AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur : ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite de ce travail expose à des poursuites pénales.

Contact : portail-publi@ut-capitole.fr

LIENS

Code la Propriété Intellectuelle – Articles L. 122-4 et L. 335-1 à L. 335-10
Loi n°92-597 du 1er juillet 1992, publiée au *Journal Officiel* du 2 juillet 1992
http://www.cfcopies.com/V2/leg/leg-droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITE DE TOULOUSE ET DE L'UNIVERSITE DE SFAX

Délivré par:

Université Toulouse Capitole

et

Ecole Nationale d'Ingénieurs de Sfax

Écoles doctorales : **Mathématiques, Informatique et Télécommunications de Toulouse**
Sciences et Technologies de Sfax

Présentée et soutenue par

Ikbel GUIDARA

le 04 Juillet 2016

Efficient Time and QoS-Aware Selection for Service Composition

---

Discipline : **Informatique**

Spécialité : **Informatique**

Unités de recherche: **LAAS-CNRS (UPR8001 LAAS)**
ReDCAD-ENIS (LR13ES26)

Directeurs de thèse: Mme, Nawal GUERMOUCHE, Maitre de Conférences, INSA
Mr, Said TAZI, Maitre de Conférences HDR, UT1 Capitole
Mr, Mohamed JMAIEL, Professeur, Université de Sfax
Mr, Tarak CHAARI, Maitre Assistant, Université de Sfax

JURY

Rapporteurs Mr, Farouk TOUMANI, Professeur, Université Blaise Pascale
Mme, Ikram AMMOUS, Maitre de Conférences HDR, Université de Sfax

Membres Mr, Djamal BENSLIMANE, Professeur, Université Lyon 1
Mr, Gene COOPERMAN, Professeur, Université Northeastern
Mr, Khalil DRIRA, Directeur de Recherche, LAAS-CNRS

To my parents.

Abstract

Service-Oriented Computing (SOC) paradigm has emerged in last years to support cooperation between loosely coupled services to build complex applications. It involves the description, discovery, selection, and composition of services to support rapid development of complex applications. Usually, theses applications can be specified as abstract business processes and the goal is to select a service for implementing each abstract task. In addition to the functional requirements that must be accomplished, the QoS (Quality of Service) parameters are of paramount importance. Due to the large number of candidate services with same functionalities but offering different QoS values, the selection of the most suitable services for implementing abstract tasks while fulfilling QoS in a timely manner is not trivial. Moreover, in real-world applications, services can have different dependencies between them (i.e., structural and temporal). Considering these dependencies, the selection problem becomes more complex. Additionally, services usually operate in highly uncertain and dynamic environments, which can cause erroneous behaviors during the execution. In this context, it is crucial to tackle the selection problem while considering functional requirements associated with QoS and temporal constraints at design and run time.

In this thesis, we contribute towards addressing the aforementioned challenges. Specifically, the main contributions of this thesis are as follows: (1) We propose pre-processing techniques to allow a scalable service selection without affecting the optimality of the selected solution. (2) We develop an efficient QoS-aware service selection approach that allows selecting the suitable service composition while fulfilling QoS and temporal constraints. The proposed approach can handle complex service selection problems while considering the aforementioned dependencies between services. (3) We propose a heuristic service selection approach to select a close-to-optimal solution based on clustering and constraints decomposition techniques. (4) To deal with dynamic and uncertain environments, we propose a proactive service selection approach for enforcing service composition adaptation at run time. The aim is to take early re-selection actions in order to reduce the possibility of execution interruption and increase the likelihood of finding a feasible solution. This approach deals with QoS fluctuations and changes in execution environments during execution (e.g., the availability of a new better service). The different contributions of the proposed approach are implemented and their efficiency is demonstrated and validated analytically and empirically through experimental results.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Introduction

Service-Oriented Architecture (SOA) paradigm allows the integration of several service components to develop complex business applications [3, 4]. SOA relies on the Service-Oriented Computing (SOC) [5], which has emerged as a new computing paradigm that enables the composition of multiple loosely coupled components usually encapsulated as services. In such paradigm, providers expose their offerings as services, which can be automatically discovered, invoked and composed to implement complex applications. These applications are usually specified as abstract business processes, which can be implemented via composite services. To do so, the selection of services for abstract business tasks while fulfilling Quality of Service (QoS) user's requirements is essential. These requirements are usually considered as the global QoS constraints of the targeted composite service. With the growing number of candidate services for each business task, which offer the same functionality but differ in their QoS attributes (e.g., cost, response

time, availability), the selection of the best combination (composition) of services that satisfies business process constraints and end-to-end user's requirements is a challenging task [6].

QoS-based service selection aims to select the best composition of services to implement a specific business process in order to fulfill global user's requirements. Despite active research, most of the service selection approaches neglect a very important aspect: *time*. In fact, in addition to QoS constraints, several temporal constraints can be required in order to guarantee the successful execution of the business process. The main focus of this thesis is to deal with service selection problem while considering QoS and temporal constraints. The selection problem is processed at both design time and execution time when several changes can occur in the environment.

The rest of the chapter is organized as follows. In Section 1.2, we detail the main scope and challenges of our research work. A motivating scenario is described in Section 1.3 to better illustrate the purposes of this thesis. The contributions of the thesis are detailed in Section 1.4. Finally, a brief description of the structure of the thesis is presented in Section 1.5.

1.2 Service Selection: Research Scope and Challenges

In SOC paradigm, several service components can be integrated into composite services to execute abstract business processes whose execution requires the selection of a set of elementary services to invoke abstract business tasks. QoS-based service selection aims to select one service for each business task so that, the resulted service composition satisfies all the functional requirements associated with QoS constraints. Usually, for each abstract task, several services can be candidate. To differentiate between these services, the selection process should be guided by not only functional properties but also by non functional attributes (i.e., QoS attributes). Nevertheless, the selection of the best combination of services that satisfies all constraints is a challenging task especially in large scale problems where the number of services, tasks and constraints can be very large. The selection of the best service composition for business processes has been widely treated in the literature. Despite active research in the context of service selection, some challenges still remain unsettled so far. These challenges are discussed in the next sections.

1.2.1 Constraints and Dependencies between Services

QoS-based service selection usually aims to select the set of services such that *global end-to-end QoS constraints* are fulfilled. However, besides global QoS constraints, in real-world scenarios, several additional constraints have to be considered to cater for not only users' requirements and service provider offers but also constraints specified at the business process level (e.g., structural, QoS and temporal constraints). Current selection approaches consider only structural constraints and assume that temporal properties can be viewed as a kind of QoS criteria. Moreover, *temporal constraints and dependencies between services* are usually considered when modeling and verifying service compositions [7–9] and neglected during the selection of the best combination of services. These constraints and dependencies can be specified implicitly (e.g., imposed by business rules and laws) or explicitly by process designers to increase the market share and profitability and gain control over the execution time of the processes [10]. Given for example a partner of electronics manufacturing organization. He can require in his business process that the manufacturing of peripheral parts must finish no later than 20 time units after the start of the process and that his organization can receive orders only at business hours. Moreover, some *QoS constraints* can also be specified in the business process [11]. For example, the business designer may require that the cost of the manufacturing task must be less than or equal to 10 cost units. Considering QoS and temporal constraints when selecting the best service combination is a crucial task since the violation of one or more constraints may affect the successful execution of the business process. These constraints make the selection problem heavily constrained and thus, more complex to resolve since the selection of each service may influence or be influenced by the selection of other services.

To guarantee the successful execution of business processes, *structural constraints* have to be also fulfilled by the set of selected elementary services. Several existing approaches focus only on sequential flows between tasks. Business process models with other branching structures have to be transformed to sequential model using existing techniques [12]. However, merging several execution paths in one or multiple sequential flows may lead to lose some important dependencies between business tasks and thus, global constraints might be violated. Moreover, this transformation can be impossible to apply in complex processes where several dependencies may exist between abstract tasks mainly structural and temporal dependencies. Thus, handling different structure branching is a very important challenge that needs to be considered when selecting the best service composition.

1.2.2 Time-dependent QoS

Most of the service selection approaches assume that QoS values offered by service providers are static and overlooked a commonly important aspect in reality: QoS values could change over time and might depend on the time of the execution. Indeed, within different time periods, QoS attributes of candidate services can have different values [1, 13–15]. For instance, the response time of a service during daytime can be longer than night time due to access tendency. Another example is that the invocation of the service during business hours can be more expensive than invoking it outside these peak hours. Thus, assuming only static QoS values is very restrictive to effectively represent services and reflect the impact of time on the QoS attributes. Moreover, as candidate services can have different QoS with respect to time, each service can be considered as offering more than one instance. Hence, the number of service combinations that have to be compared becomes larger and thus, the selection process becomes more time consuming.

1.2.3 Scalability and Optimality

A main feature that has to be considered in service selection problems is scalability. In fact, since the selection process aims to select the best service composition, all combinations of services have to be compared. Comparing all possible combinations is not practical and may lead to scalability issues mainly in large service selection problems where the number of possible combinations is huge and several constraints should be considered. This issue is more critical in dynamic selection when a solution has to be found in a reasonable time in order to guarantee end-to-end constraints during execution. A second important issue is the optimality of the selected solution. In fact, given user requirements, service selection aims to select the optimal solution among a large set of possible feasible solutions. However, achieving optimality may lead to scalability issues since all possible combinations should be compared. Hence, there is a trade-off between scalability and optimality.

1.2.4 Uncertainty and Dynamic Environments

Although static service selection at design time allows selecting a satisfactory solution, it does not guarantee that the selected services offer the estimated values during execution. In fact, service-oriented systems often operate in highly uncertain and dynamic environments. Due to uncertainties, current QoS delivered by services may not be the one foreseen and may deviate from the original specification due to several factors such

as the high overhead and network delay. For example, due to a high overhead, the response time of a selected service can be higher than estimated. Such deviations can affect the successful execution of the service combination during execution and thus, the satisfaction of end-to-end constraints. In addition to changes that can be observed on the selected services, due to the dynamic nature of the execution environments, several changes can appear on the system when executing services. Indeed, services can join or leave the system or change their characteristics at any time. These changes may also have several impacts on the selected solution. For instance, if a selected service is no more available, it should be substituted. Another example is when a new service that offers better values than the selected service, it should be considered to proactively enhance the selected solution in order to face, for instance, service deviations. To ensure the successful execution of the different business tasks while guaranteeing the satisfaction of end-to-end global user's constraints, service selection process needs to continuously react to environmental conditions variations during execution.

Despite active research to overcome these issues, most of existing approaches do not deal with temporal properties. These latter make the aforementioned issues very challenging because of the large number of constraints that should be considered comparing to existing approaches.

1.3 Motivating Scenario

To better illustrate the application of our approach, in this section, we introduce the electronic device production in a manufacture enterprise scenario. The corresponding business process is depicted in Figure 1.1.

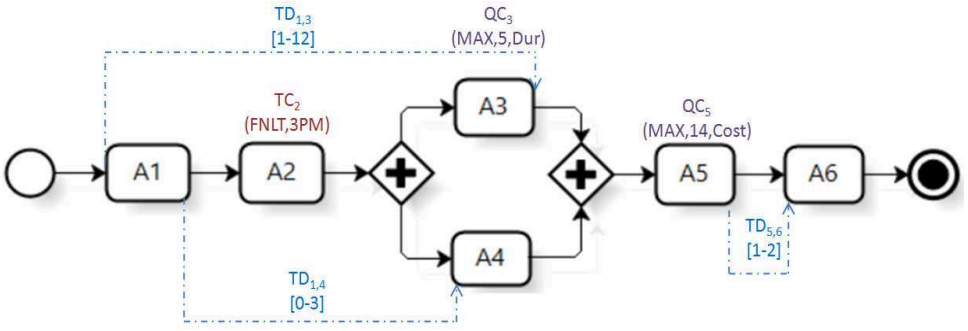

FIGURE 1.1: The business process for the production of an electronic device

The production process has six abstract tasks. This process starts by receiving and writing technical reports (A_1). Then, the adequate model of the device is designed (A_2). After that, the manufacturing of the electronic and peripheral parts is executed in parallel (A_3

TABLE 1.1: Description of the business activities.

Activity	Description
A_1	Receive the order and write the technical reports
A_2	Design the model of the device
A_3	Manufacture the electronic parts of the device
A_4	Manufacture the peripheral parts of the device
A_5	Assemble all parts
A_6	Test the final product

and A_4). After manufacturing the various parts of the product, these parts are assembled together (A_5). Finally, the product has to be tested before delivering (A_6).

In addition to *structural constraints* between the different tasks of the process, *QoS and temporal constraints* can be specified by business designers and market laws. For example, business designers can assign *QoS constraints* to one or more business tasks, we call *intra-task QoS constraints* (QC). In our scenario, we consider the following intra-task QoS constraints:

- QC_3: The duration of the manufacturing of the electronic parts (i.e., the task A_3) must not exceed 5 time units.
- QC_5: The cost of the parts assembly task (i.e., A_5) must not exceed 14 units.

Moreover, given that some tasks can be released internally in the enterprise (i.e. expressed in its internal business view), this latter may have some temporal constraints, related to the availabilities of their internal departments and laws. In our example, we consider the following *intra-task temporal constraint* (TC) related to the task A_2:

- TC_2: The business designer requires that the design of the product model (i.e., A_2) has to finish no later than 3 p.m. (i.e., 15 units of time in our example) since it should be checked by the designer who is available every day from 8 a.m. to 4 p.m.

Additionally, in order to increase the market share and profitability and to gain control over the production time, some temporal constraints between tasks can be imposed called *temporal dependencies* (TD). In the example presented in Figure 1.1, three temporal dependencies are specified:

- $TD_{1,3}$: The manufacturing of electronic parts (i.e., A_3) has to finish no earlier than one time unit and no later than 12 time units after the receipt of the order (i.e., A_1).

- $TD_{1,4}$: The manufacturing of peripheral parts (i.e., A_4) starts no later than 3 time units after the writing of the technical reports (i.e., A_1).
- $TD_{5,6}$: The test of the final product (i.e., A_6) has to start no earlier than one time unit and no later than 2 time units after the assembly of all parts (i.e., A_5) considering the time required for the transportation of the product after the assembly of its parts.

In addition, *global constraints* can be associated with the business process. In the example we consider, three global constraints are required:

- The global cost must not exceed 68 units.
- Once the process started, the global execution duration must not exceed 13 units of time.
- The deadline (i.e., the finish time) of the process must not exceed 10 p.m. (i.e., 22 in our example).

To be implemented, each abstract activity has a set of functionally similar concrete candidate services. Some of these services offer different QoS values according to their temporal properties (Figure 1.2). For example, when the service S_{13} is invoked from 9 to 13 units of time, it offers a duration of 3 time units and a cost of 23 cost units. However, from 14 to 20 units of time, it offers an execution duration equal to 4 time units with a cost equal to 19.

	Time Span	Duration	Cost
S_{11}	[8, 15]	4	12
S_{12}	[17, 21]	2	8
S_{13}	[9, 13] [14, 20]	3 4	23 19

	Time Span	Duration	Cost
S_{21}	[12, 17]	1	10
S_{22}	[7, 13]	4	12
S_{23}	[9, 12] [14, 21]	2 1	17 15

	Time Span	Duration	Cost
S_{31}	[13, 22]	4	12
S_{32}	[8, 14]	6	12
S_{33}	[10, 17] [17, 23]	4 5	13 23

	Time Span	Duration	Cost
S_{41}	[6, 13] [13, 23]	7 9	22 18
S_{42}	[17, 24]	6	15
S_{43}	[12, 20]	4	12

	Time Span	Duration	Cost
S_{51}	[7, 12] [15, 20]	1 1	8 14
S_{52}	[8, 16]	3	20
S_{53}	[14, 22]	2	12

	Time Span	Duration	Cost
S_{61}	[19, 22]	1	7
S_{62}	[9, 13] [14, 18]	6 4	9 10
S_{63}	[8, 20]	1	6

FIGURE 1.2: Candidate services of each abstract task

Let us now search the best service combination to implement the business process without taking into account temporal constraints. First, we suppose that all QoS values are static and do not depend on the time of the execution. Thus, the best combination of

services that satisfies the user requirements is $C = (S_{12}, S_{21}, S_{31}, S_{43}, S_{51}, S_{63})$. This combination can be selected based on the notion of dominance [16, 17]. In other words, each selected service chosen to implement an abstract activity offers the best QoS (i.e., it dominates all the other candidate services).

As stated previously, QoS parameters may change according to time. For example, the cost of the service S_{13} is equal to 23 form 9 to 13 units of time and to 19 from 14 to 20 units of time. When considering time-dependent QoS, the combination C is no more valid even if the selected services are the best. Indeed, although the task A_2 should be executed after the task A_1, the service S_{21} is available in a time span before that of the service S_{12} and thus, these two services can not be parts of the same solution. The combination $C^{'} = (S_{11}, S_{21}, S_{31}, S_{43}, S_{53}, S_{63})$ where the availability intervals are respectively, [8,12], [12,13], [13,17], [13,17], [17,19] and [19,20] is a satisfactory solution.

Again, the combination $C^{'}$ can not be a satisfactory solution if we deal with further constraints expressed in the business process. Consider, for instance, the temporal dependency $TD_{5,6}$ that states that the test of the final product has to start no earlier than one time unit and no later than 2 time units before the assembly of all parts. This constraint can not be satisfied by the combination $C^{'}$. In fact, since the service S_{53} ends its execution at 19, according to the temporal dependency $TD_{5,6}$, the service S_{63} can not start before 20, which is impossible. Thus, another service combination should be selected such as $C^{''} = (S_{11}, S_{21}, S_{31}, S_{43}, S_{53}, S_{61})$ with the following availability intervals [8,12], [12,13], [13,17], [13,17], [17,19] and [20,21], respectively.

To summarize, considering time-dependent QoS attributes associated with QoS and temporal constraints is not a trivial task and makes the selection problem very complex. The existing selection approaches can not be applied since most of them consider only static QoS values and do not deal with temporal constraints. Moreover, they usually suppose that the business process has a sequential structure. To overcome these limitations, we propose a time-aware selection approach while dealing with heavily constrained selection problems and considering several QoS and temporal constraints. In the next section, we present the different contributions of our work.

1.4 Research Aims and Contributions

In this thesis, we focus on the problem of service selection to implement an abstract business process. Particularly, the problem we are interested in can be described as follows: given an abstract business process in which complex structural, QoS and temporal constraints are specified, and a set of candidate services that offer time-dependent

QoS, our goal is to select the adequate services to build the best service combination that implements the abstract business process while fulfilling user's requirements. The aim is to propose a novel service selection approach to handle the challenges presented previously and deal with the limitations of existing approaches. The contributions of the thesis can be summarized as follows:

- First, we propose a constraint model of the selection problem. It allows to capture global user constraints as well as constraints at business and service levels. Particularly, it enables the specification of structural, QoS and temporal constraints while considering time-dependent QoS values. Unlike existing approaches, the proposed model allows handling complex composition structures including sequential, parallel, choice and loop patterns.
- Second, in order to deal with scalability issues, we propose a *service pruning process* based on dominance and constraint-based pruning techniques. Dominance pruning allows identifying the set of non-dominated services for each class whereas, constraints pruning is based on a set of computed *local thresholds* to narrow the search space and eliminate non adequate services and thus, reduce the number of service combinations to be considered prior to performing the selection algorithm. Furthermore, in case of failure (i.e., no solution is possible), the pruning step allows for identifying the cause of the failure at earlier stages (i.e., before performing the selection process). Based on this, we propose strategies to improve the selection problem. These contributions have been published in [18] and [19].
- Third, based on the results of the pruning phase, we propose a service selection algorithm that takes into consideration both time-dependent QoS attributes and business level constraints and ensures the selection of the best service composition while handling complex business process. This work has been extended later to deal with large scale problems where an optimal solution requires a high computation time to be found. Hence, we propose a heuristic service selection approach to select a close-to-optimal solution more efficiently. The proposed approach is based on clustering and constraints decomposition techniques. These contributions have been published in [20] and [21].
- Finally, to deal with uncertainties and environment changes, a novel dynamic service selection approach is presented [22] and [23]. The goal is to try to prevent at run-time the violation of the specified constraints. To do so, we propose a proactive selection approach whose aim is to enhance the selected service composition so that violations can be avoided. Moreover, reactions to changes and violations are made as soon as they occur in order to avoid execution interruption and increase the likelihood of finding a satisfactory solution.

