formal SBOM modeling: existing SBOM formats lack formal semantics for automated reasoning. We propose to define a structured and formal SBOM model to support precise analysis and tool integration.

- Extended dependency analysis: we aim to expand SBOM content to cover broader build-related elements such as language versions, compilers, and builders. This extension should be a key for detecting obsolescence or supply chain risks.

- Vulnerability assessment: taking into account the actual impact of risks. As opposed to current approaches based on simple dependency path analysis (SBOM declares a direct or indirect dependency on a vulnerable element), it will be necessary to analyse the code from a semantic perspective (the context of use means that the vulnerability can actually be triggered) in order to rule out false negatives (O'Donoghue et al. [183]).

- SBOM updates: we propose developing algorithms that enable automated SBOM updates to address identified issues while minimizing disruption to system functionality.

- Automated code repair: based on SBOM diffs and patterns learned from dependency graphs across ecosystems, we aim to develop tools for recommending or performing safe code and configuration repairs automatically.

Managing complex and evolving dependency graphs is a key challenge in modern large-scale software systems. This research proposal presents an approach that combines formal modeling, algorithmic techniques, and supply chain principles to improve maintainability and automation in dependency management. By leveraging Software Bill of Materials (SBOMs) as a central tool, this research proposal aims to transform how dependencies are understood and managed, providing structured insight and automated support for updates and evolution. Looking ahead, future extensions of this research may include machine learning–based analysis for even more sophisticated dependency management.

This future research work will be part of Fanuel Mehari doctoral thesis (2025−) that I will co-supervised with a colleague from Sorbonne Université.

Object-Centric Process Mining for Service-oriented Systems

Context. Service-oriented business processes have become foundational to modern organizations, which rely on interconnected services to deliver business capabilities. These large-scale systems generate vast volumes of event logs, capturing service interactions, transactions, and system events. Such logs are rich in information about process execution, service behavior, performance metrics, and system interactions, making them valuable for understanding and optimizing organizational operations (Dumas et al. [74], Van der Aalst [224]). However, extracting actionable insights from this complex data requires advanced analytical techniques.

Traditional process mining methods have proven effective in discovering process flows, identifying bottlenecks, and detecting deviations from expected behavior (Van der Aalst [223]). Yet, they are fundamentally constrained by a single-case perspective, assuming each event belongs to exactly one process instance. This model falls short in service-oriented systems, where processes span multiple services, involve various entities, and exhibit complex many-to-many relationships between events and cases. Object-Centric Process Mining (OCPM) addresses these limitations by offering a more holistic view. Rather than relying on a single case notion, OCPM considers multiple object types and allows events to involve multiple objects (Van der Aalst[225]), Berti and Van der Aalst [27]). This multidimensional approach enables richer process analysis and is more aligned with the realities of service-oriented systems. Moreover, the complexity and granularity of object-centric data open new opportunities for advanced prediction and automation tasks that were previously infeasible in traditional process mining settings (Ghahfarokhi et al. [97]).

As part of Wissam Gherissi's doctoral thesis, we proposed several contributions that aimed at effectively exploiting object-centred information to improve predictive performance. In Gherissi et al. [99], we proposed a flattening-based approach that integrated object-centric information through explicit feature engineering. Combined with sequential deep learning models, this approach proved effective across various predictive tasks, including a dedicated method for remaining time prediction. In Gherissi et al. [98], we introduced an end-to-end deep learning model that integrates graph-based structural encoding using Graph Attention Networks (GAT) with sequential temporal modeling via LSTM. This hybrid approach enhances predictive accuracy by capturing both structural context and temporal dependencies within flattened object-centric event logs (OCELs). Lastly, in Gherissi et al. [100], we presented a modular and extensible framework for Object-Centric Predictive Process Monitoring (OCPPM). This framework unifies the development and evaluation of OCPPM approaches by organizing the prediction pipeline into configurable components, allowing flexible experimentation with preprocessing techniques, data representations, graph embeddings, and predictive models.

Objective. Object-centric predictive process monitoring approaches have mainly extended traditional prediction tasks, such as remaining time estimation and next activity forecasting, to multi-object scenarios (Galanti et al. [91]). However, the rich relational structures and temporal dynamics inherent in object-centric data open the door to new classes of predictions, offering deeper insights into process behavior and enabling more advanced management strategies. To further build on our previous contributions, we identify several key directions for advancing the field:

- Anomaly detection and prediction: traditional anomaly detection focuses on deviations within individual cases. In object-centric settings, however, anomalies often arise from interactions between objects, cascading failures, or implicit cross-object constraints that are invisible when analyzing objects in isolation (Yeshchenko et al. [247]). These include temporal synchronization anomalies (*i.e.*, unexpected timing relationships between linked objects), interaction pattern anomalies (*i.e.*, deviations in the nature or frequency of inter-object interactions), or cascade failure (*i.e.*, failures propagating across object types).

- Causal inference in object-centric processes: understanding causality is important for intervention and optimization. Unlike correlation-based methods, causal inference answers "what-if" questions. This is particularly complex in multi-object settings due to inter-object confounding (*i.e.*, multiple pathways of influence between objects), or causal relationships that evolve over time.

- Predictive maintenance and resource optimization: in object-centric environments, resources often serve multiple object types with diverse and evolving needs. Effective optimization must account for multi-object resource dependencies (*i.e.*, shared resources with varying priority levels across object types), cascading resource failures (*i.e.*, one resource's failure may differently affect multiple object types), and dynamic resource requirements (*i.e.*, object-specific needs that vary with process state or external factors).

Methodology. To address these three directions, we propose a develop a comprehensive framework that integrates advanced object-centric process mining techniques, analytical methods and automation. The aim is to support complex large-scale service-oriented systems in which multiple objects interact dynamically.

- Graph-based anomaly detection: we propose to develop an approach based on multi-layer Graph Neural Network (GNN) architecture for modeling both intra-object temporal patterns and inter-object relational dynamics. Object-centric data can be encoded as dynamic heterogeneous graphs, where nodes represent individual object instances (*e.g.*, orders, customers, products), and edges denote interactions or relationships among them. Both nodes and edges can be enriched with temporal attributes to capture event timings and durations.
- Causal inference in multi-object temporal graphs: understanding causality in object-centric environments is essential for reliable prediction and intervention. To this end, we propose leveraging deep learning methods to learn causal embeddings that represent object-object dependencies, enabling both causal discovery and prediction.
- Reinforcement learning for resource optimization: in dynamic service-oriented systems, resources are often shared among multiple object types. To support adaptive, efficient allocation and scheduling, we propose to use a reinforcement learning approach to forecast short and long-term resource demands, informed by the historical behavior of object types.

This future research work will be part of Silas Workman's doctoral thesis (2025−) that I will co-supervised with a colleague from Paris Dauphine Université.

Bibliography

Self-citations appear in color.