1.5 Structure of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we review related literature on service composition selection and adaptation. In Chapter 3, a constraint-based service selection model is presented. In Chapter 4, we detail our pruning approach to eliminate inadequate services based on dominance and constraints pruning strategies. The pruning approach is followed by a description of an enhancement process to enable improving the selection problem when there is no a feasible solution. In Chapter 5, we present our service selection approaches. First, the optimal service selection algorithm is introduced to select the optimal solution when dealing with small selection problems. Second, to enhance the computation time when the selection problem is very large, we propose a heuristic approach to select a close-to-optimal solution while guaranteeing that the selected solution fulfills all constraints. A proactive service selection approach is outlined in Chapter 6 to deal with environment changes and uncertainties during the execution time. In Chapter 7, we evaluate our approach through experimental results. Finally, Chapter 8 concludes the thesis and gives some future directions.

Chapter 2

Background and Related Work

Contents

2.1 Introduction

After presenting the main research goals of this thesis in Chapter 1, in this chapter, we present a review of the state of the art related to service selection. We pay particular attention to the works that handle QoS and temporal constraints. In Section 2.2, we provide a brief overview of the service oriented computing paradigm. We discuss QoS based service selection models and techniques in Sections 2.3 and 2.4, respectively. We propose a detailed review of static service selection approaches at design time in Section 2.5. This is followed by a review of dynamic service selection approaches at run-time in Section 2.6. In Section 2.7, we discuss the limitations of existing approaches with respect to the research challenges presented in Chapter 1. Finally, we conclude the chapter in Section 2.8.

2.2 Service Oriented Computing

Service-Oriented Computing (SOC) paradigm enables the sharing of functionalities and resources via the integration of loosely-coupled components, exposed as *services* in order to build complex applications [5]. According to the definition proposed in [24], "*Services are self-describing, platform-agnostic computational elements*". The functionalities offered by services can vary from a simple request to complex business processes. In SOC paradigm, interacting components are usually implemented in different languages, developed by several independent providers and dispersed through different platforms.

2.2.1 Service Oriented Architecture

To build the service model, SOC paradigm relies on the service oriented architecture (SOA). SOA enables the integration of applications and resources by (1) modeling each application or resource as a service that has a specified interface, (2) allowing services to exchange information (e.g., messages, business objects, documents) and (3) allowing flexible coordination between services, so that they can be discovered, selected and invoked to be used by users or by other services independently on the platforms or the programming languages [25].

The basic SOA adopted in SOC is depicted in Figure 2.1. In this architecture, services follow a specified protocol: *publication*, *discovery* and *invocation*. Mainly, three actors are involved: *service providers*, *service consumers* and *service registries*. Service providers generate descriptions of their services using accepted standard formats and publish them in a service registry. Service consumers can then, discover and invoke the

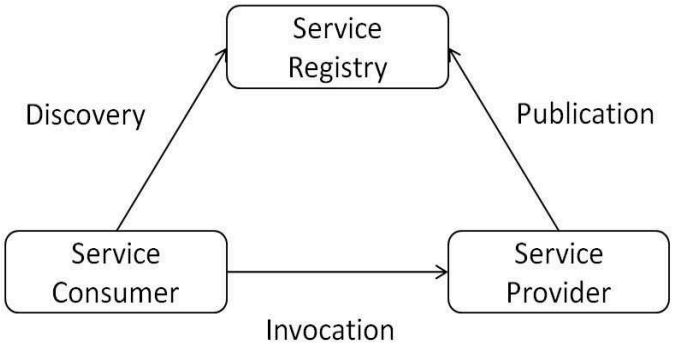

FIGURE 2.1: Roles and interactions involved in SOA

available services in the registries. For example, in the context of *web services*, the most adopted technology in SOC, services are described by an XML-based standard format (e.g., Web Service Description Language (WSDL) [26] and Web Ontology Language (OWL-S) [27]). Service descriptions can be accessible to the clients by advertising them in a Universal Description, Discovery and Integration (UDDI) service registry [28]. The interactions between the service providers and consumers to invoke services are achieved through Simple Object Access Protocol (SOAP) messages [29].

The main advantage of the above service technologies is to enable the automated composition of services which is discussed in the next section.

2.2.2 Service Composition Methods

Very often, user's requirements can not be fulfilled by an elementary service and require the aggregA fundamental step to satisfy complex user's requirements and enable the aggregation of multiple functionalities of services is the service composition. Due to the huge amount of available services, the composition of several elementary services becomes a challenging task. Service composition aims to determine the way of combining a set of services from a functional point of view in order to meet given requirements. Several methods have been proposed to compose services. These methods can be categorized into tow maim categories [30]: *AI-planning-based* and *workflow-based* approaches.

2.2.2.1 AI Planning based Composition

Artificial Intelligence (AI) planning composition methods assume that the service composition can be viewed as a plan and that a composition of services can be generated automatically to achieve a feasible plan [31–33]. These methods suppose that each service can be specified by a set of inputs and outputs. Based on these latter and on the user's requirements, the planner searches for a sequence of services that consumes and

produces the considered inputs and outputs, respectively, while satisfying the user's requirements. Although these methods allow to automatically generate the process model, they support simple sequential composition and do not take into account structural patterns between services (sequence, parallel, choice, loop).

2.2.2.2 Workflow based Composition

The workflow-based approaches [34, 35] can be classified into static and dynamic approaches. *Static workflow generation* methods assume that an abstract process model is given prior to the composition of services. The process model is composed of a set of abstract tasks and dependencies between them using multiple composition patterns [36, 37]. Based on the given abstract process, concrete services are then selected to implement each abstract task based on services availability and QoS requirements. *Dynamic workflow generation* methods create the process model and select services automatically based on user's requirements. Several specification languages have been proposed to specify the process model such as the graph-based languages [38] and Petri-net-based languages [39, 40].

2.3 Service Selection Models

Service selection problem can be formulated as a multi-objective combinatorial optimization problem (MOCOP) [41]. Combinatorial problems involves finding an assignment of a finite set of objects that satisfies a set of conditions. The solutions of a combinatorial problem are combinations of components that satisfy all conditions. Combinatorial optimization problems relate to an objective function that has to be optimized (minimized or maximized) while searching for the optimal combination. When several objectives must be optimized, we get the multi-objective combinatorial optimization problem [42]. This problem allows finding the optimal solution that balances multiple (often conflicting) objectives simultaneously. Hence, the service selection problem can be formulated as an MOCOP by mapping the values of candidate services for the different QoS parameters to the values of the objects in the MOCOP [1, 43]. Moreover, global QoS constraints can be mapped to the set of conditions of the optimization problem.

The most adopted methods to solve an MOCOP problem are the Simple Additive Weighting (SAW) method [44] and the pareto ordering [45–50]. The SAW method involves aggregating the values of all parameters into an overall utility value while giving a weight for each parameter. In this case, the problem is considered as a single-objective

optimization problem. Pareto ordering consists of finding the solution that is not dominated by any other solution (i.e., the solution that has the best values for all parameters) [51–54]. If there is no single solution that dominates all other combinations, a set of several pareto solutions is identified.

Several sub-problems of the MOCOP have been proposed in the literature to formally specify the service selection problem and facilitate its resolution [41]. The mot adopted ones are the Multi-dimension Multi-choice Knapsack Problem (MMKP) and Multi-constraint Optimal Path Problem (MCOP) which are defined in the following.

2.3.1 Multi-dimension Multi-choice Knapsack Problem (MMKP)

The knapsack problem can be presented as follows. Given a set of items with each item having an associated profit and a number of required resources, the aim is to select a sub-set of items to put into a knapsack while maximizing the overall profit and respecting the maximum allowed capacity of the knapsack. If the items are combined into several classes and only one item must be selected from each class, we obtain the multi-choice knapsack problem. Moreover, if several constraints must be fulfilled (e.g., the maximum allowed weight and volume of the knapsack), we get the multi-dimension multi-choice knapsack problem. Given this definition, QoS-aware service selection problem can be formulated as an MMKP [35] by mapping the candidate services of each abstract task to the set of items in each class and the combination of services with global QoS constraints to the knapsack with limited capacities.

2.3.2 Multi-constraint Optimal Path Problem (MCOP)

The multi-constrained optimal path (MCOP) problem involves finding the optimal path between a couple of nodes in a graph such that constraints imposed on the attributes of the different nodes are satisfied [55]. In such a problem, each node in the graph has a profit value and a set of attributes. The selected path is the path with the highest profit amongst all possible feasible paths between the two nodes such that all constraints are fulfilled. Thus, by mapping candidate services and structural dependencies between them to the set of nodes and edges between them, service selection problem can be modeled as an MCOP problem that selects the best combinations of services (one service for each node) with the highest overall utility while satisfying all global constraints [35].

2.4 Service Selection Techniques

Several techniques are proposed to solve the service selection problem. These techniques can be divided into four main categories: *mathematical programming*, *constraint programming* and *meta-heuristic and heuristics*.

2.4.1 Mathematical Programming

Mathematical programming (MP) is the dominant standard technique used to find solutions to optimization problems. It involves the use of mathematical models to assist making optimal decisions. The most known MP technique is the *Linear Programming (LP)*. LP consists on optimizing an objective function subject to a set of linear constraints over a set of real variables. *Integer Programming (IP)* is an extension of LP where all variables are constrained to be integer. If only some variables are required to be integer, then the problem is called a *Mixed Integer Programming (MIP)* problem. One important feature of MIP is the use of constraint relaxation techniques.

2.4.2 Constraint Programming

Recently, *Constraint Programming (CP)* has emerged to extend MP methods while offering more flexibility on modeling optimization problems such as the possibility of modeling logical constraints [56, 57]. In CP paradigm, relations between variables are expressed in the form of constraints that can be non-linear. Indeed, CP formulates the problem as a constraints satisfaction model composed of a set of variables that have values ranging over specific finite domains and are linked by a set of constraints that have to be satisfied. CP deals with *Constraint Satisfaction Problems (CSP)* in which a set of constraints should be satisfied by the selected solution. *Constraint Optimization Problems (COP)* are considered as a generalization of CSPs to deal with optimization problems in which an objective function should be optimized. The key element of CP is the use of constraint propagation, which makes it very suitable and efficient for solving several optimization problems such as scheduling and sequencing problems.

2.4.3 Meta-heuristics and Heuristics

Meta-heuristics are used in several combinatorial optimization problems and can be applied to a broad range of applications [58]. They are problem-independent and thus, they can be used as black boxes. They aim to search for good but not necessarily optimal solutions in order to solve a hard optimization problem in an acceptable amount of

time. Most of meta-heuristic algorithms are based on evolutionary algorithms [59] (e.g., *Genetic Algorithms (GA)*) or on nature-inspired algorithms (e.g., *ant colonies*, *artificial immune systems*). *Heuristics*, on the other hand, are problem-dependent and they are usually adapted to a specific problem to try to find accurate solutions in a reasonable time while considering the particularities of this problem.

Recently, some works proposed to combine two or more techniques to solve the selection problem. Next, we give a review on static service selection approaches.

2.5 Static Service Selection

QoS-aware service selection requires the selection of a combination of services that satisfies all constraints while optimizing several QoS values at the same time. In this section, we provide a review of service selection strategies and approaches to resolve the selection problem.

2.5.1 Static Service Selection Strategies

Based on the service selection models presented above, QoS-aware service selection approaches proceed to the selection of services using two broad strategies: *local optimization* and *global optimization* strategies.

2.5.1.1 Local Service Selection

Local optimization strategy aims to select the best service from each class of candidate services (i.e., candidate services of each task) independently from other classes. This strategy is especially used in distributed environments where a global QoS management is not required [60–65]. Even though the local optimization strategy is very efficient in terms of computation time, it is not suitable for QoS-based service selection since it does not guarantee that the global QoS constraints are satisfied.

2.5.1.2 Global Service Selection

Global optimization strategy relies on a bottom-up method, which is adopted at the composite service level. This approach consists in searching candidate services that optimize the QoS of the composite service and fulfill all global QoS constraints while considering several combinations of services [35, 66–71]. Although global optimization approaches

guarantee that the optimal solution will be found, they have a poor performance especially in terms of computation time. Consequently, these approaches can be applied only when the number of candidate services is very limited.

2.5.1.3 Hybrid Service Selection

To deal with the limitations of these two strategies, some researchers proposed *hybrid selection strategy* adopting a top-down method. These approaches assume that global constraints can be considered as the aggregation of a set of local ones. Several techniques have been proposed to decompose global constraints to local ones [16, 21, 72–74]. Based on these latter, local optimization algorithms are applied to select the best service for each abstract task such that all local constraints are fulfilled. The aim of these approaches is to reduce the complexity of the service selection problem while increasing the performance of the proposed algorithm.

2.5.2 Static Service Selection Approaches

The selection of the best service combination to implement abstract business tasks has been widely treated in the literature [11, 16, 17, 35, 43, 66, 67, 75]. To solve the service selection problem, various approaches have been proposed to explore and select service compositions. To enable the selection of the concrete service combinations, we identify two main categories of service selection approaches: *exact* and *Approximate* approaches. For each category, we distinguish between approaches that apply pre-processing techniques to reduce the search space before the selection process and those that do not apply any pre-processing techniques.

2.5.2.1 Exact Service Selection

Exact selection approaches usually adopt exhaustive methods to evaluate all possible combinations of services in order to select the optimal solution (i.e., the combination of services that offers the best quality values according to the specified requirements) [66, 67, 76].

- *Exact methods without pre-processing:*

A popular way to solve the service selection problem is the use of constraint programming methods. Constraint programming is widely adopted in the literature to solve

combinatorial optimization problems in different applications. In [76], Ben Hassine et al. discuss a web service composition approach using constraint programming. The authors formalize the service selection problem based on a constraint optimization problem, which is a generalization of the constraint satisfaction problem where the goal is to find a solution that maximizes an objective function. A similar approach is proposed in [77] to model the service selection problem as a COP while handling both hard and soft QoS constraints to specify respectively, constraints that have to be satisfied and optional constraints. Another way of solving the service selection problem is to model it as a mixed integer programming problem. In [66], Zeng et al. use MIP techniques to select the optimal combination of services and achieve global optimization of QoS attributes. This work is extended in Ardagna et al. [67] to include local QoS constraints and loop peeling to deal with composition structures with cycles. These techniques are usually applied to tackle global service selection in order to select the optimal solution. These approaches use the SAW method to evaluate the quality of the service composition. In contrast, Klopper et al. [14] identify pareto-optimal service compositions (i.e., all non-dominated service compositions) while taking into consideration time-dependent QoS values. The authors formulate the service selection problem as a multi-objective timed composition problem. In this work, authors assume that all QoS attributes are monotonically decreasing. All the previous approaches are suitable for small problems, as the complexity of the selection algorithm increases exponentially when the problem size increases. This causes several scalability issues especially when dealing with a large number of candidate services for each task.

- *Exact methods with pre-processing:*

Reducing the search space prior to the selection process allows to enhance the performance of service selection algorithms. Some methods reduce the number of candidate services based only on functional properties [78] and thus, they cannot be applied in QoS-aware service selection problems. Barakat et al. [17] apply two space reduction techniques to reduce the number of candidate services and the number of alternative abstract plans. The authors use the notion of dominance to select representative services for each task without affecting the optimal solution. The selection problem is modeled as an MCOP problem. To solve the selection problem, the authors adopt an algorithm that extends the Bellman-Ford algorithm [79] (called multi-constrained Bellman-Ford algorithm). In [80] Huang et al. propose pruning techniques to reduce the number of services to be considered. First, they remove services that cannot be executed (i.e., they have no inputs). Then, they eliminate services that cannot offer optimal QoS values. However, these approaches do not deal with the time dimension.

In [52], a compositional decision making process is introduced to select the pareto-optimal composition solutions. Pareto-optimal services are first selected at the service level to discard all uninteresting service instances for each business task. Then, the pareto-optimal approach is used at the composed level to select all non-dominated combinations of services. In this work, authors propose two dominance-based pruning strategies, which rely on both QoS objectives and constraints, where objectives are the requirements that have to be optimized and constraints present the requirements that have to be satisfied. A similar approach is proposed in [54] to select the set of the optimal combinations of services based on constraints and objective dominance. These approaches can be time consuming when the number of services is very high due to the significant computational overhead caused by the pairwise comparisons.

2.5.2.2 Approximate Service Selection

To overcome the limitations of exact solutions, several work proposed *approximate selection approaches*. These approaches aim to reduce the computation time and deal with scalability issues. They are used to find a near-to-optimal solution more efficiently than exact solutions.

- *Approximate methods without pre-processing:*

Several works proposed approximate algorithms to reduce the computation time and deal with scalability issues. These works propose heuristics that can be used to find a near-to-optimal solution more efficiently than exact solutions. Some heuristic approaches are based on global search. Yu et al. [35] introduce two alternative models for the QoS-based service composition problem: the combinatorial model to define the selection problem as a multi-dimension multi-choice knapsack problem (MMKP) and the graph model to define the selection problem as a multi-constraint optimal path (MCOP) problem. Based on these models, authors propose heuristic algorithms to achieve efficient selection process. Other approaches are based on evolutionary algorithms to select a near-to-optimal solution. Such algorithms usually browse the combinations of services in a random way to iteratively find near-to-optimal solutions. In [11, 81–83], authors model and resolve the service selection problem based on genetic algorithms (GA). Canfora et al. [11] encode the selection problem as a genome and adopt a fitness function to evaluate the quality of each service combination. Since GAs are unconstrained procedures, the authors propose a distance-based penalty approach to penalize individuals that do not meet the global constraints and integrate constraint-handling during the selection process. This is achieved by incorporating constraints in the fitness function

in order to drive the evolution towards constraint satisfaction. A similar approach is presented in [81] to enable service composition in cloud computing. In these approaches, QoS values are normalized using the SAW method. Other approaches adopt other meta-heuristic algorithms such as ant colony [84], immune [85, 86] and bees based methods [87]. These approaches, however, are restrictive in heavily constrained problems especially in the presence of several temporal constraints and dependencies between services. Moreover, these algorithms can run endlessly (if a stop criterion is not defined) without any guarantee on the quality of the obtained solution.

Some approaches cater for the time dimension when composing and selecting services. In [10], Liang et al. propose a penalty-based genetic algorithm to select services under temporal constraints. In this work, authors assume that QoS values are static and do not depend on the time of the execution and only upper bound temporal constraints between activities are considered. Zhang et al. [88] model the selection problem as a multi-domain scheduling problem that minimizes service resources under time constraints. The proposed service selection algorithm allows identifying the start and the finish time of each selected service while handling complex business structures. Wagner et al. [1] propose a service selection approach with time-dependent QoS attributes. Apart from that, inter-service dependencies are considered to describe general dependencies between QoS values of interacting services (e.g., the cost of a *compress movie* service may depend on the amount of data generated by the service *retrieve file*). The authors employ a GA to solve the multi-objective optimization problem. The proposed algorithm selects a set of feasible solutions while specifying the start and the finish time of each selected service instance according to the values of QoS attributes at each time period. Both approaches [88] and [1] do not consider temporal constraints at the business level.

- *Approximate methods with pre-processing:*

Recently, some approaches propose techniques to decompose global QoS constraints into local ones. The idea consists in defining allowed local constraints for each business task so that the global constraints are satisfied. Thus, for each business task, the candidate services that do not satisfy the local constraints can be removed. Local optimization is then applied to select the best service for each abstract task such that all local constraints are fulfilled. For instance, Alrifai et al. [16] use a mixed integer programming model to compute local constraints of each task based on a set of QoS levels for each QoS attribute. After that, a local selection strategy is applied to select the best service for each task. As a step forward, Qi et al. [72] suggest a local optimization method to further reduce the number of candidate services based on QoS levels. In [73], Sun et al. introduce a QoS decomposition approach based on the mean and the standard

deviation of each QoS attribute while considering several composition structures. Another approach is proposed in [74] to define local constraints using genetic algorithm. Although the proposed solutions scale better when dealing with large problems, they rely on greedy pruning methods when computing local constraints that can affect the ability to find an optimal solution. Additionally, these works are not able to handle time-dependent QoS attributes associated with temporal constraints and dependencies between services.

The technology of the skyline has been also considered as an important concept in QoS-based service selection for eliminating candidate services. Skyline analysis was first treated in mathematical problems [89] and then it was investigated into the database domains [90–92]. Recently, skyline computation has received significant consideration in service selection problems. The skyline is a set of services that are not dominated by any other service based on their QoS values. A service s_1 is said to dominate another service s_2 if and only if it is better than or equal to s_2 in all QoS parameters and it is strictly better than s_2 in at least one parameter. Yu et al. [51] identify service skylines from uncertain QoS information to reduce the search space. The authors introduce a pruning strategy to remove the p-dominated providers (i.e., the chance that these providers are dominated by other providers is greater than or equal to a given probability threshold p). In [53], Alrifai et al. propose hierarchical clustering method to cluster the skyline services according to their utilities in order to prune all non-skyline services. The authors build a tree structure of representative skyline services that will be used as inputs to a MIP model. First, a small subset of services in inserted into the MIP. The search space of the representative tree is then expanded iteratively by including more representative services until a solution to the MIP is found. Benouaret et al. [93] introduce two skyline variants, the σ-dominant skyline and the α-dominant skyline to deal with both size and quality requirements. The first requirement allows reducing the size of skyline when it is very large, whereas, the second requirement aims to enhance the quality of the skyline by including interesting services and removing services with bad compromise between their QoS parameters. This work is only suitable to select skyline services for an individual abstract task and does not handle the selection of a service composition. Moreover, none of the above approaches identifies skyline services while handling time-dependent QoS.

2.5.3 A Synthesis of Static Service Selection Approaches

In this Section, we present a synthesis of existing selection approaches considering the following criteria (See Table 2.1):

TABLE 2.1: A synthesis of static service selection approaches.

	Exact/ Approximate	**Global/ Hybrid**	**Pre-processing Technique**	**Selection Technique**	**Constraints**	**Time-dependent QoS**	**Service Dependencies**
[66]	Exact	Global	x	IP	Global QoS	x	Structural (C)
[67]	Exact	Global	x	MIP	Local and Global QoS	x	Structural (C)
[76]	Exact	Global	x	COP	Global QoS	x	Structural (C)
[17]	Exact	Global	Task and Plan based Pruning	Bellman-Ford algorithm	Global QoS	x	Structural (S)
[14]	Exact	Global	x	Pareto-optimal solutions	Global QoS	√	Structural (S) Temporal (FS)
[52]	Exact	Global	Dominance based Pruning	Pareto-optimal solutions	Global QoS	x	Structural (C)
[11]	Approximate	Global	x	GA	Global QoS	x	Structural (C)
[1]	Approximate	Global	x	GA	Global QoS	√	Structural (C) QoS Temporal (FS)
[35]	Approximate	Global	x	Heuristic algorithm	Global QoS	x	Structural (C)
[10]	Approximate	Global	x	GA	Global QoS	x	Structural (C) Temporal (FS)
[53]	Approximate	Global	Dominance based Pruning	MIP	Global QoS	x	Structural (S)
[16]	Approximate	Hybrid	Local Constraint based Pruning	QoS decomposition (MIP) Local selection	Global QoS	x	Structural (S)
[72]	Approximate	Hybrid	Selection of promising services	MIP	Global QoS	x	Structural (S)
[73]	Approximate	Hybrid	Local Constraint based Pruning	QoS decomposition (Mean and standard deviation) Local selection	Global QoS	x	Structural (C)
[74]	Approximate	Hybrid	Local Constraint based Pruning	QoS decomposition (GA) Local selection	Global QoS	x	Structural (S)

- *Exact/Approximate*: to indicate if the selection approach investigates an exact or an approximate selection approach
- *Local/Global*: to indicate if the selection approach adopts a global or a hybrid selection strategy
- *Pre-processing technique*: to present the pre-processing technique if it is applied before selection
- *Selection technique*: to define the different steps of the proposed selection approach
- *Constraints*: to present the different constraints taken into account including local and global QoS and temporal constraints
- *Time-dependent QoS*: to indicate if the selection approach considers time-dependent QoS
- *Service dependencies*: to specify the dependencies considered between services including structural, QoS and temporal dependencies. The structure can be complex (C) (i.e., various structural patterns are considered) or simple (S) (i.e., only sequential patterns are considered).

2.6 Dynamic Service Selection

Service selection approaches discussed in the previous section are solely used to obtain an ex-ante service combination. During execution, the QoS of the selected services might deviate from their estimated values at design time. In fact, since service-oriented systems are very likely to be executed in uncertain and dynamic environments, several changes can affect the selected services and may lead to the violation of end-to-end global constraints. Therefore, enabling service compositions to evolve in uncertain and dynamic environments and adequately deal with violations and changes is imperative to guarantee the fulfillment of the required needs. To do so, it is necessary to detect changes in the environment as well as violations of the specified constraints during execution. The detection of changes can be achieved through various monitoring techniques [94, 95]. These techniques can be reactive [96, 97] (i.e., violations are detected only after their occurrence) or proactive [98] (i.e., potential deviations can be identified before their occurrence).

To enable the adaptation of running service compositions in order to ensure fulfilling QoS requirements, and/or to optimize the selected composition, various approaches have been proposed. These approaches allow either the adaptation of the *behavior of*

the composition or the *services forming the composition*. *Behavioral adaptation* aims to execute the service composition with respect to an alternative behavior after a possible change or a violation during execution [99]. Alternative behaviors can be obtained by changing the structure of the service composition (i.e., by changing structural patterns of the composition) and/or the granularity of services (e.g., by merging coarse-grained services into fine-grained services or inversely). *Adapting services forming the composition* at run-time (also called *dynamic service selection*) aims to adjust the selected service composition to face changes and violations during execution [67, 95, 100–103]. Dynamic service selection allows substituting involved failed services in the composition with functionally equivalent ones while fulfilling end-to-end constraints. Dynamic service selection approaches assume that an initial service combination is identified using a static selection approach and aim to dynamically re-select services according to changes that might occur during execution such that all constraints remain satisfied. In this thesis, we particularly focus on dynamic service selection approaches.

2.6.1 Dynamic Service Selection Strategies

Dynamic service selection approaches at run-time adopt either a *reactive* or a *proactive* selection strategy to adapt service compositions [104].

2.6.1.1 Reactive Service Selection

Reactive service selection approaches [67, 100, 101] react to changes and failures and deal with erroneous behaviors after their occurrence. However, handling changes once failures have occurred might lead to undesirable effects. For instance, a delayed reaction to changes may lead to the inability to find a feasible solution that satisfies all constraints or a selection of a new solution that has a lower quality compared to the solution that could be found if the reaction to changes is triggered earlier. Moreover, the late reaction to changes might cause a significant interruption time during the execution of services, which is highly undesirable mainly in time-sensitive applications. Another limitation of reactive approaches is that most of them do not take into account environment changes which should be considered to enhance the selected solution (e.g., when a new better service is added).

2.6.1.2 Proactive Service Selection

To deal with the limitations of reactive selection strategies, some approaches adopt proactive selection strategy. Proactive approaches anticipate required adaptation actions

prior to the occurrence of possible violations [95, 103, 105–107]. These approaches perform adaptation actions before reaching an erroneous state. The main advantage of this strategy is to increase the likelihood of finding a possible solution while enhancing the overall quality and avoiding the interruption of service execution due to for instance the invocation of a faulty service. These approaches can also be used to optimize the selected solution by considering the environment changes at run-time such as the addition of new better services.

2.6.2 Dynamic Service Selection Approaches

Dynamic service selection represents a broad research topic. In this section, we consider two classes of dynamic service selection: *re-selection of services* and *backup solutions*.

2.6.2.1 Re-selection of Services

Re-selection through substituting services of non-executed tasks is a popular way to ensure the fulfillment of global constraints during execution. The substitution of services can be *global* (i.e., applied to all non-executed services in the composition), *partial* (i.e., applied to a subset of non-executed services) or *local* (i.e., applied to one service in the composition). When a partial or a global substitution is required, some approaches apply the same service selection algorithm used to select the initial combination of services while considering the values of the already executed services (See Section 2.5.2).

- *Global Re-selection*

Canfora et al. [101] introduce a service re-planning approach to re-plan the service process during execution. The re-planning algorithm is triggered as long as services are executed. If a considerable deviation in QoS values is observed, the execution is stopped and the re-planning is triggered for non-executed tasks using a genetic algorithm. In [67], Ardagna et al. propose re-optimization plans when possible violations occur at run-time. The re-optimization is based on global optimal approach that re-select services for all non-executed tasks to search for the best solution by adopting integer programming. A similar approach is proposed by Zeng et al. [66] to re-select services for the non-executed part of the process each time a change occurs in an executed service. In these approaches, all potential candidate services for abstract tasks are considered in the re-selection step which can be time consuming. Moreover, temporal properties and time-dependent QoS are not taken into account.

To avoid considering all candidate services, which is not appropriate during the execution of services, Ramacher et al. identify a set of alternative services for each abstract task [108]. The approach proposed in [108] deals with uncertainties of the response time and the temporal conditions during execution based on a time-sensitive selection approach. A MIP model is defined to adjust the combination of services when a deviation or a violation is observed. If a large deviation of the start time of one abstract task is observed or the execution time constraint is expected to be violated, a re-selection action is triggered for non-executed tasks using MIP while considering only the set of pre-selected services. An interesting feature of this approach is that it takes into account time-critical service compositions while considering the temporal conditions on the execution time of the whole composition. However, this work considers only one quality attribute that should be optimized. Du et al. [109] propose a penalty-based genetic algorithm to dynamically select services for business processes under temporal constraints. The authors propose to re-select services for all tasks that have not yet been executed each time a violation of a temporal constraint occurs. Both approaches [108] and [109] can not handle environment changes such as the availability of new better services and they are only applied for corrective purposes. Moreover, while considering violations at execution time, they do not take into consideration time-dependent QoS values, which can make the re-selection problem more complex to resolve. In fact, when considering time-dependent QoS values, a violation of the execution duration of one service may lead to a change in QoS values of its successor service.

Although the previous approaches guarantee the satisfaction of global QoS constraints, they suffer from a high computation cost and require the interruption of the execution.

- *Partial Re-selection*

The algorithm proposed in [107] tries to find a replacement to the failed task from the set of its candidate services. If there is no suitable replacement, a re-selection region is increased incrementally to include other non-executed tasks until a solution is found. In [110], Lin et al. define selection regions to adapt the selected combination of services. In this work, a selection region is identified for each faulty service. These regions are enlarged using a distance measure until a satisfactory solution is found. The selection algorithm is triggered for each region to search for the optimal alternative composition. However, in these approaches, to re-select services, all candidate services are considered. A similar work that adopt a partial re-selection strategy is presented in [111] while considering only a small set of alternative candidate services for each task. Li et al. [111] identify a set of alternative candidate services for each task to reduce the number of services that have to be considered during the selection. Authors preselect two services

for each business task based on a distance measure with respect to the primary selected services. Considering only two services may lead to the inability to find a feasible solution in some cases based on the already executed services. Moreover, the measures used to select alternative services are applied locally for each abstract task and do not consider dependencies between the services of the different tasks. All the previous approaches do not consider specific characteristics related to the presence of temporal properties and can not handle changes in temporal values (e.g., start and finish time of services and temporal dependencies between services). Moreover, these approaches do not cater for optimization purposes and can only be used to find a recovery solution. Additionally, re-selection actions are taken only when a violation occurs without preventing possible violations.

As a step forward, Ismail et al. [112] address the issue of handling the violation of Service Level Agreements (SLA) while considering the time impact analysis. In other words, when a violation is observed, an impact region is defined to include all the impacted services based on the time impact conditions in order to reduce the amount of service changes during execution. In this work, only time dimensions are considered and no support to QoS values violations is proposed. Moreover, there is no indication on how new services are selected for each impact region.

- *Local Re-selection*

Local re-selection at run-time aims to substitute one service in the selected solution by another service from the set of candidate services [103]. To enhance the efficiency of the re-selection process, some approaches identify a set of alternative candidate services for each abstract task. Azmeh et al. [113] specify a set of alternative services prior to the execution. These services are considered in case of failure in order to substitute a failed service. However, in this work, only functional properties are considered. Given a workflow, in [114], Wagner et al. identify backup services for each task during the selection of services. First, functional clustering is applied on the set of services. If a service fails during execution, it is replaced by one of the backup services in its sub-cluster. The service with the shortest distance to the originally selected service is then selected to be executed. In this approach, QoS values of backup services are taken into account during the selection of services, which allows the estimation of the overall quality of the compositions of services in case of failure.

2.6.2.2 Backup Solutions

To deal with the high overhead of the re-selection of services during execution, some researches resort to the use of backup solutions. This mechanism is widely used to

adapt service compositions at run-time. Backup solutions are identified prior to the execution of the service composition so that, if a problem occurs, these solutions can be used to maintain the successful execution of services.

In [115], Chafle et al. pre-specify several alternative plans at different levels that can be selected at run-time. While this work allows for handling several changes, it may be the case where no predefined plan is feasible even though a solution to the problem does exist, which decreases the efficiency of the proposed approach. In [102], Yu et al. propose an offline algorithm to react to service changes. The proposed algorithm identifies a secondary path from each service so that if a service becomes faulty at run-time, the secondary path from this service will be used to repair the combination of services. However, only one failure can be handled, which is not appropriate in highly uncertain and dynamic environments. In [116], Ben Mabrouk et al. classify the most representative services of each abstract task according to their QoS values. A search tree is then applied to select the best combination of services during execution by checking services in an ordered way. Multiple service compositions that satisfy QoS constraints are identified during selection, so that they can be used if a change occurs during execution. Yang et al. and Dai et al. [105, 106] propose proactive selection approach based on backup solutions. When a failure occurs, the execution is switched to a backup solution. Despite the early reaction to changes, only services that succeed the failed service are considered. However, in some cases including also non-executed services that precede the affected service might lead to a better solution. Moreover, the proposed approach caters only of repair purposes and ignore optimization issues that can prevent violations.

In these approaches, all backup solutions are determined prior to the execution without considering environment changes. Considering static backup solutions at request time is not a practical solution when dealing with highly uncertain and dynamic environment and heavily constrained problems.

Barakat et al. [117] introduce a reactive service selection algorithm to deal with service changes during the selection process while dealing with dynamic backup solutions. The main idea is to react to service changes during selection so that the selected solution can be executable, satisfactory and optimal prior to execution. Authors focus on service failures and changes during the selection phase by adjusting the selected services when new information becomes available. This approach, however, suffers from a high computation cost since possible backup paths are recomputed each time a change occurs. Moreover, the proposed approach is restricted to changes during the selection phase while neglecting deviations that may occur during the execution phase.

2.6.3 A Synthesis of Dynamic Service Selection Approaches

Table 2.2 provides a synthesis of the current dynamic service selection approaches based on the following criteria.

- *Reactive/Proactive*: to indicate if the dynamic selection approach adopts a reactive or a proactive service selection strategy
- *Selection technique*: to specify the technique used to adapt the service combination during execution. If a re-selection is applied, (L), (P) and (G) indicate a local, a partial or a global re-selection, respectively. If the selection approach is based on a set of backup plans, (S) denotes static backups and (D) denotes dynamic backups plans.
- *Optimization purpose*: to indicate if the dynamic selection approach allows optimizing the selected solution or adaptation actions are only applied for corrective purposes
- *Solution optimality*: to indicate if the new solution after a re-selection is optimal
- *Environment changes*: to indicate if the environment changes are taken into account
- *Execution interruption*: to indicate if the dynamic selection approach allows for avoiding the interruption of execution
- *Time-dependent QoS*: to indicate if the selection approach considers time-dependent QoS
- *Service dependencies*: to specify the dependencies considered between services including structural, QoS and temporal dependencies.

2.7 Summary and Discussion

In recent years, QoS-aware service selection at both design time and run-time has received great importance. Despite active research, several limitations can be identified according to the research challenges presented in Chapter 1.

TABLE 2.2: A synthesis of dynamic service selection approaches.

	Reactive/ Proactive	**Selection Technique**	**Optimization Purpose**	**Solution Optimality**	**Environment Changes**	**Execution Interruption**	**Time-dependent QoS**	**Service Dependencies**
[66]	Reactive	Re-selection (G) IP	√	√	√	x	x	Structural (C)
[101]	Reactive	Re-selection (G) GA	x	x	√	x	x	Structural (C)
[67]	Reactive	Re-selection (G) MIP	√	√	√	x	x	Structural (C)
[110]	Reactive	Re-selection (P) IP	x	√	x	x	x	Structural (C)
[112]	Reactive	Re-selection (P)	x	-	x	√	x	Structural (C) Temporal (FS)
[114]	Reactive	Re-selection (L) (Backup services)	√	√	x	√	x	Structural (C)
[108]	Reactive	Re-selection (G) MIP (Backup services)	x	x	x	x	x	Structural (C) Temporal (FS)
[111]	Reactive	Re-selection (P) LP (Backup services)	x	x	x	x	x	Structural (C)
[102]	Reactive	Backup Plans (S)	x	x	x	√	x	Structural (C)
[115]	Reactive	Backup Plans (S)	x	x	x	√	x	Structural (C)
[116]	Reactive	Backup Plans (S)	x	√	x	√	x	Structural (C)
[117]	Reactive	Backup Plans (D)	√	√	√	√	x	Structural (S)
[106]	Proactive	Backup Plans (S)	x	x	x	Partially	x	Structural (C)
[105]	Proactive	Backup Plans (S)	x	x	x	Partially	x	Structural (C)
[103]	Proactive	Re-selection (L)	x	x	x	Partially	x	Structural (C)

- *Constraints and Dependencies between Services*

  Structural constraints between services in a composition are specified by sequential dependencies in several approaches. This is very restrictive to effectively present structural dependencies between abstract tasks in a business process. Moreover, existing attempts to service selection usually consider only structural dependencies between services and neglect QoS and temporal constraints and dependencies that might be specified at the business level (See Table 2.1). However, in various cases, several constraints and dependencies may arise between services in order to gain control over the successful execution of business tasks. Hence, several service selection approaches can not be used in heavily constrained selection problems and they are only suitable for simple problems where all tasks are supposed to be executed in sequence without considering dependencies between them.

- *Time-dependent QoS*

  According to the state-of-the art discussed in the previous sections, most of current QoS-aware service selection approaches usually consider static QoS values. Unlike static QoS values, which have been deeply studied in the existing service selection approaches, time-dependent QoS are insufficiently taken into consideration. In real-world applications, most of QoS attributes (e.g., response time, cost) are time-dependent. As a result, in different time periods, QoS values may considerably vary. Considering only static QoS values prevents from capturing the variations of QoS values over time. In addition, this may result in inaccurate QoS representation that might lead to service compositions that are not satisfactory at run-time.

- *Scalability and Optimality*

  Different service selection approaches have been proposed to select optimal and close-to-optimal solutions. These methods rather adopt exact or approximate selection strategies that can be used to select the primary service selection or to re-select services at run-time. On one hand, exact solutions usually suffer from scalability issues and poor efficiency especially in large-scale selection problems. This is highly undesirable notably when re-selecting services at run-time since it may cause a long interruption of the service execution. On the other hand, approximate approaches generally degrade the quality of the selected solution. Hence, it is essential to find a trade-off between the computation time required to find a feasible solution and the optimality of the selected solution. Several approaches propose pre-processing techniques to reduce the search space prior to the selection process. These techniques usually rely on greedy reduction space techniques that can prune candidate services that can be part of the optimal solution.