[1] P. Abate, R. Di Cosmo, J. Boender, and S. Zacchiroli. Strong dependencies between software components. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 89–99. IEEE, 2009. 76

[2] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *Proceedings of the 33rd annual Hawaii international conference on system sciences*, pages 9–pp. IEEE, 2000. 61

[3] F. N. Abu-Khzam, C. Bazgan, J. El Haddad, and F. Sikora. On the complexity of qos-aware service selection problem. In *International Conference on Service-Oriented Computing*, pages 345–352. Springer, 2015. 10, 11, 21, 26, 37

[4] P. S. Adler. Market, hierarchy, and trust: The knowledge economy and the future of capitalism. *Organization science*, 12(2):215–234, 2001. 61

[5] A. Agresti. *Categorical data analysis*, volume 792. John Wiley & Sons, 2012. 4

[6] J. Al-Sharawneh. *Social networks: Service selection and recommendation*. University of Technology Sydney (Australia), 2012. 59, 61

[7] J. Al-Sharawneh and M.-A. Williams. Abms: Agent-based modeling and simulation in web service selection. In *2009 International Conference on Management and Service Science*, pages 1–6. IEEE, 2009. 62

[8] A. Alhosban, Z. Malik, K. Hashmi, B. Medjahed, and H. Al-Ababneh. A two phases self-healing framework for service-oriented systems. *ACM Transactions on the Web (TWEB)*, 15(2):1–25, 2021. 43

[9] G. Alonso, F. Casati, H. Kuno, V. Machiraju, G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web services*. Springer, 2004. 5

[10] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the 18th international conference on World wide web*, pages 881–890, 2009. 5, 24

[11] M. Alrifai, T. Risse, and W. Nejdl. A hybrid approach for efficient web service composition with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 6(2):1–31, 2012. 23

[12] V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena. Refactoring product lines. In *Proceedings of the 5th international conference on Generative programming and component engineering*, pages 201–210, 2006. 36

[13] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, et al. Business process execution language for web services, 2003. 6

[14] R. Angarita, Y. Cardinale, and M. Rukoz. Faceta: Backward and forward recovery for execution of transactional composite ws. In *Extended Semantic Web Conference*, pages 343–357. Springer, 2012. 6, 43

[15] R. Angarita, M. Rukoz, and Y. Cardinale. Modeling dynamic recovery strategy for composite web services execution. *World Wide Web*, 19:89–109, 2016. 18

[16] D. Ardagna and B. Pernici. Global and local qos guarantee in web service selection. In *International Conference on Business Process Management*, pages 32–46. Springer, 2005. 23, 26, 37

[17] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on software engineering*, 33(6):369–384, 2007. 5, 23

[18] E. K. Asl, J. Bentahar, H. Otrok, and R. Mizouni. Efficient coalition formation for web services. In *2013 IEEE International Conference on Services Computing*, pages 737–744. IEEE, 2013. 62, 64

[19] P. K. Atrey, M. A. Hossain, and A. El Saddik. Association-based dynamic computation of reputation in web services. *International Journal of Web and Grid Services*, 4(2):169–188, 2008. 5

[20] I. Awan, M. Younas, and S. Benbernou. Convergence of cloud, internet of things, and big data: New platforms and applications, 2021. 24

[21] V. Azevedo, M. Mattoso, and P. Pires. Handling dissimilarities of autonomous and equivalent web services. *Proc. of Caise-WES*, 2003. 48

[22] S. K. Bansal, A. Bansal, and M. B. Blake. Trust-based dynamic web service composition using social network analysis. In *2010 IEEE International Workshop on: Business Applications of Social Network Analysis (BASNA)*, pages 1–8. IEEE, 2010. 59, 62, 63, 65

[23] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. Dovey, D. Feygin, A. Kochman, P. Macias, M. Novotny, M. Paolucci, et al. Universal description, discovery and integration specification (uddi) 3.0. *Online: http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm*, 2002. 1

[24] N. Ben Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny. Qos-aware service composition in dynamic service oriented environments. In *Middleware 2009: ACM/IFIP/USENIX, 10th International Middleware Conference, Urbana, IL, USA, November 30–December 4, 2009. Proceedings 10*, pages 123–142. Springer, 2009. 5, 24

[25] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, and F. Toumani. On automating web services discovery. *The VLDB Journal*, 14(1):84–96, 2005. 5

[26] E. Berscheid and H. T. Reis. Attraction and close relationships. In D. Gilbert, S. T. Fiske, and G. Lindzey, editors, *Handbook of Social Psychology*, pages 193–281. Oxford University Press, 4 edition, 1998. 61, 67

[27] A. Berti and W. M. Van der Aalst. Oc-pm: analyzing object-centric event logs and process models. *International Journal on Software Tools for Technology Transfer*, 25(1):1–17, 2023. 78

[28] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 141–150, 2009. 2

[29] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *Computer*, 32(7):38–45, 1999. 2

[30] S. Bhiri, O. Perrin, and C. Godart. Ensuring required failure atomicity of composite web services. In *Proceedings of the 14th international conference on World Wide Web*, pages 138–147, 2005. 15, 23, 24, 25

[31] S. Bhiri, O. Perrin, and C. Godart. Extending workflow patterns with transactional dependencies to define reliable composite web services. In *Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW'06)*, pages 145–145. IEEE, 2006. 23, 24

[32] H. Billhardt, R. Hermoso, S. Ossowski, and R. Centeno. Trust-based service provider selection in open environments. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 1375–1380, 2007. 59, 61, 62

[33] E. Billionniere, D. Greiman, and K. Gosha. A comparison of social service selection techniques. In *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 260–265. IEEE, 2009. 62

[34] E. Blanco, Y. Cardinale, M.-E. Vidal, J. El Haddad, M. Manouvrier, and M. Rukoz. A transactional-qos driven approach for web service composition. In *International Workshop on Resource Discovery*, pages 23–42. Springer, 2012. 10, 11, 21, 25, 29

[35] P. A. Bonatti and P. Festa. On optimal service selection. In *Proceedings of the 14th international conference on World Wide Web*, pages 530–538, 2005. 5, 7, 23, 26

[36] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture. *Online: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/*, 2004. 1

[37] A. Bouguettaya, M. Singh, M. Huhns, Q. Z. Sheng, H. Dong, Q. Yu, A. G. Neiat, S. Mistry, B. Benatallah, B. Medjahed, et al. A service computing manifesto: the next 10 years. *Communications of the ACM*, 60(4):64–72, 2017. 5

[38] J. Bourdon, L. Vercouter, and T. Ishida. A multiagent model for provider-centered trust in composite web services. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 216–228. Springer, 2009. 62, 64

[39] M. Brambilla, S. Ceri, S. Comai, and C. Tziviskou. Exception handling in workflow-driven web applications. In *Proceedings of the 14th international conference on World Wide Web*, pages 170–179, 2005. 42

[40] A. Brogi, S. Corfini, and R. Popescu. Semantics-based composition-oriented discovery of web services. *ACM Transactions on Internet Technology (TOIT)*, 8(4):1–39, 2008. 24, 31

[41] K. Bryson, M. Luck, M. Joy, and D. T. Jones. Applying agents to bioinformatics in geneweaver. In *International Workshop on Cooperative Information Agents*, pages 60–71. Springer, 2000. 64

[42] T. Burnaby. On a method for character weighting a similarity coefficient, employing the concept of information. *Journal of the International Association for Mathematical Geology*, 2(1):25–38, 1970. 66

[43] C. Canal, J. M. Murillo, P. Poizat, et al. Software adaptation. *Obj. Logiciel Base données Réseaux*, 12(1):9–31, 2006. 2

[44] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1069–1075, 2005. 23

[45] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 131–140, 2009. 5

[46] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti. Flow-based service selection forweb service composition supporting multiple qos classes. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 743–750. IEEE, 2007. 5, 23

[47] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz. Web service selection for transactional composition. In *International Conference on Computational Science (ICCS)*, pages 2683–2692, 2010. 10, 11, 21, 25, 29

[48] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz. Cpn-tws: a coloured petri-net approach for transactional-qos driven web service composition. *International Journal of Web and Grid Services*, 7(1):91–115, 2011. 10, 11, 21, 25, 26, 29, 44

[49] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz. Transactional-aware web service composition: a survey. In *Handbook of Research on Service-Oriented Systems and Non-Functional Properties: Future Directions*, pages 116–141. IGI Global, 2011. 24

[50] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz. Web service composition based on petri nets: Review and contribution. In *International Workshop on Resource Discovery, Revised Selected Papers, LNCS 8194*, pages 83–122. Springer, 2013. 10, 11, 39, 43, 44

[51] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz. Measuring fuzzy atomicity for composite service execution. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 62–71. IEEE, 2016. 10, 11, 39, 43, 51

[52] Y. Cardinale, J. El Haddad, M. Manouvrier, and M. Rukoz. Fuzzy acid properties for self-adaptive composite cloud services execution. *Concurrency and Computation: Practice and Experience*, 31(2):e4360, 2019. 10, 11, 39, 43, 51

[53] Y. Cardinale and M. Rukoz. Fault tolerant execution of transactional composite web services: An approach. In *Proc. Fifth Int, Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 158–164, 2011. 48

[54] Y. Cardinale and M. Rukoz. A framework for reliable execution of transactional composite web services. In *Proceedings of the International Conference on Management of Emergent Digital Ecosystems*, pages 129–136, 2011. 43

[55] Y. Cardinale, M. Rukoz, and R. Angarita. Modeling snapshot of composite ws execution by colored petri nets. In *Resource Discovery: 5th International Workshop, RED 2012, Co-located with the 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Greece, May 27, 2012, Revised Selected Papers 5*, pages 23–44. Springer, 2013. 42

[56] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Journal of web semantics*, 1(3):281–308, 2004. 5

[57] F. Casati and G. Cugola. Error handling in process support systems. In *Advances in Exception Handling Techniques*, pages 251–270. Springer, 2001. 42

[58] K. Chard, K. Bubendorfer, S. Caton, and O. F. Rana. Social cloud computing: A vision for socially motivated resource sharing. *IEEE Transactions on services computing*, 5(4):551–563, 2011. 59

[59] Y. Charif and N. Sabouret. Dynamic service composition enabled by introspective agent coordination. *Autonomous agents and multi-agent systems*, 26(1):54–85, 2013. 62

[60] L. Chen. Microservices: architecting for continuous delivery and devops. In *2018 IEEE International conference on software architecture (ICSA)*, pages 39–397. IEEE, 2018. 76

[61] M. Chen and J. P. Singh. Computing and using reputations for internet ratings. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 154–162, 2001. 67

[62] S. Chen and K. Nahrstedt. Distributed quality-of-service routing in ad hoc networks. *IEEE Journal on Selected areas in Communications*, 17(8):1488–1505, 1999. 4

[63] H. Cheng, M. Zhong, and J. Wang. Diversified keyword search based web service composition. *Journal of Systems and Software*, 163:110540, 2020. 5

[64] R. Chinnicii, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (wsdl) version 2.0. *Online: https://www.w3.org/TR/wsdl/*, 2007. 1

[65] I. Čilić, V. Jukanović, I. P. Žarko, P. Frangoudis, and S. Dustdar. Qedgeproxy: Qos-aware load balancing for iot services in the computing continuum. In *2024 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 67–73. IEEE, 2024. 24

[66] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected areas in Communications*, 13(6):1048–1056, 1995. 5

[67] I. da Silva and A. Zisman. A framework for trusted services. In *International Conference on Service-Oriented Computing*, pages 328–343. Springer, 2012. 59

[68] J. Day and R. Deters. Selecting the best web service. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pages 293–307, 2004. 62, 63

[69] J. De Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, E. Oren, et al. Web service modeling ontology (wsmo). *Interface*, 5(1):50, 2005. 2

[70] V. Dialani, S. Miles, L. Moreau, D. De Roure, and M. Luck. Transparent fault tolerance for web services based architectures. In *Euro-Par 2002 Parallel Processing: 8th International Euro-Par Conference Paderborn, Germany, August 27–30, 2002 Proceedings 8*, pages 889–898. Springer, 2002. 6

[71] P. Dolog, M. Schäfer, and W. Nejdl. Design and management of web service transactions with forward recovery. In *Advanced Web Services*, pages 3–27. Springer, 2013. 6

[72] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216, 2017. 76

[73] M. Drezza. The qos-aware web service composition problem: a new approach for the selection step, 2014. 37

[74] M. Dumas, L. M. Rosa, J. Mendling, and A. H. Reijers. *Fundamentals of business process management*. Springer, 2018. 78

[75] S. Dustdar, D. Schall, F. Skopik, L. Juszczyk, and H. Psaier. *Socially enhanced services computing: modern models and algorithms for distributed systems*. Springer Science & Business Media, 2011. 59

[76] S. Dustdar and W. Schreiner. A survey on web services composition. *International journal of web and grid services*, 1(1):1–30, 2005. 5

[77] M. Eid, A. Alamri, and A. El Saddik. A reference model for dynamic web service composition systems. *International Journal of Web and Grid Services*, 4(2), 2008. 5

[78] J. El Hadad, M. Manouvrier, and M. Rukoz. Tqos: Transactional and qos-aware selection algorithm for automatic web service composition. *IEEE Transactions on Services Computing*, 3(1):73–85, 2010. 10, 11, 19, 21, 25, 27

[79] J. El Haddad. Optimization techniques for qos-aware workflow realization in web services context. In *International Workshop on Resource Discovery, Revised Selected Papers, LNCS6799*, pages 134–149. Springer, 2012. 23

[80] J. El Haddad, M. Manouvrier, G. Ramirez, and M. Rukoz. Qos-driven selection of web services for transactional composition. In *2008 IEEE International Conference on Web Services (ICWS)*, pages 653–660, 2008. 10, 11, 21, 25, 27

[81] J. El Haddad, M. Manouvrier, and M. Rukoz. A hierarchical model for transactional web service composition in p2p networks. In *IEEE International Conference on Web Services (ICWS)*, pages 346–353, 2007. 39, 44, 57

[82] J. El Haddad and O. Spanjaard. Composition de services web et équité vis-à-vis des utilisateurs finaux. *ROADEF 2009*, page 251, 2009. 10, 11, 21, 25, 26, 32, 34

[83] R. ElGhondakly, S. M. Moussa, and N. Badr. Service-oriented model-based fault prediction and localization for service compositions testing using deep learning techniques. *Applied Soft Computing*, 143:110430, 2023. 43

[84] T. Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 1900. 1

[85] V. Ermolayev, N. Keberle, and S. Plaksin. Towards agent-based rational service composition–racing approach. In *International Conference on Web Services*, pages 167–182. Springer, 2003. 62, 64

[86] C.-L. Fang, D. Liang, F. Lin, and C.-C. Lin. Fault tolerant web services. *Journal of Systems Architecture*, 53(1):21–38, 2007. 5