- *Uncertainty and Dynamic Environments*

Dealing with changes and violations during service execution is necessary to guarantee the successful execution of service compositions. Dynamic service selection approaches can be reactive (i.e., they take adaptation actions after a violation occurs) or proactive (i.e., they take adaptation actions at early stages without delaying the reaction until the faulty service is executed) (See Table 2.2). Most of current dynamic service selection approaches follow a reactive selection strategy and do not allow proactive selection during service execution. Reactive approaches suffer from significant interruption time. Some efforts have been proposed to resolve this issue by identifying backup solutions or by reducing the number of services considered in the re-selection step. Current approaches suppose that the set of backup services and solutions is static (i.e., does not change during execution), which may deteriorate the efficiency of the selection algorithm especially in highly dynamic and uncertain environments when various changes may be observed during execution. In fact, identifying the set of backup services and solutions during the initial selection might lead to undesirable effects (e.g., there is no backup service that can participate in a satisfactory solution after a violation). While proactive approaches handle erroneous behaviors at early stages, most state-of-the-art approaches trigger re-selection only for repair purposes and neglect optimization purposes.

2.8 Conclusion

In this chapter, we have analyzed and discussed related work on static and dynamic service selection approaches while considering QoS and temporal constraints. This analysis shows that although the service selection problem has been widely treated in literature and has received the attention of several researchers, there are still some limitations that have to be considered.

In this thesis, we aim to overcome these limitations. In particular, to allow adequate evaluation of QoS variations over time, we consider *time-dependent QoS*. Moreover, in order to effectively capture the dependencies between services, we provide a rich model of the service selection problem. This model characterizes QoS, temporal and structural constraints through handling complex business structures.

To deal with scalability and optimality issues, we develop *pruning techniques* to remove inadequate services without any impact on the optimality. An *exact and approximate service selection* algorithms are then presented to find an optimal and a near-to-optimal solution, respectively, while reaching a reasonable computation time. Finally, we propose a *proactive dynamic service selection* approach to handle changes and violations during

execution. A key feature of the dynamic service selection approach is the early reactions to changes to avoid possible violations at run-time and allow finding solutions with good quality values. The different contributions of the proposed approach are discussed in the next chapters.

Chapter 3

Specification of The Service Selection Model

Contents

3.1 Introduction

Service selection problem we are interested in consists on finding the adequate services so that constraints at the business and the service levels are satisfied. In this thesis, we assume that an abstract process model is already given by a business designer prior to the selection of service combinations. In this chapter, we introduce the specification of the

service selection problem as well as the notations used throughout the thesis. In Section 3.2, we give a detailed description of the different business constraints. This includes structural, QoS and temporal constraints that may be specified by business designers and global user constraints. In Section 3.3, we introduce service level constraints. In Section 3.4, we define the quality model of the composite service followed by a definition of the constraint optimization problem. Finally, in Section 3.5, we conclude the chapter.

3.2 Business Level Constraints

A business process is usually defined by a set of activities (or abstract tasks) hereafter, denoted by $\mathcal{A} = \{A_1, ..., A_n\}$. Several constraints may be specified at the business level. In the following, we present the constraints considered in our work (i.e., structural, QoS and temporal constraints).

3.2.1 Structural Constraints

Usually, abstract tasks of each business process have structural dependencies between them. We assume that each business process is characterized by a single initial task and a single end task. We denote by $Pd(A_i)$ the set of immediate predecessors of the task $A_i \in \mathcal{A}$.

We consider the four commonly adopted structural patterns, which cover most of the structures specified by composition languages such as BPEL (Business Process Execution Language) [118]. A more detailed classification, including other structural patterns is provided in [36].

- *Sequence*: in this pattern, activities are executed in sequence. For example, in Figure 3.1(a), the task A_j is executed after the task A_i is completed. We denote by $\mathcal{S}$ the set of activities that belong to a sequence structure.
- *Parallel (AND)*: this pattern indicates that activities are executed in parallel (Figure 3.1(b)). It involves the AND-split and AND-join flow patterns, which divide the control into parallel branches, and merge these branches, respectively. We denote by $\mathcal{P}$ the set of activities that belong to a parallel structure and $\mathcal{SP}$ the set of parallel structures. $P_l \in \mathcal{SP}$ indicates the parallel structure l.
- *Exclusive Choice (XOR)*: exclusive choice pattern indicates that only one activity can be executed in each choice structure (Figure 3.1 (c)). We denote by $\mathcal{C}$ the set of activities that belong to a choice structure. In the following, $\mathcal{SC}$ indicates the

set of choice structures and $C_l \in \mathcal{SC}$ indicates the choice structure l. For each choice structure $C_l \in \mathcal{SC}$, p_{li} is the probability to execute the branch of a task A_i, $\forall A_i \in C_l$. The probability of each choice branch is a value in [0,1] s.t. $\sum_{i=1}^{nC_l} p_{li} = 1$ and nC_l is the number of disjoint branches in the choice structure C_l.

- *Loop*: loop pattern indicates that activities can be executed in an iterative manner. In the following, we denote by $\mathcal{L}$ the set of activities that will be executed iteratively. We suppose that their is an upper bound for the number of iterations of each loop. The maximum number of iterations can be identified through a history based estimation and from previous task executions. The maximum number of loops of each activity $A_i \in \mathcal{L}$ is denoted by α_i (Figure 3.1 (d)).

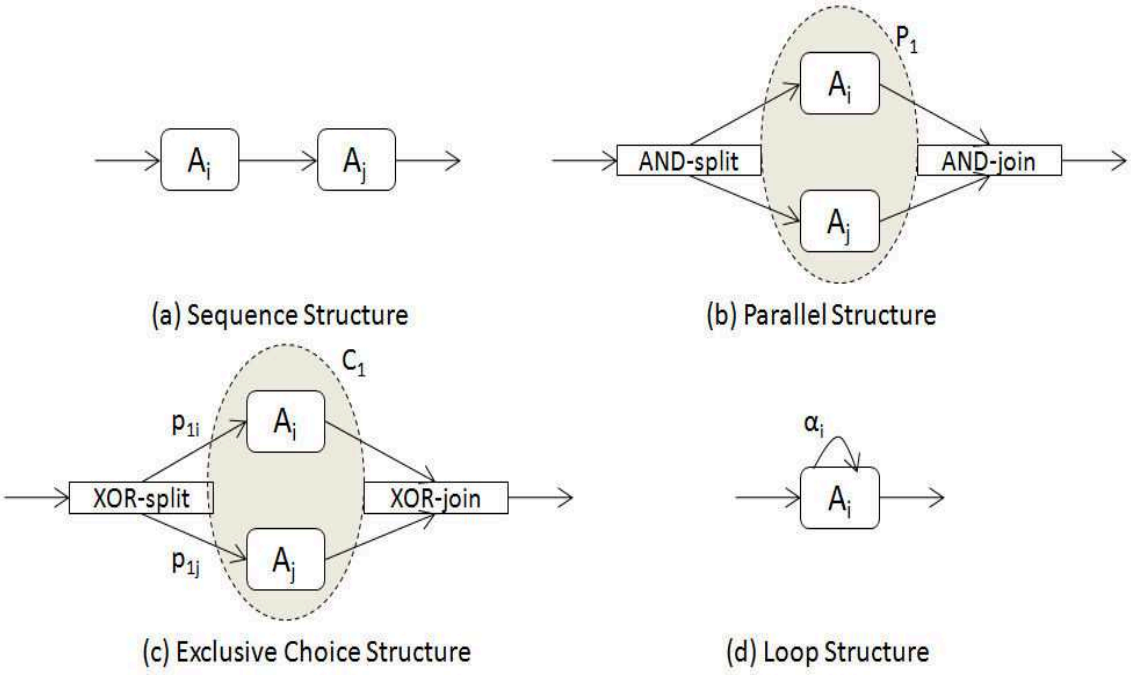

FIGURE 3.1: Common structural patterns

3.2.2 Global Constraints

In order to select the best composite service CS (i.e., the best combination of services), the user can specify in his request a set of attributes whose values must be satisfied (i.e., *constraint attributes*) and a set of attributes whose values must be optimized (i.e., *objective attributes*). Constraint attributes are usually specified as *global constraints* (e.g., the user requires that the cost of the composite service does not exceed 20 units). These constraints are based on non-temporal QoS attributes (e.g., cost, reliability) and on temporal attributes (e.g., execution time, deadline). For simplicity, in what follows all attributes specified in the user requirements are considered as QoS attributes. Objective

attributes define the attributes that the user wants to maximize or minimize (e.g., the user desires to minimize the cost of the composite service). In this thesis, constraint and objective attributes are handled in the same way (i.e., all attributes have to be fulfilled and optimized).

Let $\mathcal{QS}$ denoting the set of user's QoS attributes. The y^{th} QoS attribute is denoted by q_y s.t., $q_y \in \mathcal{QS}$ with $1 \leq y \leq m$ where m is the number of global QoS constraints. We consider four categories of QoS attributes that are widely used in the literature: Additive, Average, Multiplicative and Max-Operator [37, 73, 124]. Further details about these categories are presented in the next sections. Each QoS attribute $q_y \in \mathcal{QS}$ has either:

- an *increasing better value direction* (i.e., the quality is better when the attribute value increases)
- a *decreasing better value direction* (i.e., the quality is better when the attribute value decreases).

On the other hand, QoS attributes can be divided into two classes:

- *quantitative attributes* that can be measured using metrics (e.g., availability, response time)
- *qualitative attributes* that can not be measured and they are generally evaluated based on boolean values (e.g., privacy, security)

For the sake of simplicity, henceforth, we do not consider QoS attributes with increasing value direction since they can be easily transformed to decreasing value direction based attributes by multiplying their values by -1. In addition, we consider only quantitative attributes. Qualitative attributes can be considered as quantitative attributes that can be measured based on two boolean metrics (i.e., 0 and 1).

In the following, $Q(q_y)$ denotes the value of the global constraint of the QoS attribute q_y of the composite service (e.g., $Q(cost) = 68$ indicates that the value of the global cost of the composite service is equal to 68 cost units). Note that since we consider only quantitative attributes with decreasing value direction, only upper bound QoS constraints are taken into account when dealing with global user constraints. Moreover, all attributes have to be minimized. To represent his preferences, the user may specify a weight for each QoS attribute q_y denoted by W_y, s.t., $\sum_{q_y \in \mathcal{QS}} W_y = 1$. Otherwise, all quality attributes are considered at the same level of preferences.

3.2.3 QoS Constraints

In addition to structural constraints, business designers may specify a set of *intra-task QoS constraints* at the business level denoted by $\mathcal{QC}$ [67]. Intra-task QoS constraints define quality for a given task in the process. Each intra-task QoS constraint $qc_i(TP, V, q_y) \in \mathcal{QC}$ is characterized by:

- the activity $A_i \in \mathcal{A}$ concerned by the QoS constraint
- the type of the QoS constraint TP, which can be either MIN to denote a lower bound constraint or MAX to denote an upper bound constraint (i.e., $TP \in \{$MIN, MAX$\}$)
- a value V that must not be exceeded
- a QoS parameter $q_y \in \mathcal{QS}$ for which the constraint is applied

For instance, in the motivating scenario presented in Section 1.3, the constraint QC_5 can be denoted by $qc_5(MAX, 14, cost)$ to indicate that the cost of the task A_5 must be less than 14.

Some approaches consider dependencies between QoS values of interacting services [1, 14, 119]. For instance, the QoS value of a service may depend on the QoS value of another service. In this thesis, we suppose that there are no QoS dependencies between the services of the different business tasks.

3.2.4 Temporal Constraints

Temporal constraints may also be associated with business processes. We distinguish between intra- and inter-task temporal constraints [7, 120].

- *Intra-task temporal constraints*: relate to the start and the finish time of each task. We denote by $\mathcal{TC}$ the set of intra-task temporal constraints. A temporal constraint $tc_i(TP, T) \in \mathcal{TC}$ is characterized by:
  - the activity concerned by the temporal constraint (i.e., $A_i \in \mathcal{A}$)
  - a type $TP \in \{$must start on (MSO), must finish on (MFO), start no earlier than (SNET), finish no earlier than (FNET), start no later than (SNLT), finish no later than (FNLT)$\}$
  - a time point T

For example, the constraint TC_2 in the scenario presented in Section 1.3 is denoted by $tc_2(FNLT, 15)$ to indicate that the task A_2 must finish no later than 3 p.m.

- *Inter-task temporal constraints*: specify temporal dependencies between tasks. These dependencies define the time lags between two directly or indirectly succeeding tasks to restrict the time span between them. The set of inter-task temporal constraints is denoted by $\mathcal{TD}$. Each temporal dependency $td_{i,v}(TP, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD}$ is characterized by:
  - a source and a destination tasks $A_i \in \mathcal{A}$ and $A_v \in \mathcal{A}$
  - a type $TP \in$ {start-to-start (SS), start-to-finish (SF), finish-to-start (FS), finish-to-finish (FF)}
  - a minimum and a maximum duration between the source and the destination tasks (i.e., A_i and A_v) denoted by D_{iv}^{min} and D_{iv}^{max}, respectively

  For instance, the temporal dependency $TD_{1,3}$ in Section 1.3 is denoted by $td_{1,3}(SF, 1, 12)$ to indicate that the task A_3 has to finish no earlier than 1 time unit and no later than 12 time units after the start of the task A_1.

Some work extends the BPMN language to handle temporal properties in business processes [7, 120–122]. We argue that these approaches can be used to specify intra- and inter-task constraints at the business level. In this thesis, we assume that business processes are well-structured and that all constraints are verified (i.e., there are no conflicts between them). The verification of the consistency of business processes while considering several constraints is widely treated in literature [7, 123]. This step it is out of the scope of this thesis.

3.3 Service Level Constraints

Apart from constraints specified at the business level, other constraints may also be defined at service level. Each activity A_i of a business process has a set $\mathcal{S}_i$ of potential candidate services. The candidate services of an activity A_i (i.e., a service class) are functionally equivalent and can be distinguished by their QoS attributes.

3.3.1 Time-dependent QoS

As stated in Chapter 1, in this thesis, we assume that QoS values of candidate services may change over the time. For instance, services may have temporal constraints related

to their availabilities (e.g., a service can be unavailable each day from 8 p.m to 11 p.m). Moreover, a service may assign temporal constraints to QoS attributes, called time-dependent QoS. For example, a service offers a smaller response time from 7 p.m to 10 p.m.

Each service $S_{ij} \in \mathcal{S}_i$ is characterized by a set $\mathcal{T}_{ij}$ of disjoint time intervals during which it offers different QoS values. To capture QoS variations related to these time-dependent QoS, we introduce the notion of *timed service instance* of candidate services. Each timed service instance (service instance for short) is associated with a time interval. We denote by S_{ijk} the k^{th} timed instance of the service S_{ij} corresponding to the time interval $T_{S_{ijk}} \in \mathcal{T}_{ij}$. Each time span $T_{S_{ijk}}$ is characterized by absolute minimum start and maximum finish time values denoted by $t^{min}_{S_{ijk}}$ and $t^{max}_{S_{ijk}}$, respectively.

We denote by $Q(S_{ijk}, q_y)$ the value of the QoS attribute $q_y \in \mathcal{QS}$ offered by the service S_{ij} at the time span $T_{S_{ijk}}$. Given our motivating scenario presented in Section 1.3, service instances of the different candidate services are presented in Figure 3.2. In this example, for instance, $Q(S_{132}, cost)$=19 denotes that the cost of the service instance S_{132} of the service S_{13} is equal to 19 cost units. The QoS values of this service instance are offered in the time span [14,20].

	Time Span	Duration	Cost
S_{111}	[8, 15]	4	12
S_{121}	[17, 21]	2	8
S_{131}	[9, 13]	3	23
S_{132}	[14, 20]	4	19

	Time Span	Duration	Cost
S_{211}	[12, 17]	1	10
S_{221}	[7, 13]	4	12
S_{231}	[9, 12]	2	17
S_{232}	[14, 21]	1	15

	Time Span	Duration	Cost
S_{311}	[13, 22]	4	12
S_{321}	[8, 14]	6	12
S_{331}	[10, 17]	4	13
S_{332}	[17, 23]	5	23

	Time Span	Duration	Cost
S_{411}	[6, 13]	7	22
S_{412}	[13, 23]	9	18
S_{421}	[17, 24]	6	15
S_{431}	[12, 20]	4	12

	Time Span	Duration	Cost
S_{511}	[7, 12]	1	8
S_{512}	[15, 20]	1	14
S_{521}	[8, 16]	3	20
S_{531}	[14, 22]	2	12

	Time Span	Duration	Cost
S_{611}	[19, 22]	1	7
S_{621}	[9, 13]	6	9
S_{622}	[14, 18]	4	10
S_{631}	[8, 20]	1	6

FIGURE 3.2: Candidate service instances of our motivating scenario

The set of QoS parameters as well as their corresponding intervals (i.e., the intervals during which a service offers the announced QoS) each service instance s_i of a task A_i establishes a $(m+2)$-dimensional parameter vector denoted by $PV^{s_i}\langle p_1^{s_i}, .., p_y^{s_i}, .., p_{m+2}^{s_i}\rangle$ with $p_y^{s_i} = Q(s_i, q_y), \forall 1 \leq y \leq m$, $p_{m+1}^{s_i}$ is the minimum start time of the service instance s_i (i.e., $t^{min}_{s_i}$) and $p_{m+2}^{s_i}$ is the maximum finish time of s_i (i.e., $t^{max}_{s_i}$).

3.3.2 QoS Distributions and Patterns

QoS values of each service class may follow different distributions that may affect the performance of the selection problem. Moreover, when considering temporal properties, we distinguish several patterns and schemes for QoS values according to the time spans in which they are specified. In this section, we give examples of QoS distributions and patterns.

3.3.2.1 QoS Distributions

Three types of distributions of quality attributes are defined in [53]. These distributions are presented in Figure 3.3 while considering two-dimensional search space:

- *Independent*: the values of quality attributes are independent of each others
- *Correlated*: if the value of one quality attribute of a service is good, the values of the other quality attributes of this service are also good
- *Anti-correlated*: if the value of one quality attribute of a service is good, the values of the other quality attributes are bad

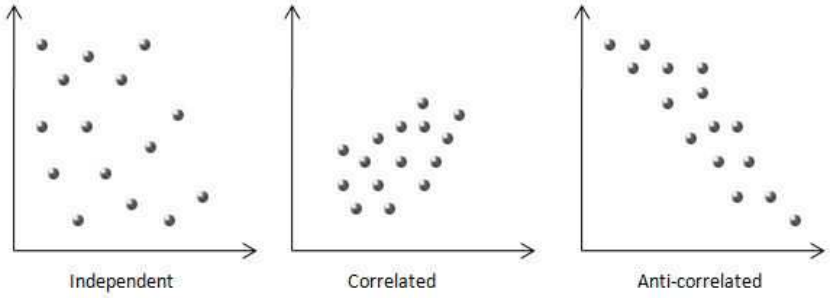

FIGURE 3.3: Different distributions of quality attributes

In the following chapters, we explain how these distributions may influence the performance of the selection process.

3.3.2.2 Patterns of Time-dependent QoS

Time-dependent QoS was introduced in several approaches in multiple domains such as [1, 2, 14, 125–128]. In these approaches, QoS attributes can have different schemes and patterns.

In [1], two QoS schemes have been introduced: *saturation* and *repeated*. Figure 3.4 shows the different QoS schemes that can be used to present time-dependent QoS. In

this example, authors expose the variations of the values of the price of a compress movie service. The repeated scheme is used when the variation of QoS values is repeated over a time period (e.g., the price of the service is more expensive during weekdays then in the weekends). The saturation scheme is used when QoS values follow a monotonous variations (e.g., the price of the compress movie service is more expensive if the user want to watch the film right after the release date of the film). In [14], authors assume that if QoS values of a service instance do not change monotonically, this instance can be split into several service instances.