[87] J. Farrell and H. Lausen. Semantic annotations for wsdl and xml schema. *W3C recommendation*, 28, 2007. 2

[88] G. Friedrich, M. G. Fugini, E. Mussi, B. Pernici, and G. Tagni. Exception handling for repair in service-based processes. *IEEE Transactions on Software Engineering*, 36(2):198–215, 2010. 42

[89] W. Gaaloul, S. Bhiri, and M. Rouached. Event-based design and runtime verification of composite service transactional behavior. *IEEE transactions on services computing*, 3(1):32–45, 2010. 15

[90] V. Gabrel, M. Manouvrier, and C. Murat. Web services composition: complexity and models. *Discrete Applied Mathematics*, 196:100–114, 2015. 23

[91] R. Galanti, M. De Leoni, N. Navarin, and A. Marazzi. Object-centric process predictive analytics. *Expert Systems with Applications*, 213:119173, 2023. 78

[92] N. Gamez, J. El Haddad, and L. Fuentes. Managing the variability in the transactional services selection. In *Proceedings of the 9th International Workshop on Variability Modelling of Software-intensive Systems*, pages 88–95, 2015. 10, 11, 21, 26, 34

[93] N. Gamez, J. El Haddad, and L. Fuentes. Spl-tqsss: a software product line approach for stateful service selection. In *2015 IEEE International Conference on Web Services*, pages 73–80. IEEE, 2015. 10, 11, 21, 26, 34

[94] L. Gao, S. D. Urban, and J. Ramachandran. A survey of transactional issues for web service composition and recovery. *International Journal of Web and Grid Services*, 7(4):331–356, 2011. 6, 42

[95] S. K. Gavvala, C. Jatoth, G. Gangadharan, and R. Buyya. Qos-aware cloud service composition using eagle strategy. *Future Generation Computer Systems*, 90:273–290, 2019. 7

[96] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682, 1987. 64

[97] A. F. Ghahfarokhi, G. Park, A. Berti, and W. M. van der Aalst. Ocel: a standard for object-centric event logs. In *European Conference on Advances in Databases and Information Systems*, pages 169–175. Springer, 2021. 78

[98] W. Gherissi, M. Acheli, J. El Haddad, and D. Grigori. Predictive process monitoring using object-centric graph embeddings. In *International Conference on Service-Oriented Computing*, pages 55–67. Springer, 2024. 78

[99] W. Gherissi, J. El Haddad, and D. Grigori. Object-centric predictive process monitoring. In *International Conference on Service-Oriented Computing*, pages 27–39. Springer, 2022. 78

[100] W. Gherissi, J. El Haddad, and D. Grigori. A framework for object-centric predictive process monitoring using graph-based process executions. In *IEEE International Conference on Web Services ICWS*, 2025. 78

[101] J. Golbeck. Generating predictive movie recommendations from trust in social networks. In *International Conference on Trust Management*, pages 93–104. Springer, 2006. 62, 63, 67

[102] V. Grassi and S. Patella. Reliability prediction for service-oriented computing environments. *IEEE Internet Computing*, 10(3):43–49, 2006. 5

[103] J. Gray. The transaction concept: Virtues and limitations. In *VLDB*, volume 81, pages 144–154, 1981. 22

[104] N. Griffiths and M. Luck. Coalition formation through motivation and trust. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 17–24, 2003. 62, 64

[105] D. Grigori, J. C. Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval. In *2006 IEEE International Conference on Web Services (ICWS'06)*, pages 145–152. IEEE, 2006. 5

[106] D. Grigori, J. C. Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval: Application to conversation protocols. *Information Systems*, 33(7-8):681–698, 2008. 2

[107] O. M. Group. Business process model and notation, 2014. 6

[108] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013. 5

[109] C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Transactions on software engineering*, 26(10):943–958, 2002. 42

[110] R. Hamadi and B. Benatallah. A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference-Volume 17*, pages 191–200, 2003. 6

[111] M. Hammoum, M. E. Khanouche, N. Khoulalene, and B. Benatallah. Uncertainty qos-aware services composition: a systematic literature review for services community. *Service Oriented Computing and Applications*, 18(2):121–143, 2024. 24

[112] C.-W. Hang and M. P. Singh. Trust-based recommendation based on graph similarity. In *Proceedings of the 13th international workshop on trust in agent societies (TRUST). Toronto, Canada*, volume 82, 2010. 63

[113] C.-W. Hang, Y. Wang, and M. P. Singh. Operators for propagating trust and their evaluation in social networks. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1025–1032, 2009. 62, 63

[114] Q. He, J. Yan, H. Jin, and Y. Yang. Quality-aware service selection for service-based systems based on iterative multi-attribute combinatorial auction. *IEEE Transactions on Software Engineering*, 40(2):192–215, 2014. 38

[115] Q. He, R. Zhou, X. Zhang, Y. Wang, D. Ye, F. Chen, J. C. Grundy, and Y. Yang. Keyword search for building service-based systems. *IEEE Transactions on Software Engineering*, 43(7):658–674, 2016. 5

[116] D. Hutchison, G. Coulson, A. Campbell, and G. S. Blair. Quality of service management in distributed systems. *Network and Distributed Systems Management*, 1(273-303):13, 1994. 5

[117] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Autonomous agents and multi-agent systems*, 13(2):119–154, 2006. 62

[118] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, and C.-H. Chen. Dynamic web service selection for reliable web service composition. *IEEE Transactions on services computing*, 1(2):104–116, 2008. 5

[119] A. Immonen and D. Pakkala. A survey of methods and approaches for reliable dynamic service compositions. *Service Oriented Computing and Applications*, 8(2):129–158, 2014. 42

[120] V. Issarny, F. Tartanoglu, A. Romanovsky, and N. Levy. Coordinated forward error recovery for composite web services. In *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.*, pages 167–176. IEEE, 2003. 6

[121] D. Izquierdo, M.-E. Vidal, and B. Bonet. An expressive and efficient solution to the service selection problem. In *The Semantic Web–ISWC 2010: 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I 9*, pages 386–401. Springer, 2010. 24

[122] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl. Qos aggregation for web service composition using workflow patterns. In *Proceedings. Eighth IEEE International Enterprisei Distributed Object Computing Conference, 2004. EDOC 2004.*, pages 149–159. IEEE, 2004. 4, 13, 22, 23

[123] D. Jaime, J. El Haddad, and P. Poizat. Goblin: A framework for enriching and querying the maven central dependency graph. In *Proceedings of the 21st International Conference on Mining Software Repositories*, pages 37–41, 2024. 76

[124] D. Jaime, J. El Haddad, and P. Poizat. Navigating and exploring software dependency graphs using goblin. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, pages 369–371. IEEE, 2025. 76

[125] D. Jaime, P. Poizat, J. El Haddad, and T. Degueule. Balancing the quality and cost of updating dependencies. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 1834–1845, 2024. 76

[126] C. Jatoth, G. Gangadharan, and R. Buyya. Computational intelligence based qos-aware web service composition: a systematic literature review. *IEEE Transactions on Services Computing*, 10(3):475–492, 2015. 7

[127] S. Kalepu, S. Krishnaswamy, and S. W. Loke. Reputation= f (user ranking, compliance, verity). In *Proceedings. IEEE International Conference on Web Services, 2004.*, pages 200–207. IEEE, 2004. 62, 63