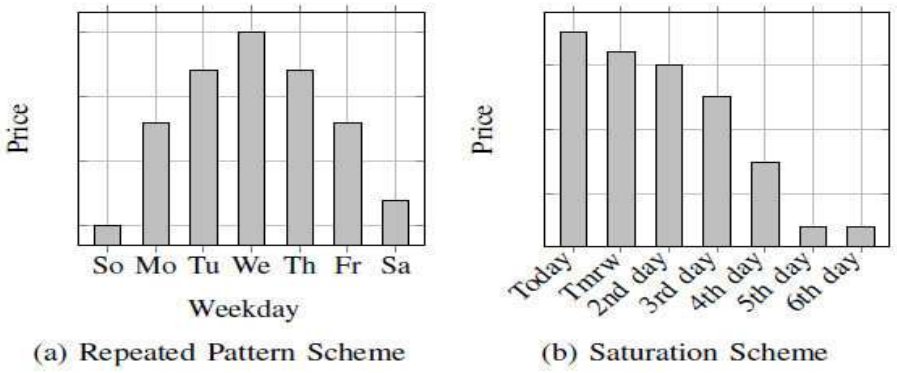

FIGURE 3.4: Different schemes for time-dependent QoS [1]

QoS variations may also have irregular scheme or change over the days [2] (See Figure 3.5).

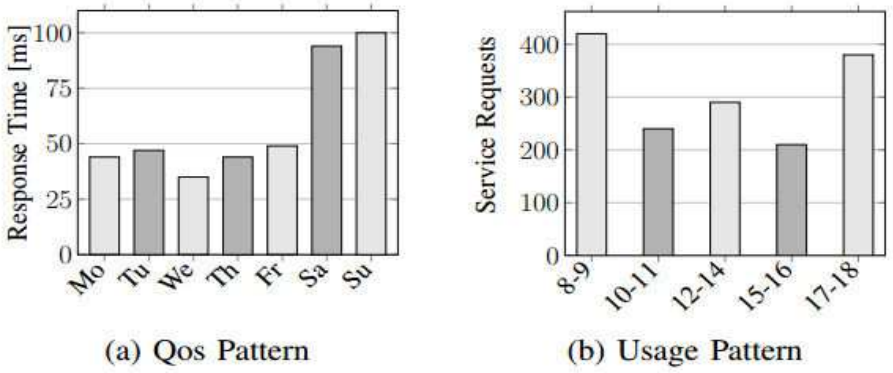

FIGURE 3.5: Examples of QoS patterns [2]

An extension to the WS-Agreement specification to cater for temporal properties in service offers has been proposed in [125, 129]. This extension allows the specification of time cycles and periods in service offers. An example of temporal-aware agreement template and offer proposed in [129] is given in Appendix C. In this thesis, we do not consider any special scheme or pattern for time-dependent QoS values and we suppose that time-dependent QoS models are defined by service providers. The specification of these models can be achieved based on the history of previous service invocations [51] or using existing QoS prediction methods [13, 128]. Moreover, existing work such as

[125, 126] can be used to model service offers while considering time-dependent QoS values.

3.4 Composite Service Quality Model

QoS-based service selection requires the evaluation of the QoS of service compositions in order to select the best solution based on estimated QoS values of services.

3.4.1 Aggregation Function

The value of a QoS attribute q_y for a composite service CS is denoted by $Q(CS, q_y)$. It is computed by the aggregation of the corresponding quality values of its elementary services. The aggregation function Agg depends on the distinguish characteristics of quality attributes (i.e., Additive, Average, Multiplicative and Max-Operator) and the structure of the business process (i.e., the structural patterns involved such as sequence, parallel, choice and loop patterns). Table 3.1 shows examples of aggregation functions considered in this thesis with n is the number of activities. Thus, $Q(CS, q_y) = Agg_{A_i \in \mathcal{A}}(Q(A_i, q_y))$, where, $Q(A_i, q_y)$ denotes the value of the quality attribute q_y for the component service corresponding to the task A_i.

TABLE 3.1: Examples of aggregation functions.

Category	Sequence	Parallel	Choice	Loop
Additive	$\sum_{i=1}^{n} Q(A_i, q_y)$	$\sum_{i=1}^{n} Q(A_i, q_y)$	$\sum_{i=1}^{n} p_{li} Q(A_i, q_y)$	$\alpha_i Q(A_i, q_y)$
Average	$\frac{1}{n} \sum_{i=1}^{n} Q(A_i, q_y)$	$\frac{1}{n} \sum_{i=1}^{n} Q(A_i, q_y)$	$\sum_{i=1}^{n} p_{li} Q(A_i, q_y)$	$\alpha_i Q(A_i, q_y)$
Multiplicative	$\prod_{i=1}^{n} Q(A_i, q_y)$	$\prod_{i=1}^{n} Q(A_i, q_y)$	$\sum_{i=1}^{n} p_{li} Q(A_i, q_y)$	$Q(A_i, q_y)^{\alpha_i}$
Max-operator	$\sum_{i=1}^{n} Q(A_i, q_y)$	$max_{i=1}^{n}\{Q(A_i, q_y)\}$	$\sum_{i=1}^{n} p_{li} Q(A_i, q_y)$	$\alpha_i Q(A_i, q_y)$

In the following, we define the aggregation function of a whole composition according to the different categories of quality parameters by exploring the structure of the composition. For simplicity, we denote by Agg the aggregation function $Agg_{A_i \in \mathcal{A}}(Q(A_i, q_y))$.

3.4.1.1 Additive Attributes

The value of an additive attribute (e.g., the execution cost) for a composite service can be determined through the sum of the values of this attribute for all the component

services. To measure the local thresholds for an additive attribute, the aggregation function can be presented as follows:

$$Agg = \sum_{A_i \in \mathcal{S}} Q(A_i, q_y) + \sum_{A_i \in \mathcal{P}} Q(A_i, q_y) + \sum_{C_l \in \mathcal{SC}} \sum_{A_i \in C_l} p_{li} Q(A_i, q_y) + \sum_{A_i \in \mathcal{L}} \alpha_i Q(A_i, q_y) \quad (3.1)$$

3.4.1.2 Average Attributes

In this category, the value of the attribute of the composite service is measured by the average of the values of the attribute of its atomic services (e.g., the availability). To compute local thresholds of average attributes, we propose the following aggregation function with ns is the number of component services.

$$Agg = \frac{1}{ns}\left(\sum_{A_i \in \mathcal{S}} Q(A_i, q_y) + \sum_{A_i \in \mathcal{P}} Q(A_i, q_y) + \sum_{C_l \in \mathcal{SC}} \sum_{A_i \in C_l} p_{li} Q(A_i, q_y) + \sum_{A_i \in \mathcal{L}} \alpha_i Q(A_i, q_y)\right) \quad (3.2)$$

3.4.1.3 Multiplicative Attributes

Multiplicative attributes of composite services (e.g., the reputation) can be computed by multiplying the values of the attribute of the component services. Constraint (3.3) presents the aggregation function of component services values.

$$Agg = \prod_{A_i \in \mathcal{S}} Q(A_i, q_y) * \prod_{A_i \in \mathcal{P}} Q(A_i, q_y) * \prod_{C_l \in \mathcal{SC}} \left(\sum_{A_i \in C_l} p_{li} Q(A_i, q_y)\right) * \prod_{A_i \in \mathcal{L}} Q(A_i, q_y)^{\alpha_i} \quad (3.3)$$

3.4.1.4 Max-Operator Attributes

These attributes differ from other attribute categories in that different aggregation functions are used in sequential and parallel structures (See Table 3.1). Max-operator attributes for a composite service are measured by the sum of attribute values of its atomic services in a sequential structures, and the highest branch value in each parallel structure (e.g., the execution time). Thus, the aggregation function used to check the satisfaction of the global QoS constraint is as follows:

$$Agg = \sum_{A_i \in \mathcal{S}} Q(A_i, q_y) + \sum_{P_l \in \mathcal{SP}} max_{A_i \in P_l}\{Q(A_i, q_y)\} + \sum_{C_l \in \mathcal{SC}} \sum_{A_i \in C_l} p_{li} Q(A_i, q_y) + \sum_{A_i \in \mathcal{L}} \alpha_i Q(A_i, q_y) \quad (3.4)$$

3.4.2 Utility Function

To evaluate the quality of the composite service based on user preferences, we define an *utility function*. This latter is a normalized function whose values range over [0,1]. It enables the aggregation of the service composition quality values into a single value while considering user preferences in order to select the best services. Therefore, the utility value of a composite service CS is computed as follows using the Simple Additive Weighting method (SAW) [44]:

$$U(CS) = \sum_{y=1}^{m} W_y * \frac{Q(q_y)^{max} - Q(CS, q_y)}{Q(q_y)^{max} - Q(q_y)^{min}} \tag{3.5}$$

Where $Q(q_y)^{min}$ and $Q(q_y)^{max}$ denote respectively, the minimum and maximum aggregated values of the y^{th} quality attribute of CS with $Q(q_y)^{max} - Q(q_y)^{min} \neq 0, \forall q_y \in \mathcal{QS}$. These values are computed as follows:

$$Q(q_y)^{min} = Agg_{A_i \in \mathcal{A}}(Q(A_i, q_y)^{min})$$

and

$$Q(q_y)^{max} = Agg_{A_i \in \mathcal{A}}(Q(A_i, q_y)^{max})$$

with

$$Q(A_i, q_y)^{min} = min\{Q(S_{ijk}, q_y) \mid S_{ij} \in \mathcal{S}_i, T_{S_{ijk}} \in \mathcal{T}_{ij}\}$$

and

$$Q(A_i, q_y)^{max} = max\{Q(S_{ijk}, q_y) \mid S_{ij} \in \mathcal{S}_i, T_{S_{ijk}} \in \mathcal{T}_{ij}\}$$

In what follows, we denote by $t_{A_i}^{min}$ and $t_{A_i}^{max}$, the minimum possible start and maximum finish time of the task A_i, respectively, $\forall A_i \in \mathcal{A}$, with:

$$t_{A_i}^{min} = min\{t_{S_{ijk}}^{min} \mid S_{ij} \in \mathcal{S}_i, T_{S_{ijk}} \in \mathcal{T}_{ij}\}$$

and

$$t_{A_i}^{max} = max\{t_{S_{ijk}}^{max} \mid S_{ij} \in \mathcal{S}_i, T_{S_{ijk}} \in \mathcal{T}_{ij}\}$$

Note that since the minimum and maximum values of all domains are independent of user requirements, they can be determined at the design time and then, they can be continuously updated in response to the environment changes (e.g., the addition of a new service, the change of quality values of one or more services).

3.4.3 Optimal Service Composition

Constraint programming is a competitive paradigm for solving single-objective constraint optimization problems (COP) (See Section 2.4.2). In this thesis, we propose to use the COP formalism in the different steps of our approach. The choice of COP is based on its simplicity to model several real-world constraint-based problems due to its natural expressiveness and the efficiency of the existing underlying solvers. Hereafter, we give definitions of both CSP and COP as well as feasible and optimal solutions.

Definition 3.1. (Constraint Satisfaction Problem (CSP)). A CSP is a triplet $P = (X, D, C)$ where:

- X is an n-tuple of *variables* that can take values in certain ranges with $X = \{x_1, x_2, ..., x_n\}$
- D is an n-tuple of *domains*, which represent the possible values that each variable can take with $D = D_1 \times D_2 \times ... \times D_n$ such that D_i is the domain of the variable $x_i \ \forall \ i = 1, 2, ..., n$
- C is a t-tuple of *constraints*, which state the relations between the different variables with $C = \{C_1, C_2, ..., C_t\}$

A CSP is said satisfied or feasible if it has at least one feasible solution.

Definition 3.2. (Feasible Solution). A feasible solution of a CSP is an n-tuple $d \in D$ that gives a proper assignment of domain values to all variables in X where $d = (d_1, d_2, ..., d_n)$ and $d_i \in D_i \ \forall \ i = 1, 2, ..., n$ such that all constraints in C are satisfied simultaneously.

Definition 3.3. (Constraint Optimization Problem (COP)). A COP is a CSP that aims to optimize (minimize or maximize) an objective function. It can be denoted by a quadruple $P' = (P, f)$ where:

- $P = (X, D, C)$ is a CSP
- $f : D \rightarrow \mathbb{R}$ is the objective function of P'

Definition 3.4. (Optimal Solution). An optimal solution of a COP is a solution of P that is optimal with respect to f.

A solution to the service selection problem is then, a combination of concrete services (each service implements one abstract business task) that complies with business constraints and satisfies all global user constraints while optimizing the overall utility (i.e.,

the objective function). Further details about the service selection approach will be given in Chapter 5.

In the following, we denote by $CS^* = \{s_1^s, ..., s_i^s, ..., s_n^s\}$ the selected solution where s_i^s denotes the selected service for the task A_i. Thus, $Q(CS^*, q_y) = Agg_{A_i \in \mathcal{A}}(Q(s_i^s, q_y))$, $\forall A_i \in \mathcal{A}$. Each selected service s_i^s has a set of quality values as well as two temporal values, which are identified considering the set of all selected services: the estimated start time and the estimated finish time denoted by st_i and ft_i, respectively.

3.5 Conclusion

In this chapter, we have presented the different constraints we consider to tackle the service selection problem. We first introduced the different constraints at business and service level. The proposed model assumes that business processes are well defined and well structured (i.e., there are no conflicts between the involved constraints). In this thesis, we consider complex business structures while handling several categories of QoS attributes and different types of temporal properties and dependencies between services.

QoS-based service selection should guarantee that all constraints are fulfilled. In this context, we characterized the optimal service composition problem which can be modeled as a COP which relies on the service composition quality model we introduced. In the next chapter, we present our pruning techniques in order to reduce the search space and enhance the efficiency of the service selection process.

Chapter 4

Service Pruning Approach

Contents

4.1 Introduction

Intuitively, to select the best combination of services, all candidate services are considered [66, 67, 76]. However, this is impractical when the number of services increases since the time needed to solve the service selection problem becomes exponential. Moreover, the presence of time-dependent QoS and temporal constraints increases the number of possible solutions and decision variables. For instance, given a business process with 6 tasks, 500 candidate services for each task and two timed instances for each service, the number of possible combinations of services is $(2 * 500)^6$. However, not all services are potential candidates for the feasible solution.

To handle this issue, we propose a service pruning approach, which aims to reduce the number of candidate services for each business task. The goal is to reduce the number of possible combinations of services that have to be considered while guaranteeing that the optimal solution still be found. On the other side, we intend to avoid discarding any candidate service that might be part of a feasible solution. To do so, we have proposed two pruning strategies : *dominance-based pruning* and *constraint-based pruning*. Furthermore, in case there is no feasible solution to the selection problem, the pruning process allows for identifying the cause of the failure at earlier stages (i.e., before performing the selection process). Based on this, we propose improvement strategies to enhance the selection problem.

To summarize, the aim of the pruning process is two-fold: (1) it allows reducing the search space and thus, enhancing the efficiency of the selection process; (2) it helps to improve the selection problem by detecting at earlier stages possible causes of failures even before performing the selection process.

This chapter is organized as follows. In Section 4.2, we present our dominance-based pruning strategy. Section 4.3 details the constraint-based pruning techniques. Improvement strategies are proposed in Section 4.4 to enhance the service selection problem prior to the selection process. Finally, Section 4.5 concludes the chapter.

4.2 Dominance-based Pruning Process

In order to reduce the search space of each service class while considering time-dependent QoS values, we propose to extend the notion of the dominance relationship introduced in the literature [51, 53, 93]. The dominance-based pruning process depends on the characteristics of candidate services without the need of using a specific ranking function. However, when defining the set of non-dominated services for each service class (i.e., skyline services), the large number of services can introduce significant computational overhead for the pairwise comparison. To deal with this, dominant services are identified offline since they are independent of the user's requirements.

Definition 4.1. (Skyline Services). Given a service class $\mathcal{S}_i$ for a specific business task $A_i \in \mathcal{A}$ and two service instances $s_1, s_2 \in \mathcal{S}_i$, s_1 dominates s_2 (denoted by $s_1 \succ s_2$) iff s_1 is better than or equal to s_2 in all parameters and strictly better in at least one parameter. In other words, $\forall 1 \leq y \leq m+2$, $p_y^{s_1} \succeq p_y^{s_2}$ and $\exists\ 1 \leq k \leq m+2$ s.t., $p_k^{s_1} \succ p_k^{s_2}$ with:

$$p_y^{s_1} \succeq p_y^{s_2} \Longrightarrow \begin{cases} Q(s_1, q_y) \leq Q(s_2, q_y) & \text{if} \quad 1 \leq y \leq m \\ t_{s_1}^{min} \leq t_{s_2}^{min} & \text{if} \quad y = m+1 \\ t_{s_1}^{max} \geq t_{s_2}^{max} & \text{if} \quad y = m+2 \end{cases} \tag{4.1}$$

And

$$p_y^{s_1} \succ p_y^{s_2} \Longrightarrow \begin{cases} Q(s_1, q_y) < Q(s_2, q_y) & \text{if} \quad 1 \leq y \leq m \\ t_{s_1}^{min} < t_{s_2}^{min} & \text{if} \quad y = m+1 \\ t_{s_1}^{max} > t_{s_2}^{max} & \text{if} \quad y = m+2 \end{cases} \tag{4.2}$$

Hence, the set of skyline services of a service class $\mathcal{S}_i$ (denoted by $\mathcal{S}_{Skyi}$) is constituted of the candidate services in $\mathcal{S}_i$ that are not dominated by any other service in the same class. This can be achieved iff, for two service instances $s_1, s_2 \in \mathcal{S}_i$, one of the following cases is true:

- $\forall 1 \leq y \leq m$, $Q(s_1, q_y) \leq Q(s_2, q_y)$ and $\exists\ 1 \leq y \leq m$ s.t. $Q(s_1, q_y) < Q(s_2, q_y)$ and $t_{s_1}^{min} \leq t_{s_2}^{min}$ and $t_{s_1}^{max} \geq t_{s_2}^{max}$

- $\forall 1 \leq y \leq m$, $Q(s_1, q_y) \leq Q(s_2, q_y)$ and $t_{s_1}^{min} \leq t_{s_2}^{min}$ and $t_{s_1}^{max} > t_{s_2}^{max}$

- $\forall 1 \leq y \leq m$, $Q(s_1, q_y) \leq Q(s_2, q_y)$ and $t_{s_1}^{min} < t_{s_2}^{min}$ and $t_{s_1}^{max} \geq t_{s_2}^{max}$

Example 4.1. To better illustrate, let's consider the example presented in Figure 4.1, which shows the set of dominated and non-dominated candidate service instances for a given business task. Service instances are represented as points in two-dimensional space according to their QoS values. Each service instance is defined by two QoS parameters (i.e., the execution duration and the cost) and a time interval that defines its start and finish time. If we consider only static QoS, the set of skyline services is presented by red rectangles in Figure 4.1, since they are not dominated by any other services based on their QoS values. However, when dealing with time-dependent QoS, services presented by green triangles will also be considered as skyline services, since none of the three cases presented previously is fulfilled. Let's consider, for instance, the two services a and b. Although the service a offers better QoS values, its time span does not cover that of the service b. Hence, this latter can be better than the service a according to its time properties for a specific selection problem and thus, it should be preserved. Another example is that the service c can be pruned since it is dominated by the service a. Actually, this latter offers better QoS values and its time interval covers that of the service c.

Back to the electronic device production example we have introduced above, according to the service instances depicted in Figure 3.2, the service instance S_{631} dominates the

service instances S_{621} and S_{622} since it offers better quality values and has a larger time interval. Thus, these two latter instances can be removed from the search space.