[128] K. Khadir, N. Guermouche, A. Guittoum, and T. Monteil. A genetic algorithm-based approach for fluctuating qos aware selection of iot services. *IEEE Access*, 10:17946–17965, 2022. 24

[129] M. Klusch and K. Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In *Coordination of Internet agents: models, technologies, and applications*, pages 197–224. Springer, 2001. 69

[130] N. Kokash and V. D'Andrea. Evaluating quality of web services: A risk-driven approach. In *International Conference on Business Information Systems*, pages 180–194. Springer, 2007. 23

[131] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*, volume 14. Springer Science & Business Media, 1996. 34

[132] K. Kritikos, B. Pernici, P. Plebani, C. Cappiello, M. Comuzzi, S. Benbernou, I. Brandic, A. Kertész, M. Parkin, and M. Carro. A survey on service quality description. *ACM Computing Surveys (CSUR)*, 46(1):1–58, 2013. 3, 4

[133] N. B. Lakhal, T. Kobayashi, and H. Yokota. Fenecia: failure endurable nested-transaction based execution of composite web services with incorporated state analysis. *The VLDB Journal*, 18(1):1–56, 2009. 24, 43

[134] F. Lalanne, A. Cavalli, and S. Maag. Quality of experience as a selection criterion for web services. In *2012 Eighth International Conference on Signal Image Technology and Internet Based Systems*, pages 519–526. IEEE, 2012. 16, 61, 62, 66

[135] D. Le Tien, O. Villin, and C. Bac. Corba application tailored manager for quality of service support. In *Proceedings Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000)(Cat. No. PR00607)*, pages 52–59. IEEE, 2000. 5

[136] K. Lee, K. C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *International Conference on Software Reuse*, pages 62–77. Springer, 2002. 26

[137] A. L. Lemos, F. Daniel, and B. Benatallah. Web service composition: a survey of techniques and tools. *ACM Computing Surveys (CSUR)*, 48(3):1–41, 2015. 6

[138] J. Lewis and M. Fowler. Microservices: a definition of this new architectural term. *MartinFowler.com*, 25(14-26):12, 2014. 76

[139] L. Li, C. Liu, and J. Wang. Deriving transactional properties of compositeweb services. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 631–638. IEEE, 2007. 15, 23, 24

[140] L. Li and Y. Wang. The roadmap of trust and trust evaluation in web applications and web services. In *Advanced Web Services*, pages 75–99. Springer, 2013. 61

[141] L. Li, Y. Wang, and E.-P. Lim. Trust-oriented composite service selection and discovery. In *European Conference on a Service-Based Internet*, pages 50–67. Springer, 2009. 59, 61

[142] M. Li, Z. Hua, J. Zhao, Y. Zou, and B. Xie. Arima model-based web services trustworthiness evaluation and prediction. In *International Conference on Service-Oriented Computing*, pages 648–655. Springer, 2012. 59, 61, 62

[143] Q. A. Liang and S. Y. Su. And/or graph and search algorithm for discovering composite web services. *International Journal of Web Services Research (IJWSR)*, 2(4):48–67, 2005. 6

[144] A. Liu, L. Huang, and Q. Li. Qos-aware web services composition using transactional composition operator. In *International Conference on Web-Age Information Management*, pages 217–228. Springer, 2006. 22, 42

[145] A. Liu, L. Huang, Q. Li, and M. Xiao. Fault-tolerant orchestration of transactional web services. In *International Conference on Web Information Systems Engineering*, pages 90–101. Springer, 2006. 41, 42

[146] A. Liu, Q. Li, L. Huang, and M. Xiao. Facts: A framework for fault-tolerant composition of transactional web services. *IEEE Transactions on Services Computing*, 3(1):46–59, 2009. 24, 39, 42

[147] A. Liu, H. Liu, Q. Li, L.-S. Huang, and M.-J. Xiao. Constraints-aware scheduling for transactional services composition. *Journal of Computer Science and Technology*, 24(4):638–651, 2009. 24

[148] G. Liu and Y. Wang. Trust-oriented service provider selection in complex online social networks. In *Advanced Web Services*, pages 363–380. Springer, 2013. 61, 62

[149] G. Liu, Y. Wang, and M. Orgun. Trust transitivity in complex social networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 1222–1229, 2011. 62, 63

[150] H. Liu, W. Zhang, K. Ren, Z. Zhang, and C. Liu. A risk-driven selection approach for transactional web service composition. In *2009 Eighth International Conference on Grid and Cooperative Computing*, pages 391–397. IEEE, 2009. 24

[151] F. Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif. Quality characteristics for software architecture. *Journal of object Technology*, 2(2):133–150, 2003. 3

[152] A. Louati, J. El Haddad, and S. Pinson. Trust-based service discovery in multi-relation social networks. In *International Conference on Service-Oriented Computing*, pages 664–671. Springer, 2012. 11, 59, 62, 63, 64

[153] A. Louati, J. El Haddad, and S. Pinson. A distributed decision making and propagation approach for trust-based service discovery in social networks. In *Group Decision and Negotiation. A Process-Oriented View: Joint INFORMS-GDN and EWG-DSS International Conference, GDN 2014, Toulouse, France, June 10-13, 2014. Proceedings*, pages 262–269. Springer, 2014. 11, 59, 62, 63, 64

[154] A. Louati, J. El Haddad, and S. Pinson. A multilevel agent-based approach for trustworthy service selection in social networks. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 3, pages 214–221. IEEE, 2014. 11, 59, 62, 63, 64

[155] A. Louati, J. El Haddad, and S. Pinson. Towards agent-based and trust-oriented service discovery approach in social networks. In *TRUST@ AAMAS*, pages 78–89. Citeseer, 2014. 11, 59, 62, 63, 64

[156] A. Louati, J. El Haddad, and S. Pinson. A multi-agent approach for trust-based service discovery and selection in social networks. *Scalable Computing: Practice and Experience*, 16(4):381–402, 2015. 11, 59, 62, 63, 64

[157] A. Louati, J. El Haddad, and S. Pinson. Trust-based coalition formation for dynamic service composition in social networks. In *Web Information Systems Engineering–WISE 2015: 16th International Conference, Miami, FL, USA, November 1-3, 2015, Proceedings, Part I 16*, pages 570–585. Springer, 2015. 11, 20, 59, 64, 69

[158] N. Luhmann. *Trust and power*. John Wiley & Sons, 2018. 61

[159] A. D. Luise. Qos-aware cloud service selection, 2016. 38

[160] Z. Maamar, S. K. Mostefaoui, and H. Yahyaoui. Toward an agent-based and context-oriented approach for web services composition. *IEEE transactions on knowledge and data engineering*, 17(5):686–697, 2005. 62

[161] Z. Maamar, N. C. Narendra, D. Benslimane, and S. Subramanian. Policies for context-driven transactional web services. In *Advanced Information Systems Engineering: 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007. Proceedings 19*, pages 249–263. Springer, 2007. 15, 25, 42

[162] Z. Maamar, L. K. Wives, Y. Badr, S. Elnaffar, K. Boukadi, and N. Faci. Linkedws: A novel web services discovery model based on the metaphor of "social networks". *Simulation Modelling Practice and Theory*, 19(1):121–132, 2011. 67