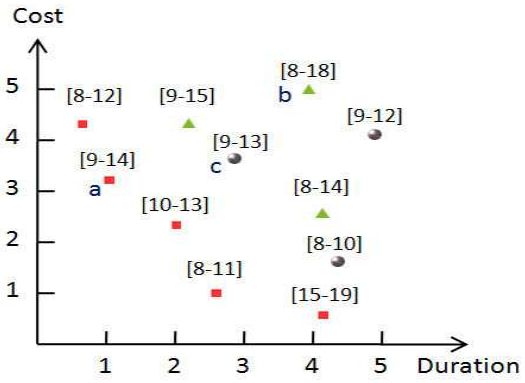

FIGURE 4.1: Skyline services based on QoS attributes and temporal properties

In what follows, we argue that the dominance-based pruning step does not discard the optimal solution if it exists. Before explaining the proposed Lemma, we give Theorem 4.1.

Theorem 4.1. *If a combination of services CS satisfies all constraints, a combination of services CS' derived from CS by substituting at least one service in CS by another better service (for at least one quality or temporal attribute and with the same or better values for other attributes), while preserving all other services, satisfies all constraints and has the same or better utility value under two conditions: (1) the QoS aggregation functions are monotone (i.e., higher (lower) values produce a higher (lower) overall aggregated value) and (2) all services are independent (i.e., the quality values of one service do not depend on the quality values of another service).*

We point out that in this work, there are no QoS dependencies between services (i.e., replacing one service in the composition by another service does not have influence on the QoS values of the other services in the composition) (See Section 3.2.3). Moreover, all aggregation functions are monotone. This latter condition has been proven in [54, 130] for several QoS categories.

Lemma 4.1. *The set of skyline services of all tasks allows computing the best solution (if it exists) under the monotone and the independent conditions.*

Theorem 4.1 and Lemma 4.1 are proven in Appendices A.1 and A.2, respectively.

Although selecting the skyline services for each business task is a promising solution to reduce the number of uninteresting services, the set of the selected services can still very large. For this, we propose to prune further inadequate services based on business constraints (Section 3.2). The constraint-based pruning process is discussed below.

4.3 Constraint-based Pruning Process

In this step, we propose to further eliminate inadequate services in order to reduce the search space. The main idea is to eliminate all services that will certainly violate the global constraints. Some methods are used to eliminate services that do not fulfill global constraints [17, 52]. For example, if the user requires that the global cost of the composite service should not exceed 10 cost units, any service instance with a cost greater or equal to 10 cost units can be discarded. However, further uninteresting services can be eliminated considering business constraints. In addition, temporal properties have to be considered in the pruning process when dealing with time-dependent QoS values. This is done by computing *local thresholds* for each business task while ensuring that these thresholds are relaxed as much as possible. In our work, we propose two search space reduction techniques: (1) *QoS constraint-based pruning* and (2) *temporal constraint-based pruning*. In the following, we detail how we measure thresholds using these two techniques.

4.3.1 Overview of the Constraint-based Pruning Process

The aim of the pruning process is to select the most adequate services (hereafter called *preselected services*) while guaranteeing that the optimal solution can still be found. This pruning process is based on a set of computed *local thresholds* so that if of one of these thresholds is not satisfied by a candidate service, then, the global constraints are not satisfied. In other words, if a service instance violates at least one local threshold, all possible combinations of services that include it will violate global user constraints, and thus, it can be pruned from the set of candidate services to narrow the search space.

Example 4.2. Let us consider a business process that has two sequential abstract tasks A_1 and A_2 with two global QoS constraints $Q(cost) = 5$ and $Q(dur) = 6$. Figure 4.2 shows a set of candidate services for the tasks A_1 and A_2. To compute local QoS thresholds, we first, identify minimum values of the QoS attributes for both tasks. In this example: $Q(A_1, cost)^{min} = 2$, $Q(A_2, cost)^{min} = 1$, $Q(A_1, dur)^{min} = 1$ and $Q(A_2, dur)^{min} = 2$. Based on these values and on the global QoS constraints, we compute local QoS thresholds for each quality attribute for both tasks (presented by the dashed lines in Figure 4.2). For instance, as the cost is an additive attribute and since the minimum cost value of the task A_2 is equal to 1 in the best case, all services with a cost greater than 4 ($Q(cost) - Q(A_2, cost)^{min}$) of the task A_1 can not be part of a feasible solution. In fact, if one of these services is selected, the global cost constraint will be violated even if the service with the minimum cost value of the second task is

selected. Thus, all services in the grey area will be eliminated. In the same way, we can define the remaining thresholds.

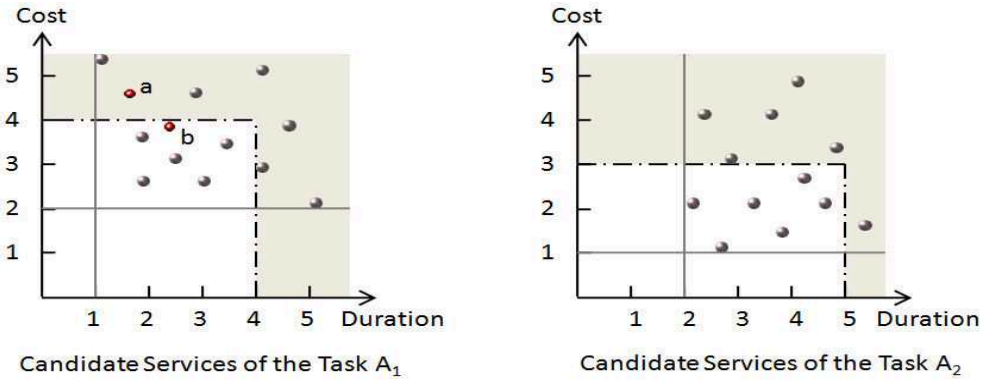

FIGURE 4.2: Preselected candidate services based on our pruning approach

In addition to local QoS thresholds, local temporal thresholds are computed when considering time-dependent QoS. In the following sections, we explain how we compute both QoS and temporal thresholds.

4.3.2 QoS Constraint-based Pruning

QoS-based pruning strategy aims to compute *local QoS thresholds* for individual tasks for each QoS attribute $q_y \in \mathcal{QS}$ that will serve as local upper bound constraints (since we consider only QoS attributes with decreasing value direction) so that the non-satisfaction of one of these constraints by a candidate service guarantees the non-satisfaction of the global constraints. In other words, if a service has at least one QoS value that does not satisfy a local threshold, all combinations of services that include it will violate global user constraints, and thus it is not worth considering and it can be pruned from the set of candidate services to narrow the search space.

Several existing approaches propose decomposition techniques to decompose global QoS constraints into local ones to reduce the number of candidate services per task [16, 72–74]. Nevertheless, these approaches rely on greedy reduction space techniques that can prune candidate services that are likely to be part of the optimal solution. In fact, it might be the case where a service does not meet a local constraint for one QoS attribute but offers good values for the other QoS attributes and thus, it can be selected in the optimal solution. To deal with this issue, in this section, we introduce a novel pruning approach based on QoS constraints. The proposed approach guarantees that only services that violate the global QoS constraints will be removed from the search space. Our goal is to define an approach that is generic enough to be applied for complex business processes

with different structural patterns and with the presence or absence of intra-task QoS constraints at the business level.

A local threshold $Q_{LT}(A_i, q_y)$ for the y^{th} attribute of the task A_i depends on the value of the required global global constraint $Q(q_y)$ and the minimum and maximum values of this QoS attribute that can be offered by services (i.e., $Q(A_i, q_y)^{min}$ and $Q(A_i, q_y)^{max}$). The main idea is to compute for each task its maximum allowed value (i.e., the worst case) considering the minimum quality values of all other tasks (i.e., their best cases) such that the global and intra-task QoS constraints are satisfied. Computing these thresholds needs to consider both the structure of the business process and the distinctive characteristics for each QoS attribute. For this, we propose to compute local thresholds for each business task based on a constraint optimization model. This model can be applied for generic business process structures that can contain sequential, parallel, choice and loop patterns with several categories of quality attributes (See Table 3.1). For simplicity, the proposed model assumes that business processes contain only atomic activities. If the structure of the business process contains sub-processes activities, we propose to compute local thresholds in a recursive manner. The local thresholds of each sub-process in the whole process will be considered as its global constraints. The proposed model is as follows:

$$\text{minimize} \sum_{A_i \in \mathcal{A}, i \neq b} Q(A_i, q_y) - Q(A_b, q_y) \tag{4.3}$$

$$Agg_{A_i \in \mathcal{A}}(Q(A_i, q_y)) \leq Q(q_y), \forall A_i \in \mathcal{A} \tag{4.4}$$

$$\sum_{A_i \in C_l} p_{li} = 1, \forall C_l \in \mathcal{SC} \tag{4.5}$$

$$p_{li} \in \{0, 1\}, \forall A_i \in \mathcal{C} \tag{4.6}$$

$$A_b \in \mathcal{C} \Rightarrow p_{lb} = 1 \tag{4.7}$$

$$Q(A_i, q_y) \leq V, \forall qc_i(MAX, V, q_y) \in \mathcal{QC} \tag{4.8}$$

$$V \leq Q(A_i, q_y), \forall qc_i(MIN, V, q_y) \in \mathcal{QC} \tag{4.9}$$

$$Q(A_i, q_y) \in [Q(A_i, q_y)^{min}, Q(A_i, q_y)^{max}], \forall A_i \in \mathcal{A}, i \neq b \tag{4.10}$$

$$Q(A_b, q_y) \in [0, Q(A_b, q_y)^{max}] \tag{4.11}$$

Constraint (4.3) presents the objective function of the proposed model. This function allows computing the maximum quality value of each activity A_b for the attribute $q_y \in \mathcal{QS}$ while minimizing the quality values of all the other tasks $A_i, i \neq b$. The maximum quality value of each activity A_b will be considered as its local threshold (i.e.,

$Q_{LT}(A_b, q_y) = Q(A_b, q_y)$, $\forall A_b \in \mathcal{A}$). To guarantee that the global QoS constraints are satisfied, we propose Constraint (4.4). The value of a QoS attribute of the composite service is computed using the aggregation function presented in Section 3.4.1. Unlike existing approaches, which consider only sequential flow, our approach allows computing local thresholds considering several composition structures. To deal with choice structures, we add constraints (4.5) and (4.6). Usually, the value of a quality attribute in a choice structure is measured by the sum of the different execution paths multiplied by their probabilities. However, since we consider the best case of all tasks A_i and the worst case of the task A_b, we need to consider only one path for each branch independently on the probabilities of the different paths. For this reason, we suppose that p_{li} can be equal to 0 or 1 for each task in a choice structure and that only one path can be considered for each choice pattern. Constraint (4.7) indicates that if the activity for which we compute the local threshold (i.e., A_b) is in a choice structure, it should be considered as the path that will be executed and thus, its probability is equal to 1. Intra-task QoS constraints have to be guaranteed also when computing thresholds. For this purpose, we add Constraints (4.8) and (4.9). For instance, given the example in Section 1.3, if we do not consider intra-task QoS constraints, the local threshold of the task A_5 is $Q_{LT}(A_5, cost) = 68 - (8 + 10 + 12 + 12 + 6) = 20$. However, if we consider the constraint QC_5, that states that the cost of the task A_5 must not exceed 14 units, this threshold will be equal to 14 rather than 20. Finally, Constraint (4.10) indicates that the quality value of each task A_i with $i \neq b$ belongs to the interval $[Q(A_i, q_y)^{min}, Q(A_i, q_y)^{max}]$ and Constraint (4.11) defines the domain of the quality value of the task A_b. Here, we suppose that the domain of the quality value of the task A_b is $[0, Q(A_b, q_y)^{max}]$ in order to compute its minimum quality value. In the case where the computed local QoS thresholds for the task A_b is less than the minimum quality value (i.e., $Q_{LT}(A_b, q_y) < Q(A_b, q_y)^{min}$, we conclude that there is no solution to the selection problem and improvement techniques have to be taken (Section 4.4).

For instance, given the motivating example we have introduced in Section 1.3, with $Q(cost) = 68$ and by applying the proposed model, we obtain the following thresholds for each task: $Q_{LT}(A_1, cost) = 20$, $Q_{LT}(A_2, cost) = 17$, $Q_{LT}(A_3, cost) = 23$, $Q_{LT}(A_4, cost) = 22$, $Q_{LT}(A_5, cost) = 14$ and $Q_{LT}(A_6, cost) = 10$. After computing the cost thresholds for each task, the number of services instances is restricted. For example, all service instances that have a cost greater than 20 cost units for the first task will be eliminated (e.g., the service instance S_{131} in Figure 3.2).

4.3.3 Temporal Constraint-based Pruning

Although QoS constraint-based pruning keeps for each task only candidate services that are likely to be a member of the optimal solution, some uninteresting services still need to be removed when taking into consideration time-dependent QoS attributes and temporal constraints. Considering temporal constraints is inevitable to satisfy business process requirements. In the following, we detail how we consider all these constraints to further reduce the number of candidate services. Particularly, two temporal properties have to be considered: (1) the *execution duration* of each activity regarding the required global duration of the business process, and (2) the *time spans* (i.e., start and finish time) of each activity with respect to the required deadline.

4.3.3.1 Execution Duration

The execution duration of each task can be computed using the model specified in Section 4.3.2 since the execution duration attribute can be considered as a Max-Operator attribute. Nevertheless, this model is not sufficient to effectively measure local duration thresholds in presence of temporal dependencies.

Example 4.3. To illustrate the underlying issue, let's take the electronic device production case study we have introduced in Section 1.3. If we apply the QoS model presented above (Section 4.3.2) and the corresponding max-operator aggregation function (equation (3.4)), the local duration threshold of the task A_2 is equal to 4. However, if the task A_2 is executed in 4 time units, the temporal dependency $TD_{1,4}$, that states that the manufacturing of peripheral parts (i.e., the activity A_4) starts no later than 3 time units after the writing of the technical reports (i.e., the activity A_1), will be violated. Thus, services that offer an execution duration equal or greater than 4 time units for the task A_2 should be discarded. Another example is that based on the proposed QoS model, the local threshold of the task A_6, that fulfills tests of the final product, is equal to 5. Nevertheless, services that have an execution duration value equals to 5 time units for the 6^{th} task can not participate in the selection process if we consider the temporal dependency $TD_{5,6}$ that imposes that the test activity A_6 has to start no earlier than one time unit and no later that 2 time units after the assembly of all parts. In fact, even if all tasks will be executed at their minimum allowed execution duration values and considering the minimum values of the temporal dependencies, the global constraint (i.e., the duration$\leq$ 13 time units) will be violated. Hence, considering a local threshold that is equal to 4 rather than 5 time units for the task A_6 will lead to further elimination of uninteresting candidate services. In addition, intra-task temporal constraints can also affect the execution duration of the business process. Let's consider for instance the

temporal constraint TC_2, that states that the design of the product model (A_2) has to finish no later than 3 p.m. Given this constraint, and considering the service instances of the first and the second task, the duration of the task A_2 can not be greater than 3 time units. In fact, in the best case, the execution of the first task will finish at 12 a.m. given the candidate service instances presented in Figure 3.2. Thus, to guarantee that the temporal constraint TC_2 is fulfilled, the execution duration of the task A_2 should be less than 3 time units.

It is clear that identifying local thresholds is more complex when handling business processes with several structural and temporal constraints. This is explained by the fact that some temporal dependencies may overlap. Additionally, temporal constraints may have different types and thus, they should be resolved differently. To deal with these constraints, we rely on a constraint optimization model that allows computing local maximum duration for each business task while handling complex structural dependencies as well as intra and inter-task temporal constraints. This model is applied for each task $A_b \in \mathcal{A}$ to search for its maximum duration while minimizing the duration of all other tasks. Therefore, the proposed model can be expressed as follows:

$$\text{minimize} \sum_{A_i \in \mathcal{A}, i \neq b} Q(A_i, dur) - Q(A_b, dur) \tag{4.12}$$

$$ft_n - st_1 \leq Q(dur) \tag{4.13}$$

$$ft_n \leq deadline \tag{4.14}$$

$$ft_i = st_i + Q(A_i, dur), \forall A_i \notin \mathcal{L} \tag{4.15}$$

$$ft_i = st_i + \alpha_i * Q(A_i, dur), \forall A_i \in \mathcal{L} \tag{4.16}$$

$$\sum_{A_i \in C_l} p_{li} = 1, \forall C_l \in \mathcal{SC} \tag{4.17}$$

$$p_{li} \in \{0, 1\}, \forall A_i \in \mathcal{C} \tag{4.18}$$

$$A_b \in \mathcal{C} \Rightarrow p_{lb} = 1 \tag{4.19}$$

$$ft_i \leq st_v, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v), A_i \notin \mathcal{C} \tag{4.20}$$

$$\sum_{A_i \in C_l} p_{li} * ft_i \leq st_v, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v), A_i \in C_l \tag{4.21}$$

$$Q(A_i, dur) \leq V, \forall qc_i(MAX, V, dur) \in \mathcal{QC} \tag{4.22}$$

$$V \leq Q(A_i, dur), \forall qc_i(MIN, V, dur) \in \mathcal{QC} \tag{4.23}$$

$$st_i \leq T, \forall tc_i(SNLT, T) \in \mathcal{TC} \tag{4.24}$$

$$ft_i \leq T, \forall tc_i(FNLT, T) \in \mathcal{TC} \tag{4.25}$$

$$ft_i + D_{iv}^{min} \leq st_v, \forall td_{i,v}(FS, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD} \tag{4.26}$$

$$st_v \leq ft_i + D_{iv}^{max}, \forall td_{i,v}(FS, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD} \tag{4.27}$$

$$Q(A_i, dur) \in [Q(A_i, dur)^{min}, Q(A_i, dur)^{max}], \forall A_i \in \mathcal{A}, i \neq b \tag{4.28}$$

$$Q(A_b, dur) \in [0, Q(A_b, dur)^{max}] \tag{4.29}$$

$$st_i, ft_i \in [t_{A_i}^{min}, t_{A_i}^{max}], \forall A_i \in \mathcal{A} \tag{4.30}$$

The objective function of the proposed model (4.12) allows computing for each task $A_b \in \mathcal{A}$ its maximum execution duration while minimizing the durations of all other tasks. Then, the local threshold of the task A_b is $Q_{LT}(A_b, dur) = Q(A_b, dur)$. The computation of local thresholds must ensure that structural and temporal constraints are satisfied. To guarantee that the global constraint of the duration attribute is satisfied, we propose constraint (4.13). Constraint (4.14) guarantees that the deadline is respected. Constraints (4.15) and (4.16) specify the relation between the start and the finish times of each task considering both non-loop and loop tasks, respectively. To guarantee that all loop structures are verified, here, we consider that all preselected services of loop tasks must satisfy the maximum number of iterations. Similarly to the constraints from (4.5) to (4.7) specified in the Section 4.3.2, Constraints from (4.17) to (4.19) handle choice structures. To deal with *structural constraints*, we add Constraints (4.20) and (4.21). Constraints (4.22) and (4.23) allow handling intra-task QoS constraints related to the execution duration attribute. To deal with intra-task temporal constraints, we propose constraints (4.24) and (4.25). For simplicity, we only consider the temporal constraints (SNLT and FNLT). For example, the constraint (4.24) ensures that for each constraint of the form *Start No Earlier Than* T, the earliest start time of the corresponding task is less than or equal to the time point T. To deal with inter-task temporal constraints (i.e., temporal dependencies), we propose Constraints (4.26) and (4.27). For simplicity, only finish-to-start temporal dependencies are considered since other dependencies can be defined in the same manner. For instance, the constraint (4.26) ensures that once the finish time of an activity A_i arises, the activity A_v can begin at the earliest time after the minimum duration value D_{iv}^{min} of the inter-task temporal constraint $td_{i,v}$. Finally, Constraints from (4.28) to (4.30) indicate the domains of the duration and the start and the finish time of each business task.

Based on this model, local thresholds for the execution duration attribute for each business task can be defined while guaranteeing the satisfaction of all structural, QoS and temporal constraints. For instance, given the example in Section 1.3, the maximum execution duration values of all tasks are as follows: 4, 3, 5, 8, 3 and 4.