[163] A. Maaradji, H. Hacid, J. Daigremont, and N. Crespi. Towards a social network based approach for services composition. In *2010 IEEE International Conference on Communications*, pages 1–5. IEEE, 2010. 16, 59, 61, 62, 63

[164] H. E. Mansour and T. Dillon. Dependability and rollback recovery for composite web services. *IEEE Transactions on Services Computing*, 4(4):328–339, 2010. 6

[165] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al. Owl-s: Semantic markup for web services. *W3C member submission*, 22(4), 2004. 2

[166] S. Marzouk, A. J. Maâlej, I. B. Rodriguez, and M. Jmaiel. Periodic checkpointing for strong mobility of orchestrated web services. In *2009 congress on services-i*, pages 203–210. IEEE, 2009. 6

[167] P. Massa and P. Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24, 2007. 62, 63

[168] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE intelligent systems*, 16(2):46–53, 2001. 5

[169] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A transaction model for multidatabase systems. In *1992 12th International Conference on Distributed Computing System*, pages 56–57. IEEE Computer Society, 1992. 15

[170] X. Mei, A. Jiang, F. Zheng, and S. Li. Reliable transactional web service composition using refinement method. In *2009 WASE International Conference on Information Engineering*, volume 1, pages 422–426. IEEE, 2009. 24

[171] D. A. Menasce. Qos issues in web services. *IEEE internet computing*, 6(6):72–75, 2002. 4

[172] D. A. Menasce. Composing web services: A qos view. *IEEE Internet computing*, 8(6):88–90, 2004. 22, 23

[173] M. Mirakhorli, D. Garcia, S. Dillon, K. Laporte, M. Morrison, H. Lu, V. Koscinski, and C. Enoch. A landscape study of open source and proprietary tools for software bill of materials (sbom). *arXiv preprint arXiv:2402.11151*, 2024. 77

[174] S. Mistry, A. Bouguettaya, H. Dong, and A. K. Qin. Metaheuristic optimization for long-term iaas service composition. *IEEE Transactions on Services Computing*, 11(1):131–143, 2016. 7

[175] M. Moghaddam and J. G. Davis. Service selection in web service composition: A comparative review of existing approaches. *Web services foundations*, pages 321–346, 2013. 23

[176] S. B. Mokhtar, N. Georgantas, and V. Issarny. Cocoa: Conversation-based service composition in pervasive computing environments with qos support. *Journal of Systems and Software*, 80(12):1941–1955, 2007. 2, 37

[177] F. Montagut, R. Molva, and S. T. Golega. Automating the composition of transactional web services. *International Journal of Web Services Research (IJWSR)*, 5(1):24–41, 2008. 15, 23, 24, 25

[178] H. Moulin. *Axioms of cooperative decision making*. Number 15 in Econometric Society Monographs. Cambridge university press, 1991. 34

[179] I. Muller, R. Kowalczyk, and P. Braun. Towards agent-based coalition formation for service composition. In *2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 73–80. IEEE, 2006. 62, 64

[180] B. Neville, M. Fasli, and J. Pitt. Utilising social recommendation for decision-making in distributed multi-agent systems. *Expert Systems with Applications*, 42(6):2884–2906, 2015. 62

[181] S. Newman. *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2021. 76

[182] J. Odell, H. V. D. Parunak, and B. Bauer. Extending uml for agents. In *Proceedings of the agent-oriented information systems workshop at the 17th national conference on artificial intelligence*, pages 3–17, 2000. 71

[183] E. O'Donoghue, B. Boles, C. Izurieta, and A. M. Reinhold. Impacts of software bill of materials (sbom) generation on vulnerability detection. In *Proceedings of the 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, pages 67–76, 2023. 77

[184] M. Oriol, J. Marco, and X. Franch. Quality models for web services: A systematic mapping. *Information and software technology*, 56(10):1167–1182, 2014. 3

[185] J. O'Sullivan, D. Edmond, and A. Ter Hofstede. What's in a service? *Distributed and Parallel Databases*, 12:117–133, 2002. 4

[186] M. Ouzzani and A. Bouguettaya. Efficient access to web services. *IEEE Internet Computing*, 8(2):34–44, 2004. 4

[187] H.-Y. Paik, A. L. Lemos, M. C. Barukh, B. Benatallah, and A. Natarajan. *Web service implementation and composition techniques*, volume 256. Springer, 2017. 6

[188] I. Paik, D. Maruyama, and M. N. Huhns. A framework for intelligent web services: Combined htn and csp approach. In *2006 IEEE International Conference on Web Services (ICWS'06)*, pages 959–962. IEEE, 2006. 62

[189] M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, pages 3–12. IEEE, 2003. 1

[190] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223–255, 2008. 1, 5

[191] D. Perlman and B. Fehr. *The development of intimate relationships.* Sage Publications, Inc, 1987. 61

[192] P. F. Pires, M. R. Benevides, and M. Mattoso. Building reliable web services compositions. In *Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*, pages 59–72. Springer, 2002. 42

[193] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *Proc. of the Eleventh International World Wide Web Conference, Honolulu, HI*, volume 45, 2002. 62

[194] A. Portilla, G. Vargas-Solar, C. Collet, J.-L. Zechinelli-Martini, and L. García-Bañuelos. Contract based behavior model for services coordination. In *International Conference on Web Information Systems and Technologies*, pages 109–123. Springer, 2007. 23

[195] A. Portilla, G. Vargas-Solar, C. Collet, J.-L. Zechinelli-Martini, and L. García-Bañuelos. Contract based behavior model for services coordination. In *Web Information Systems and Technologies: Third International Conference, WEBIST 2007, Barcelona, Spain, March 3-6, 2007, Revised Selected Papers 3*, pages 109–123. Springer, 2008. 42

[196] R. Ranjan, B. Benatallah, S. Dustdar, and M. P. Papazoglou. Cloud resource orchestration programming: overview, issues, and directions. *IEEE Internet Computing*, 19(5):46–56, 2015. 5

[197] M. Rukoz, Y. Cardinale, and R. Angarita. Faceta*: Checkpointing for transactional composite web service execution based on petri-nets. *Procedia Computeri Science*, 10:874–879, 2012. 42, 43

[198] J. Sabater and C. Sierra. Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, pages 194–195, 2001. 62

[199] J. Sabater-Mir and L. Vercouter. Trust and reputation in multiagent systems. *Multiagent systems*, page 381, 2013. 61

[200] M. Schäfer, P. Dolog, and W. Nejdl. An environment for flexible advanced compensations of web service transactions. *ACM Transactions on the Web (TWEB)*, 2(2):1–36, 2008. 42

[201] A. Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003. 7

[202] D. Schuller, A. Miede, J. Eckert, U. Lampe, A. Papageorgiou, and R. Steinmetz. Qos-based optimization of service compositions for complex workflows. In *Service-Oriented Computing: 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10, 2010. Proceedings 8*, pages 641–648. Springer, 2010. 23, 26

[203] D. Schuller, A. Polyvyanyy, L. García-Bañuelos, and S. Schulte. Optimization of complex qos-aware service compositions. In *Service-Oriented Computing: 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings 9*, pages 452–466. Springer, 2011. 23, 26

[204] Z. Shen and J. Su. Web service discovery based on behavior signatures. In *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, volume 1, pages 279–286. IEEE, 2005. 5

[205] F. Siala, S. Lajmi, and K. Ghédira. Multi-agent selection of multiple composite web services based on cbr method and driven by qos. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, pages 90–97, 2011. 62

[206] C. Sierra and J. Debenham. Information-based reputation. In *International Conference on Reputation: Theory and Technology*. CNR-National Research Council, 2009. 62

[207] B. Singh and S. Gautam. The impact of software development process on software quality: a review. In *2016 8th international conference on computational intelligence and communication networks (CICN)*, pages 666–672, 2016. 3

[208] R. Singh and K. Kumar. Software fault prediction in service-oriented based systems. In *2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*, volume 5, pages 1131–1136. IEEE, 2024. 43

[209] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. Htn planning for web service composition using shop2. *Journal of Web Semantics*, 1(4):377–396, 2004. 62

[210] T. Stalnaker, N. Wintersgill, O. Chaparro, M. Di Penta, D. M. German, and D. Poshyvanyk. Boms away! inside the minds of stakeholders: A comprehensive study of bills of materials for software systems. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, 2024. 77

[211] V. Stantchev and C. Schropfer. Service-level enforcement in web-services-based systems. *International Journal of Web and Grid Services*, 5(2):130–154, 2009. 5

[212] Z. Stojanovic, A. Dahanayake, et al. *Service-oriented software system engineering: challenges and practices*. Idea Group Publishing, 2005. 1

[213] A. Strunk. Qos-aware service composition: A survey. In *2010 Eighth IEEE European Conference on Web Services*, pages 67–74. IEEE, 2010. 23

[214] L. Sun, S. Wang, J. Li, Q. Sun, and F. Yang. Qos uncertainty filtering for fast and reliable web service selection. In *2014 IEEE International Conference on Web Services*, pages 550–557. IEEE, 2014. 38

[215] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web semantics*, 1(1):27–46, 2003. 5

[216] M. Szell, R. Lambiotte, and S. Thurner. Multirelational organization of large-scale social networks in an online world. *Proceedings of the National Academy of Sciences*, 107(31):13636–13641, 2010. 64

[217] C. Szyperski, D. Gruntz, and S. Murer. *Component software: beyond object-oriented programming*. Pearson Education, 2002. 5

[218] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy. Dependability in the web services architecture. In *Architecting dependable systems*, pages 90–109. Springer, 2007. 39

[219] H. Tong, J. Cao, S. Zhang, and M. Li. A distributed agent coalition algorithm for web service composition. In *2009 Congress on Services-I*, pages 62–69. IEEE, 2009. 62

[220] H. Tong, J. Cao, S. Zhang, and M. Li. A distributed algorithm for web service composition based on service agent model. *IEEE Transactions on Parallel and Distributed Systems*, 22(12):2008–2021, 2011. 62

[221] I. Trummer, B. Faltings, and W. Binder. Multi-objective quality-driven service selection—a fully polynomial time approximation scheme. *IEEE Transactions on Software Engineering*, 40(2):167–191, 2013. 23, 24, 26

[222] H.-L. Truong and S. Dustdar. Principles for engineering iot cloud systems. *IEEE Cloud Computing*, 2(2):68–76, 2015. 5

[223] W. M. Van der Aalst. Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 3(2):1–17, 2012. 78

[224] W. M. Van der Aalst. Data science in action. In *Process mining: Data science in action*, pages 3–23. Springer, 2016. 76, 78

[225] W. M. Van der Aalst. Object-centric process mining: unraveling the fabric of real processes. *Mathematics*, 11(12):2691, 2023. 78

[226] W. M. Van der Aalst, A. P. Barros, A. H. Ter Hofstede, and B. Kiepuszewski. Advanced workflow patterns. In *International Conference on Cooperative Information Systems*, pages 18–29. Springer, 2000. 13

[227] W. M. Van Der Aalst and A. H. Ter Hofstede. Yawl: yet another workflow language. *Information systems*, 30(4):245–275, 2005. 6, 13

[228] M. Vargas-Santiago, S. E. P. Hernández, L. A. Morales-Rosales, and H. H. Kacem. Survey on web services fault tolerance approaches based on checkpointing mechanisms. *J. Softw.*, 12(7):507–525, 2017. 42

[229] K. Vidyasankar and G. Vossen. A multilevel model for web service composition. In *Proceedings of IEEE International Conference on Web Services*, pages 462–469. IEEE, 2004. 24

[230] L.-H. Vu, M. Hauswirth, and K. Aberer. Qos-based service selection and ranking with trust and reputation management. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 466–483. Springer, 2005. 59, 61, 62

[231] L. Wang, R. Ranjan, J. Chen, and B. Benatallah. *Cloud computing: methodology, systems, and applications*. CRC press, 2017. 5

[232] S. Wang, Z. Zheng, Q. Sun, H. Zou, and F. Yang. Cloud model for service selection. In *2011 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pages 666–671. IEEE, 2011. 38

[233] Y. Wang, L. Li, and G. Liu. Social context-aware trust inference for trust enhancement in social network based recommendations on service providers. *World Wide Web*, 18(1):159–184, 2015. 62, 63, 64, 68

[234] Y. Wang and M. P. Singh. Trust representation and aggregation in a distributed agent system. In *AAAI*, volume 6, pages 1425–1430, 2006. 62

[235] Y. Wang and J. Vassileva. Toward trust and reputation based web service selection: A survey. *International Transactions on Systems Science and Applications*, 3(2):118–132, 2007. 59, 61

[236] P. Wegner. Concepts and paradigms of object-oriented programming. *ACM Sigplan Oops Messenger*, 1(1):7–87, 1990. 5

[237] B. Wu, C.-H. Chi, and S. Xu. Service selection model based on qos reference vector. In *2007 IEEE Congress on Services*, pages 270–277. IEEE, 2007. 22, 23

[238] J. Wu and F. Yang. Qos prediction for composite web services with transactions. In *Service-Oriented Computing ICSOC 2006: 4th International Conference, Chicago, IL, USA, December 4-7, 2006, Workshops Proceedings 4*, pages 86–94. Springer, 2007. 24

[239] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu. An empirical study on software bill of materials: Where we stand and the road ahead. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2630–2642. IEEE, 2023. 76

[240] J. Xu, K. Chen, and S. Reiff-Marganiec. Using markov decision process model with logic scoring of preference model to optimize htn web services composition. *International Journal of Web Services Research (IJWSR)*, 8(2):53–73, 2011. 62

[241] J. Xu, Z. Li, H. Chi, M. Wang, C. Guan, S. Reiff-Marganiec, and H. Shen. Optimized composite service transactions through execution results prediction. In *2016 IEEE International Conference on Web Services (ICWS)*, pages 690–693. IEEE, 2016. 6

[242] Z. Xu, P. Martin, W. Powley, and F. Zulkernine. Reputation-enhanced qos-based web services discovery. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 249–256. IEEE, 2007. 59, 61, 62

[243] Y. Yan, P. Poizat, and L. Zhao. Repairing service compositions in a changing world. In *Software Engineering Research, Management and Applications 2010*, pages 17–36. Springer, 2010. 6

[244] L. Yang, Y. Dai, B. Zhang, and Y. Gao. Dynamic selection of composite web services based on a genetic algorithm optimized new structured neural network. In *2005 International Conference on Cyberworlds (CW'05)*, pages 8–pp. IEEE, 2005. 23