4.3.3.2 Time Intervals

The selection of a solution when dealing with both time-dependent QoS and temporal constraints of business tasks needs the specification of the start and finish time of each service. Nevertheless, the start time of each service is affected by the start time of its predecessors. Thus, selecting a wrong start time for one service may lead to several wrong choices for the start time of its successor services, which decreases the performance of the selection algorithm.

Example 4.4. Let's take for instance our motivating scenario presented in Section 1.3 and the services instances in Figure 3.2 and suppose that the service instance S_{121} is selected since it offers the best cost (2 cost units) and the minimum execution duration (8 time units) within the time span [17,21]. Therefore, according to this time interval, only the service instance S_{232} whose time span is [14,21] can be selected for the task A_2. This solution does not fulfill the required user constraints. In fact, in the best case, the service S_{232} will finish the execution at 20 time units. Thus, even though the services that have the minimum execution duration values are selected for the remaining tasks, the overall deadline will be violated.

To avoid possible unnecessary combinations, we propose to compute the largest time span for each task based on the deadline required by the user while satisfying structural and temporal constraints. The boundaries of these time intervals (i.e., the earliest start and the latest finish time) will be considered as *local temporal thresholds* for each task such that all service instances whose time intervals do not belong to the computed time spans will be pruned. To identify the largest time spans of each business task when dealing with complex business processes, we propose a constraint optimization model that computes the earliest start time (est) and the latest finish time (lft) for each business task so that all structural and temporal constraints are fulfilled and the deadline of the entire process is respected. The proposed model is executed for each task $A_b \in \mathcal{A}$ to compute its largest time interval.

$$\text{maximize} \quad lft_b - est_b \tag{4.31}$$

$$lft_i = est_i + Q(A_i, dur), \forall A_i \notin \mathcal{L} \tag{4.32}$$

$$lft_i = est_i + \alpha_i * Q(A_i, dur), \forall A_i \in \mathcal{L} \tag{4.33}$$

$$lft_n \leq deadline \tag{4.34}$$

$$est_i + Q(A_i, dur)^{min} \leq est_v, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v), A_i \notin \mathcal{C} \tag{4.35}$$

$$lft_i + Q(A_v, dur)^{min} \leq lft_v, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v), A_i \notin \mathcal{C} \tag{4.36}$$

$$min_{A_i \in C_l}\{est_i + Q(A_i, dur)^{min}\} \leq est_v, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v), A_i \in C_l \quad (4.37)$$

$$min_{A_i \in C_l}\{lft_i + Q(A_v, dur)^{min}\} \leq lft_v, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v), A_i \in C_l \quad (4.38)$$

$$est_i \leq T, \forall tc_i(SNLT, T) \in \mathcal{TC} \quad (4.39)$$

$$lft_i \leq T, \forall tc_i(FNLT, T) \in \mathcal{TC} \quad (4.40)$$

$$est_i + Q(A_i, dur)^{min} + D_{iv}^{min} \leq est_v, \forall td_{i,v}(FS, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD} \quad (4.41)$$

$$est_v \leq est_i + Q(A_i, dur)^{min} + D_{iv}^{max}, \forall td_{i,v}(FS, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD} \quad (4.42)$$

$$lft_i + Q(A_v, dur)^{min} + D_{iv}^{min} \leq lft_v, \forall td_{i,v}(FS, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD} \quad (4.43)$$

$$lft_v \leq lft_i + Q(A_v, dur)^{min} + D_{iv}^{max}, \forall td_{i,v}(FS, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD} \quad (4.44)$$

$$Q(A_i, dur) \in [Q(A_i, dur)^{min}, deadline], \forall A_i \in \mathcal{A} \quad (4.45)$$

$$est_i, lft_i \in [t_{A_i}^{min}, t_{A_i}^{max}], \forall A_i \in \mathcal{A} \quad (4.46)$$

Constraint (4.31) allows computing the largest time intervals for each business task A_b (i.e., maximize the distance between the earliest start time *est* and the latest finish time *lft* for each task). Constraints (4.32) and (4.33) indicate the relation between the start and the finish time of each task. To guarantee that the deadline is not violated, we add Constraint (4.34). To deal with structural dependencies, we propose Constraints (4.35) and (4.36), which guarantee that the earliest start time of each task A_v occurs after the earliest start time and the minimum duration of each of its predecessor tasks. In addition, the latest finish time of each task A_v has to be greater than or equal to the sum of its minimum execution duration and the latest finish time of all its predecessor tasks. Since we consider the largest time interval for each task, we suppose that the start and the finish time of each task should be greater than the minimum start and finish time of its predecessors if these latter belong to a choice structure (4.37) and (4.38). Furthermore, it is vital to check if temporal constraints are satisfied when computing the largest time interval of each task. To deal with intra and inter-task temporal constraints, we propose Constraints from (4.39) to (4.44), respectively. As previously, for simplicity, we consider only a limited set of temporal constraints. The duration of each task A_i should be within the interval $[Q(A_i, dur)^{min}, deadline]$ since we search for the largest time interval of each task (Constraint (4.45)). Finally, Constraint (4.46) shows the domain of the start and the finish time of each task $A_i \in \mathcal{A}$.

A solution of the optimization problem is then a set of the largest possible time intervals of all tasks. For instance, given the example presented in Section 1.3, the largest time slots of all tasks when considering all imposed constraints and with a deadline equal to 22 are, respectively, [8,14], [10,15], [11,19], [11,19], [15,20] and [17,22]. Considering these

intervals, some service instances have to be pruned (e.g., S_{121}, S_{132}, S_{332} and S_{421}) or some restrictions have to be performed to their intervals (e.g., S_{431}). This latter case is detailed in the next section.

4.3.4 Constraint-based Pruning Algorithm

After explaining how to compute the local QoS and temporal thresholds of each task, in this section, we present the constraint-based pruning algorithm. The different steps of the constraint-based pruning process are given in Algorithm 1.

Algorithm 1 Identifying Preselected Services for a Task A_i

1: **Input:** The set of non-dominated services $\mathcal{S}_{Skyi}$
2: **Output:** The set of preselected services $\mathcal{S}_{Presi}$
3: $\mathcal{S}_{Presi} = \emptyset$, $Q_{LT} = \emptyset$, $T_{LT} = \emptyset$
4: $getMinMaxValues(\mathcal{S}_i)$
5: **for** each $q_y \in \mathcal{QS}$ **do**
6: $\quad Q_{LT}(A_i, q_y) \leftarrow computeLocalQoSThresholds\ (A_i, q_y)$
7: $\{est_i, lft_i\} \leftarrow computeLocalTemporalThresholds\ (A_i)$
8: **for** each $S_{ijk} \in \mathcal{S}_{Skyi}$ **do**
9: $\quad$ **for** each $q_y \in \mathcal{QS}$ **do**
10: $\quad\quad$ **if** $Q(S_{ijk}, q_y) > Q_{LT}(A_i, q_y)$ **then**
11: $\quad\quad\quad \mathcal{S}_i = \mathcal{S}_i \setminus \{S_{ijk}\}$
12: $\quad\quad\quad$ **break**
13: $\quad$ **if** $[t^{min}_{S_{ijk}}, t^{max}_{S_{ijk}}] \cap [est_i, lft_i] = \emptyset$ **then**
14: $\quad\quad \mathcal{S}_i = \mathcal{S}_i \setminus \{S_{ijk}\}$
15: $\quad\quad$ **break**
16: $\quad$ **else** $\{[t^{min}_{S_{ijk}}, t^{max}_{S_{ijk}}] \cap [est_i, lft_i] = [X, Y]\}$
17: $\quad\quad$ **if** $Y - X < Q(S_{ijk}, dur)$ **then**
18: $\quad\quad\quad \mathcal{S}_i = \mathcal{S}_i \setminus \{S_{ijk}\}$
19: $\quad\quad\quad$ **break**
20: $\quad\quad$ **else**
21: $\quad\quad\quad$ **if** $t^{min}_{S_{ijk}} < X$ **then**
22: $\quad\quad\quad\quad t^{min}_{S_{ijk}} = X$
23: $\quad\quad\quad$ **if** $t^{max}_{S_{ijk}} > Y$ **then**
24: $\quad\quad\quad\quad t^{max}_{S_{ijk}} = Y$
25: $\quad\quad\quad \mathcal{S}_{Presi} = \mathcal{S}_{Presi} \cup \{S_{ijk}\}$
26: **if** $\mathcal{S}_i = \emptyset$ **then**
27: $\quad$ There is no solution

For each task $A_i \in \mathcal{A}$, the pruning algorithm takes as input the set of non-dominated candidate services $\mathcal{S}_{Skyi}$ and returns the set of preselected services $\mathcal{S}_{Presi}$ that are likely to be candidate of the optimal solution (i.e., they do not violate any of the local thresholds of their corresponding tasks). The algorithm begins by computing minimum and maximum values of QoS attributes and time intervals after the dominance-based pruning (line 4). Based on these values, local QoS and temporal thresholds are computed

using constraint optimization models presented in Sections 4.3.2 and 4.3.3 (line 5 to 7). The second step is to prune services based on QoS thresholds (line 8 to 12) and then based on time spans (line 13 to 24). If a local threshold is violated, it is not worth to check the fulfilment of other thresholds and the service should be removed from the set of available services. If all QoS thresholds are verified, we compare the time span of each timed service instance to the interval of its corresponding task. If the intersection between these two intervals is empty or it does not cover the duration of the service instance, this instance should be eliminated (line 13 to 19). Otherwise, the time span of the service instance should be restricted to the span of its task (line 20 to 24) and the service is considered as pertinent and it will be added to the set of preselected services (line 25). Finally, if at least one task does not have any candidate service, the selection problem has no feasible solutions (lines 26 and 27).

Lemma 4.2. *Given the optimal combination of services* $CS^* = \{s_1, ..., s_i, ..., s_n\}$ *(i.e., the one that satisfies all the required constraints and optimizes the overall utility according to equation (3.5)), each service* $s_i \in CS^*$ *belongs to the set of pre-selected services after the constraint-based pruning process.*

The proof of Lemma 4.2 is provided in Appendix A.3.

4.3.5 Iterative Pruning Process

After applying the pruning process based on local thresholds, minimum and maximum values of the QoS attributes and time intervals of each service class may change. Therefore, since our constraint-based pruning techniques are based on the minimum and maximum values for all attributes, local thresholds can be recomputed iteratively using the new values for each service class in order to prune further uninteresting services.

Example 4.5. Let's take the example shown previously in Figure 4.2 in Section 4.3.1. For instance, after the first iteration of the pruning process, the minimum values of cost and execution duration attributes of the task A_1 change as follows: $Q(A_1, cost)^{min} = 2.5$ and $Q(A_1, dur)^{min} = 2$ (the red lines in the left side of Figure 4.3). Therefore, local thresholds for the task A_2, which are initially equal to 3 for the cost attribute and 5 for the duration attribute, may change. Figure 4.3 shows the new values of local thresholds after the second iteration of the constraint-based pruning algorithm (the red dashed lines in the right side of Figure 4.3). Consequently, candidate services of the task A_2 with a cost greater than 2.5 cost units or an execution duration greater than 4 time units can not participate in the selection process.

After applying our pruning process based on the aforementioned thresholds, further services can be pruned and thus, the number of candidate services is reduced. The iterative

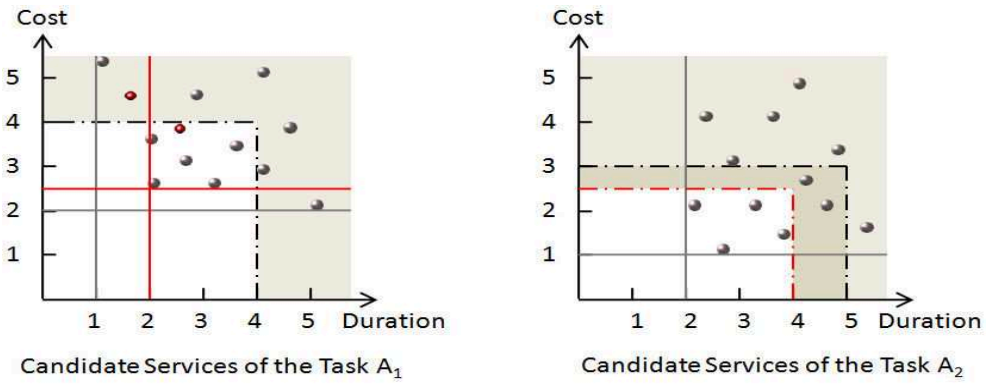

FIGURE 4.3: Preselected candidate services after the second iteration of the pruning process

pruning is more useful in anti-correlated distributions (See Section 3.3.2). As can be seen in Figure 4.3, most of the services of the task A_1 follow an anti-correlated distribution in contrast to the services of the task A_2, which follow a correlated distribution. Thus, the first iteration of the pruning process has a tremendous impact on the minimum and maximum values of the services of A_1 and thus, on the local thresholds and the number of preselected services of the task A_2. The iterative pruning is then, more relevant when the candidate service instances of at least one service class have an anti-correlated distribution. This is mainly due to the fact that in anti-correlated distributions, minimum and maximum values have more chances to be modified after the pruning process and thus, there is more chance that local thresholds will be changed. Consequently, we apply the iterative pruning in independent and anti-correlated distributions until there is no change in the local thresholds.

Table 4.1 presents local thresholds of the motivating example in Section 1.3 after the second iteration of the pruning algorithm.

TABLE 4.1: Local thresholds after the second iteration of the pruning algorithm.

	A_1	A_2	A_3	A_4	A_5	A_6
Cost	12	17	13	20	14	7
Execution Duration	6	3	6	6	3	3
Time Span	[8,14]	[12,15]	[13,19]	[13,19]	[17,20]	[19,22]

Based on these new thresholds, further service instances (Figure 3.2) can be pruned such as the services S_{411} and S_{231}.

4.4 Overview of the Improvement Process

In case of failure (i.e., there is no feasible solution), negotiation and adaptation based strategies can be used to find a solution to the selection problem [67, 111]. This is usually done once the selection process is performed. Nevertheless, this is not efficient and introduces a significant overhead. Moreover, in most cases, it is not obvious to identify the source of failures so that negotiation and adaptation actions may require several changes and improvements in order to find a solution.

In contrast to existing approaches, besides reducing the search space by discarding inadequate services, our constraint-based pruning approach enables identifying the source of failure at earlier stages. In addition, it provides indications on the possible improvements required to find a feasible combination of services that fulfills all constraints. Our goal is to characterize the minimum required set of constraints and QoS values relaxations that is sufficient to enhance the possibility of finding a feasible solution. Service providers and users can be guided to identify which quality values or constraints should be relaxed. Moreover, they can be guided on how they can change the values in order to address the problem. The owner of the business process can also be asked to modify the structure of the business process or to relax some QoS and temporal constraints. Such information can be very valuable in the negotiation and adaptation step as it enables the specification of the most suitable improvements.

After applying our pruning algorithm, three cases can be distinguished to conclude that the selection problem is infeasible:

- There is at least one local QoS threshold that is less than the minimum quality value of its corresponding task. In this case, the local QoS constraint of the corresponding attribute is not satisfied.
- Temporal thresholds can not be computed for at least one task (i.e., there is no solution for the constraint optimization model from (4.31) to (4.46)).
- There is at least one task that has no preselected services. This indicates that even though local thresholds can be computed, there is no candidate service that satisfies all thresholds.

Example 4.6. Let's take the example depicted in Figure 4.4. As in the previous examples, we consider candidate services for two business tasks in sequence A_1 and A_2. In this example, the minimum cost values of the service classes of the tasks A_1 and A_2 are equal to 2 and 3 cost units, respectively. Suppose that the user requires that the global cost should be less than 4 cost units (i.e., $Q(cost) = 4$). In this case, the local

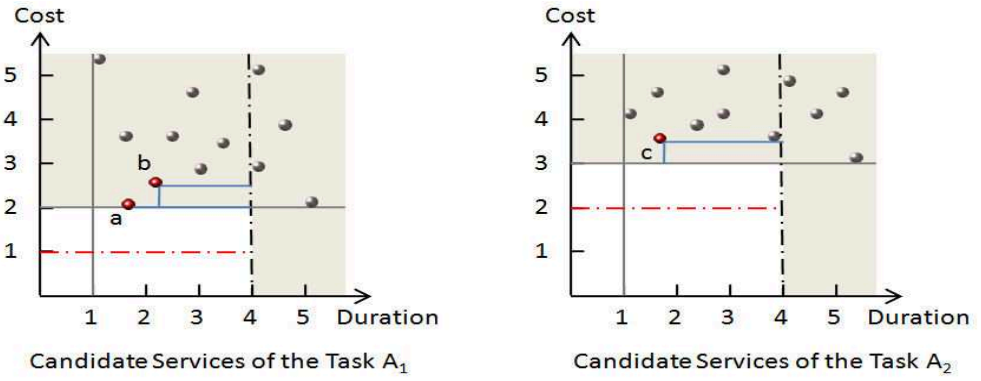

FIGURE 4.4: Example of a selection problem where there is no feasible solution

cost thresholds of the tasks A_1 and A_2 (presented by the dashed red lines in Figure 4.4) are equal to 1 and 2, respectively and thus, they are less than the minimum cost values. To this end, we can conclude that the global constraint of the cost attribute can not be fulfilled by the candidate services and thus, improvement actions are required to adapt information and enlarge the selection domain.

In this example, if constraint-pruning techniques are not used, a failure can be observed only after applying the selection process. To try to find a solution to the selection problem, we propose a relaxation based strategy while guaranteeing that only a minimum set of relaxations and improvements is required. This can include the notification of constraints that are too strict, the values of QoS and temporal properties that have to be modified and so on. Improvement actions can be handled by users to relax their constraints, by service providers to change their offers or by business process designers to adapt constraints at the business level (i.e., structural, QoS and temporal constraints). In this thesis, we only present possible improvements required by users and service providers and we do not consider the adaptation of structural, QoS and temporal constraints at the business level.

4.4.1 Improving Global Constraints

If the selection problem in infeasible, changes in the global constraints can be suggested, so that, a feasible solution can be found. We note that usually there is a tradeoff among quality attributes. Indeed, in most cases, a positive modification of one constraint implies a negative modification of another constraint (e.g., the user may agree to pay more while requiring a better response time in turn). Thus, a successful improvement is reached when (1) a minimal set of modifications is required and (2) the improvement of the constraint of one attribute does not effect the satisfaction of the global constraints of the other attributes. This can be ensured by specifying the minimum allowed values for

all quality attributes even for those whose global constraints are satisfied. This allows to get an idea on the minimum values that must not to be exceeded when updating the global constraints.

Algorithm 2 Improvement of Global Constraints

1: **Input:** The set of initial global constraints
2: **Output:** A list IU of the possible improvements
3: **for** each $q_y \in \mathcal{QS}$ **do**
4: $\quad Q(q_y)' \leftarrow$ *computeMinGlobalQoS()*
5: $\quad IU[y] \leftarrow Q(q_y) \geq Q'(q_y)$
6: $deadline' \leftarrow$ *computeMinAllowedDeadline()*
7: $IU[y+1] \leftarrow deadline \geq deadline'$

To do so, we propose the Algorithm 2 to assist users in relaxing global QoS constraints and the overall deadline so that local QoS and temporal thresholds can be computed. The function *computeMinGlobalQoS()* (line 4) allows computing the minimum possible global QoS value for each quality attribute (i.e., $Q'(q_y)$) while considering candidate services of each task.