[245] Z. Ye, A. Bouguettaya, and X. Zhou. Economic model-driven cloud service composition. *ACM Transactions on Internet Technology (TOIT)*, 14(2-3):1–19, 2014. 38

[246] Z. Ye, X. Zhou, and A. Bouguettaya. Genetic algorithm based qos-aware service compositions in cloud computing. In *International Conference on Database Systems for Advanced Applications*, pages 321–334. Springer, 2011. 38

[247] A. Yeshchenko, C. Di Ciccio, J. Mendling, and A. Polyvyanyy. Comprehensive process drift detection with visual analytics. In *International conference on conceptual modeling*, pages 119–135. Springer, 2019. 78

[248] B. Yu and M. P. Singh. A social mechanism of reputation management in electronic communities. In *International Workshop on Cooperative Information Agents*, pages 154–165. Springer, 2000. 62

[249] B. Yu and M. P. Singh. Searching social networks. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 65–72, 2003. 62, 63

[250] Q. Yu and A. Bouguettaya. *Foundations for efficient web service selection*. Springer Science & Business Media, 2009. 5

[251] T. Yu and K.-J. Lin. Service selection algorithms for composing complex services with multiple qos constraints. In *Service-Oriented Computing-ICSOC 2005: Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005. Proceedings 3*, pages 130–143. Springer, 2005. 23

[252] T. Yu and K.-J. Lin. Qcws: an implementation of qos-capable multimedia web services. *Multimedia Tools and Applications*, 30(2):165–187, 2006. 62

[253] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 1(1):6–32, 2007. 5

[254] M. Zeleny. *Multiple criteria decision making Kyoto 1975*, volume 123. Springer Science & Business Media, 2012. 28

[255] M. A. Zemni, S. Benbernou, and M. Carro. A soft constraint-based approach to qos-aware service selection. In *Service-Oriented Computing: 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10, 2010. Proceedings 8*, pages 596–602. Springer, 2010. 5

[256] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, pages 411–421, 2003. 6

[257] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Transactions on software engineering*, 30(5):311–327, 2004. 3, 5, 7, 21, 22, 23, 26, 33

[258] W. Zhang, C. K. Chang, T. Feng, and H.-y. Jiang. Qos-based dynamic web service composition with ant colony optimization. In *2010 IEEE 34th Annual Computer Software and Applications Conference*, pages 493–502. IEEE, 2010. 23

[259] W. Zhang, Y. Yang, S. Tang, and L. Fang. Qos-driven service selection optimization model and algorithms for composite web services. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, volume 2, pages 425–431. IEEE, 2007. 22, 24

[260] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou. Sinan: Ml-based and qos-aware resource management for cloud microservices. In *Proceedings of the 26th ACM international conference on architectural support for programming languages and operating systems*, pages 167–181, 2021. 24

[261] Y. Zhang, B. Zhang, and C. Zhang. Correlation-supported composite service reselection. In *2014 IEEE International Conference on Web Services*, pages 510–517. IEEE, 2014. 38

[262] H. Zhao, Z. Benomar, T. Pfandzelter, and N. Georgantas. Supporting multi-cloud in serverless computing. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, pages 285–290. IEEE, 2022. 24

[263] Z. Zhao, J. Wei, L. Lin, and X. Ding. A concurrency control mechanism for composite service supporting user-defined relaxed atomicity. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 275–278. IEEE, 2008. 15, 42

[264] X. Zheng and Y. Yan. An efficient syntactic web service composition algorithm based on the planning graph model. In *2008 IEEE International Conference on Web Services*, pages 691–699. IEEE, 2008. 6

[265] Z. Zheng and M. R. Lyu. A distributed replication strategy evaluation and selection framework for fault tolerant web services. In *2008 IEEE international conference on web services*, pages 145–152. IEEE, 2008. 5

[266] Z. Zheng and M. R. Lyu. *QoS management of web services*. Springer, 2013. 5

[267] Z. Zheng, H. Ma, M. R. Lyu, and I. King. Wsrec: A collaborative filtering based web service recommender system. In *2009 IEEE international conference on web services*, pages 437–444. IEEE, 2009. 5

[268] Z. Zheng, H. Ma, M. R. Lyu, and I. King. Qos-aware web service recommendation by collaborative filtering. *IEEE Transactions on services computing*, 4(2):140–152, 2010. 5

[269] Z. Zheng, X. Wu, Y. Zhang, M. R. Lyu, and J. Wang. Qos ranking prediction for cloud services. *IEEE transactions on parallel and distributed systems*, 24(6):1213–1222, 2012. 38

RÉSUMÉ

Les systèmes orientés services ont émergé pour offrir des architectures logicielles flexibles, réutilisables et modulaires. Ils sont constitués de services indépendants et faiblement couplés, communiquant via des protocoles normalisés et encapsulant des fonctions métier spécifiques. La composition de ces services permet de créer des processus complexes à partir de fonctionnalités existantes. Mes recherches portent sur l'intégration des propriétés non fonctionnelles dans la sélection, la composition et la réparation des services. Avec mes collaborateurs, nous avons étudié l'optimisation de la variabilité des services, la fiabilité des compositions, la mise en œuvre de stratégies de reprise après panne et la construction de la confiance entre fournisseurs de services. Nous avons développé des algorithmes heuristiques et exacts pour sélectionner les services selon des critères de Qualité de Service (QoS) tels que le temps de réponse et le coût, tout en équilibrant les contraintes des utilisateurs. Notre travail intègre également des propriétés transactionnelles afin de renforcer la robustesse et la fiabilité des services. Pour gérer les défaillances inévitables dans des environnements dynamiques, nous avons proposé des mécanismes de reprise — avant, arrière et par points de contrôle — modélisés à l'aide de réseaux de Petri et de méthodes fondées sur les graphes, garantissant une récupération cohérente des compositions de services. Enfin, nous avons exploré l'intégration de la confiance et des interactions sociales dans les services fournis par l'humain, afin d'enrichir les modèles de découverte, de sélection et de composition des services. Ce manuscrit présente les principales contributions de nos travaux, expose ma vision de l'évolution des systèmes orientés services et ouvre la voie à de futures travaux de recherche.

ABSTRACT

Service-oriented systems emerged to provide flexible, reusable, and modular software architectures. They comprise independent, loosely coupled services that communicate through standardized protocols and encapsulate specific business functions. By composing these services, complex workflows can be created from existing functionalities. My research focuses on integrating non-functional properties into service selection, composition, and recovery. Together with collaborators, I have investigated optimizing service variability, ensuring reliability, implementing fault recovery, and fostering trust among service providers. We developed heuristic and exact algorithms to select services based on their Quality-of-Service (QoS) attributes such as response time and cost, balancing user constraints and performance. Our work also integrates transactional properties to enhance reliability and robustness. To address inevitable failures in dynamic environments, we proposed recovery mechanisms — forward, backward, and checkpoint-based — modeled with Petri nets and graph-based methods to ensure consistent recovery. Additionally, we explored trust and sociability in human-provided services, integrating these factors into service discovery, selection, and composition. This manuscript summarizes our main contributions, outlines my vision for advancing service-oriented systems, and identifies promising directions for future research.