If the quality attribute is not the execution duration, the function *computeMinGlobalQoS()* derives the model presented in Section 4.3.2 from (4.3) to (4.11) while replacing (4.3) and (4.4) by the following two constraints, respectively:

$$\text{minimize} \quad Q'(q_y) \tag{4.47}$$

$$Agg_{A_i \in \mathcal{A}}(Q(A_i, q_y)) \leq Q'(q_y), \forall A_i \in \mathcal{A} \tag{4.48}$$

Moreover, Constraints (4.10) and (4.11) are replaced by the following constraint:

$$Q(A_i, q_y) \in [Q(A_i, q_y)^{min}, Q(A_i, q_y)^{max}], \forall A_i \in \mathcal{A} \tag{4.49}$$

If the quality attribute is the execution duration, the model presented in Section 4.3.3.1 from (4.12) to (4.30) will be used while replacing constraints (4.12) and (4.13) by the following constraints, respectively:

$$\text{minimize} \quad Q'(dur) \tag{4.50}$$

$$ft_n - st_1 \leq Q'(dur) \tag{4.51}$$

Constraints (4.28) and (4.29) are replaced by the following constraint:

$$Q(A_i, dur) \in [Q(A_i, dur)^{min}, Q(A_i, dur)^{max}], \forall A_i \in \mathcal{A} \tag{4.52}$$

Hence, to improve the selection problem, the user is asked to propose a global QoS value greater than or equal to the minimum global QoS value computed for each quality attribute (line 5 in Algorithm 2).

The function *computeMinAllowedDeadline()* defines the minimum allowed deadline (i.e., $deadline'$) (line 6). This latter is computed using the constraint optimization model proposed in Section 4.3.3.2 from (4.31) to (4.46) while replacing constraints (4.31) and (4.34) by the following constraints, respectively:

$$\text{minimize} \quad deadline' \tag{4.53}$$

$$lft_n \leq deadline' \tag{4.54}$$

Again, to enhance the selection problem, the deadline required by the user must be greater than or equal to the computed minimum required deadline (line 7).

4.4.2 Improving Service Offers

Providers can also be asked to change the values of their offers in order to find an agreement with the user. Based on the computed local thresholds, we can specify the minimum values of quality attributes (since we only consider quality attributes with decreasing value direction as stated previously) that can be negotiated with service providers so that global user constraints can be fulfilled. Given the example shown in Figure 4.4, the local cost thresholds of the tasks A_1 and A_2 are equal to 1 and 2, respectively. Thus, if at least one candidate service for the first task has a cost equal to or less than 1 cost unit, there is a chance to find a solution to the selection problem. The probability of finding a solution increases as well as the number of services that offer the required quality value increases. Algorithm 3 enables to identify the improvements that can be done by service providers. As explained previously, in order to guide the providers, we specify the required thresholds for all quality attributes (lines 3 to 5) to guarantee that the improvement of one or more attributes does not affect the satisfaction of the global constraints of the other attributes.

Algorithm 3 Improvement of Service Offers for a Task A_i

1: **Input:** The set of service instances and constraints
2: **Output:** A list IP of the possible improvements
3: **for** each $S_{ijk} \in \mathcal{S}_i$ **do**
4: **for** each $q_y \in \mathcal{QS}$ **do**
5: $IP[y] \leftarrow Q(S_{ijk}, q_y) \leq Q_{LT}(A_i, q_y)$
6: sortServicesBy($score$)

In addition, the chance to find a solution increases when the quality values of candidate services are closer to the required local thresholds of the unsatisfied constraints and farther to the local thresholds of the satisfied ones. For instance, in our example, it is more likely to find an agreement with those service providers that have the nearest values to the cost threshold values and the farthest values from the local threshold values for the execution duration (e.g., services a, b and c in Figure 4.4). In this way, providers of the identified services could have more flexibility to change their values without affecting the satisfaction of the global constrains. This is due to the fact that improving one attribute will usually lead to deteriorating other attributes. Hence, we rank services according to computed scores regarding local thresholds (line 6 in Algorithm 3). The providers with the maximum score values will be considered first in the negotiation process. In this step, all candidate services will be considered. The score of each service $S_{ijk} \in \mathcal{S}_i$ is computed using the equation (4.55):

$$score(S_{ijk}) = w_t(S_{ijk}) * (d^S(S_{ijk}) - d^{NS}(S_{ijk})) \tag{4.55}$$

with $w_t(S_{ijk})$ denotes the weight of the satisfaction of local temporal thresholds by the service (S_{ijk}). It is computed as follows:

$$w_t(S_{ijk}) = \begin{cases} 1 & \text{if} \quad Y - X \geq Q(S_{ijk}, dur) \\ 0.75 & \text{if} \quad Y - X < Q(S_{ijk}, dur) \\ 0.5 & \text{if} \quad [X, Y] = \emptyset \end{cases} \tag{4.56}$$

with $[X, Y] = [t^{min}_{S_{ijk}}, t^{max}_{S_{ijk}}] \cap [est_i, lft_i]$. We note that in the case where local temporal thresholds can not be computed (i.e., est_i and lft_i in the model from (4.31) to (4.46)), the weight of the satisfaction of local temporal thresholds for each service S_{ijk} is equal to 1 (i.e., $w_t(S_{ijk})$=1).

The values of $d^S(S_{ijk})$ and $d^{NS}(S_{ijk})$ denote the distance of the quality values for the service S_{ijk} with respect to the QoS thresholds for the satisfied and unsatisfied constraints, respectively. Here, we denote by $\mathcal{SQ}$ and $\mathcal{NSQ}$ the set of QoS attributes of the satisfied and unsatisfied constraints, respectively, with $\mathcal{QS} = \mathcal{SQ} \cup \mathcal{NSQ}$. We use the Euclidean distance to measure the distance between the values of a candidate service and the values of local thresholds. To do so, we propose the following two equations.

$$d^S(S_{ijk}) = \sqrt{\sum_{q_y \in \mathcal{SQ}} (Q(S_{ijk}, q_y)' - Q_{LT}(A_i, q_y)')^2} \tag{4.57}$$

$$d^{NS}(S_{ijk}) = \sqrt{\sum_{q_y \in \mathcal{NSQ}} (Q(S_{ijk}, q_y)' - Q_{LT}(A_i, q_y)')^2} \tag{4.58}$$

Where $Q(S_{ijk}, q_y)'$ and $Q_{LT}(A_i, q_y)'$ denote the normalized quality value of the service S_{ijk} and the normalized local threshold of the task A_i for the attribute q_y, respectively.

Indeed, since the domains of the values of the different quality attributes and local thresholds can be different, QoS values of candidate services and thresholds are normalized by transforming them into values between 0 and 1. The new value of each candidate service $S_{ijk} \in \mathcal{S}_i$ for each quality attribute q_y with $1 \leq y \leq m$ is computed as follows:

$$Q(S_{ijk}, q_y)' = \begin{cases} \frac{Q(A_i,q_y)^{max} - Q(S_{ijk},q_y)}{Q(A_i,q_y)^{max} - M} & \text{if} \quad Q(A_i, q_y)^{max} - M \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{4.59}$$

With $M = min(Q(A_i, q_y)^{min}, Q_{LT}(A_i, q_y))$. By the same manner, the scaling is applied also to the values of local QoS thresholds.

At this stage, several negotiation strategies can be adopted to find an agreement with users and providers. Here, we do not provide any details for the negotiation and adaptation techniques. Approaches proposed in the literature [67, 131] can be used. The negotiation terminates when a sufficient number of improvements is reached. The pruning process can still be applied along with the improvement phase until all local thresholds can be measured and a sufficient number of preselected services is ensured for each business task.

4.5 Conclusion

In this chapter, we have presented a dominance and constraints based pruning process. The proposed approach aims to reduce the number of candidate services that have to be considered in the selection process while guaranteeing that the optimal solution still be found. The pruning strategies allow for eliminating inadequate services, based on QoS and temporal constraints. To deal with this, a set of constraint optimization models have been proposed. These models can be applied in parallel in order to reduce the computation time of the pruning process. In addition, through the pruning mechanism, failures can be detected at earlier stages before performing the selection process. In case of failures, we provide pruning guided improvement techniques, which aim to try to find a solution so that successful collaboration can be carried out. The result of the pruning step is a set of preselected services for each abstract business task, which will be considered in the selection phase presented in the next chapter.

Chapter 5

Static Service Selection at Design Time

Contents

5.1 Introduction

QoS-based service selection is one of the important requirements in Service Oriented Computing (SOC). A challenging task towards this purpose is the selection of the best combination of services to implement abstract business tasks while meeting quality of service (QoS) constraints required by the user. During the selection process, candidate services are evaluated in terms of both functional and QoS properties. As a large number of services can have similar functionality to realize the awaited abstract tasks, a specific issue that emerges is which services should be selected to form the optimal solution meeting end-user's global QoS constraints.

Several approaches have been proposed in the literature to tackle the problem of service selection. Most of current work considers only static QoS values and do not deal with

temporal properties [16, 17, 66, 67, 72]. Considering time-dependent QoS values and temporal properties makes the selection problem more complex. Indeed, during the selection, mutual dependencies between the different temporal constraints may arise so that the selection of each service may influence or be influenced by the selection of other services. On other side, to find the best solution, all potential combinations of services must be compared. However, the number of these combinations may be very high, which can present a barrier for enabling effective service selection.

In this chapter, we present our service selection approach applied on the set of preselected services (i.e., $\mathcal{S}_{Presi}, \forall A_i \in \mathcal{A}$) discussed in the previous chapter. Based on the results of the pruning process, we propose two selection algorithms: exact and approximate algorithms. The exact approach is adequate in small selection problems, where the optimal solution is required. This approach is presented in Section 5.2. The approximate approach is efficient in large selection problems, where a near-to-optimal solution can be found. It is discussed in Section 5.3. Finally, we conclude the chapter in Section 5.4.

5.2 Exact Service Selection Approach

In this section, we present our approach to select the optimal solution that satisfies all constraints while considering temporal properties. In this approach, all combinations of services are considered. We note that only preselected services after applying our pruning techniques (See Chapter 4) are considered. In our approach, we model the selection problem as a constraint optimization problem (COP). Modeling the selection problem in this way ensures a relevant representation for the dependencies between services as well as the constraints that have to be fulfilled.

When dealing with time-dependent QoS values, determining the start and the finish time of each selected service s_i^s is crucial, since by delaying the execution of a service, QoS values of some attributes may be modified. Thus, in the optimization phase, two types of decision variables are taken into account. The first one is to select a concrete service for each atomic task and the second one is to determine a valid starting time for each selected service in order to match the global constraints. The proposed model selects exactly one atomic service for each abstract task with the corresponding start and finish time while optimizing the overall utility and satisfying all constraints. Hence, in addition to QoS values, two temporal values are specified for each selected service s_i^s: the estimated start time st_i and the estimated finish time ft_i.

Usually, there is no single combination of services that dominates all other solutions in terms of QoS values. To select the best solution, we use the Simple Additive Weighting

method to aggregate QoS values into a single utility value (See Section 3.4). Thus, the selection of the best combination of services can be achieved by optimizing the utility of the composite service. Therefore, the objective function of our optimization model is as follows:

$$\text{maximize} \sum_{y=1}^{m} W_y * \frac{Q(q_y)^{max} - Q(CS,q)}{Q(q_y)^{max} - Q(q_y)^{min}} \tag{5.1}$$

Such that for each $q_y \in \mathcal{QS}$:

$$Q(CS,q_y) = Agg_{A_i \in \mathcal{A}}(\sum_{S_{ij} \in \mathcal{S}_{Presi}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * Q(S_{ijk},q_y)) \tag{5.2}$$

Since the aggregation function depends on the quality attribute and the structure of the business process, we use equations from (3.1) to (3.4) presented in Section 3.4.1.

To guarantee that only one service will be selected for each task, we use a binary decision variable a_{ijk} for each service instance S_{ijk} such that $a_{ijk} = 1$ if the service S_{ijk} is selected for the abstract task A_i and $a_{ijk} = 0$ otherwise, which is presented in the following constraint:

$$\sum_{S_{ij} \in \mathcal{S}_{Presi}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} = 1, \forall A_i \in \mathcal{A} \tag{5.3}$$

We note that in this step, only preselected services after the pruning step are considered. Thus, the minimum and maximum values of each quality attribute for each business task have to be recomputed to consider only preselected services for each abstract business task (i.e., $Q(A_i,q_y)^{min}$ and $Q(A_i,q_y)^{max}, \forall\ q_y \in \mathcal{QS}$ with $1 \leq y \leq m$ and $\forall A_i \in \mathcal{A}$). Constraint (5.4) presents the domain of the quality value of each component service in the combination.

$$Q(A_i,q_y) \in [Q(A_i,q_y)^{min}, Q(A_i,q_y)^{max}], \quad \forall A_i \in \mathcal{A} \tag{5.4}$$

Since all global QoS constraints have to be satisfied when selecting the optimal solution, we add Constraint (5.5):

$$Q(CS,q_y) \leq Q(q_y), \forall q_y \in \mathcal{QS} \tag{5.5}$$

Constraint (5.6) allows selecting only one service for each choice structure:

$$\sum_{A_i \in C_l} p_{li} = 1, \forall C_l \in \mathcal{SC} \tag{5.6}$$

with,

$$p_{li} \in \{0, 1\}, \forall A_i \in \mathcal{C} \tag{5.7}$$

Moreover, we should ensure that the start and finish time of each task belong to the time span of the same selected service instance. For this, we propose the following constraints:

$$\sum_{S_{ij} \in \mathcal{S}_{Presi}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * t^{min}_{S_{ijk}} \leq st_i, \forall A_i \in \mathcal{A} \tag{5.8}$$

$$st_i \leq \sum_{S_{ij} \in \mathcal{S}_{Presi}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * (t^{max}_{S_{ijk}} - Q(S_{ijk}, dur)), \forall A_i \in \mathcal{A} \tag{5.9}$$

To specify the relation between the start and finish time of each task A_i, we add Constraints (5.10) and (5.11):

$$ft_i = st_i + \sum_{S_{ij} \in \mathcal{S}_{Presi}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * Q(S_{ijk}, dur), \forall A_i \notin \mathcal{L} \tag{5.10}$$

$$ft_i = st_i + \alpha_i * \sum_{S_{ij} \in \mathcal{S}_{Presi}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * Q(S_{ijk}, dur), \forall A_i \in \mathcal{L} \tag{5.11}$$

with,

$$st_i, ft_i \in [t^{min}_{A_i}, t^{max}_{A_i}], \forall A_i \in \mathcal{A} \tag{5.12}$$

To check the satisfaction of structural dependencies, we use constraints (4.20) and (4.21). Finally, to check the satisfaction of intra and inter-task temporal constraints, we use constraints from (4.24) to (4.27). Note that it is not worth to check the satisfaction of intra-task QoS constraints as well as the deadline constraint since we consider only pre-selected services. The selected services for all business tasks form the best combination of services.

Although the optimal service selection guarantees the selection of the best combination of services to the user's requirements, it is not practical in large selection problems since the number of feasible solutions can be very big. In fact, this approach enumerates and compares all the possible solutions, which is proven to be NP-hard [66, 67]. This is not practical in real-world applications where a solution has to be selected in a reasonable time. To reduce the computation time and deal with scalability issues, a possible alternative is to select a near-to-optimal solution.

As discussed in Section 2.5.2, several approaches propose heuristics to find a near-to-optimal solution more efficiently than exact solutions. Some methods are based on evolutionary algorithms [11, 84–87]. These algorithms can not be easily applied in heavily constrained problems. Others alternatives propose decomposition strategies to decompose global constraints into local ones [16, 72–74]. However, they are not appropriate when handling time-dependent QoS values. In fact, selecting the best service for each abstract task based on local QoS constraints can not guarantee that the temporal constraints will be satisfied. To overcome the limitations of existing approaches, in the next section, we present a heuristic service selection approach while taking into account both QoS and temporal constraints.

5.3 Approximate Service Selection Approach

In order to deal with large service selection problems where the number of preselected services after the pruning phase is very large, in this section, we present our heuristic based service selection approach to select a close-to-optimal solution. The proposed approach is based on clustering and decomposition techniques to select the best local QoS and temporal constraints for each task. Based on these latter, a local service selection is applied to select the best service for each task. The heuristic approach proceeds through four phases:

- *A. Service clustering:* This phase allows to identify a set of classes, called centroids, for each abstract business task. The aim is to identify QoS levels of candidate services of each task and then, associate each service to the adequate centroid.
- *B. Selection of the best centroids:* This step allows to select the best centroids based on their utilities such that all constraints are satisfied. The quality values of the selected centroids will be further considered as local QoS constraints that have to be respected in the selection phase.
- *C. Deadline decomposition:* The goal is to define the largest time intervals for each abstract task based on the local duration constraints such that all temporal

constraints are fulfilled. The obtained intervals will be considered as local temporal constraints to guide the selection of a near-to-optimal solution.

- *D. Local selection:* Finally, this step enables the selection of a near-to-optimal combination of services based on local selection of the best service for each abstract task.

Hereafter, we detail each step.

5.3.1 Service Clustering

Existing constraints decomposition approaches [16, 73, 74] usually deal with QoS parameters of each candidate service independently and do not consider potential correlations among them. This may lead to a greedy decomposition method with local QoS constraints that cannot be fulfilled by any candidate service even though a solution does exist. To cope with this limitation, we propose a clustering based approach to identify local QoS constraints while dealing with correlations among QoS values of each candidate service.

The clustering phase is performed locally for each abstract task in the business process. It aims to classify candidate services of each abstract business task into a set of clusters (i.e., QoS levels) according to their QoS values. Each cluster contains services that have approximately similar QoS values. The purpose of this classification is to define the most important cluster for each task with respect to the number of its candidate services and their QoS values. These levels will be considered to identify the adequate local QoS constraints for each abstract task. To do so, we use clustering techniques in particular, the K-means algorithm [132].

5.3.1.1 K-means Algorithm Overview

The K-means algorithm is commonly used to automatically partition a data set into a fixed number of groups (i.e., clusters). The main idea of this algorithm is to define a centroid for each group and then, associate each data point to the appropriate centroid (i.e., the centroid that has the shortest distance with the data point according to multiple parameters). For instance, suppose a data point x_i, which is characterized by a set of values defined by the vector $\langle Q(x_i, q_1), Q(x_i, q_2), ..., Q(x_i, q_m)\rangle$ and a centroid c characterized by the vector $\langle Q(c, q_1), Q(c, q_2), ..., Q(c, q_m)\rangle$, thus, the Euclidean distance between x_i and c can be defined as follows:

$$D(x_i, c) = \sqrt{\sum_{y=1}^{m} (Q(c, q_y) - Q(x_i, q_y))^2} \tag{5.13}$$

The values of the centroids are then updated by computing the average of the values of all their associated data points for each parameter. The clustering and the updating steps can be repeated until there is no changes in the values of the centroids or until a stop criteria is reached (e.g., convergence threshold, maximum number of iterations).

5.3.1.2 Classification of Services

In our approach, we use K-means clustering to associate services into a set of QoS levels. Each candidate service is considered as a data point, which is characterized by its QoS values denoted by $S_{ijk} = \langle Q(S_{ijk}, q_1), Q(S_{ijk}, q_2), ..., Q(S_{ijk}, q_m) \rangle$. Furthermore, each vector of QoS levels is considered as a centroid. In this step, the range of each quality attribute $q_y \in \mathcal{QS}$ is partitioned into a set of K discrete quality levels for each abstract task where K is a constant number strictly greater than 1. We suppose that the number of levels (i.e., K) is fixed by domain experts and can be different from one task to another according to the values of candidate services. This number can be determined using several techniques (e.g., [133, 134]). In the following, we denote by QL_{iy}^{z} the QoS value of the attribute q_y for the z^{th} level of the task A_i with $1 \leq z \leq K$ and $1 \leq y \leq m$. To speed up the classification algorithm, we compute the initial values of QoS levels as follows:

$$QL_{iy}^{z} = Q(A_i, q_y)^{min} + \frac{z-1}{K-1} * (Q(A_i, q_y)^{max} - Q(A_i, q_y)^{min}), \forall A_i \in \mathcal{A} \tag{5.14}$$

Figure 5.1 shows an example of 3 clusters of candidate services of an abstract task using K-means algorithm.

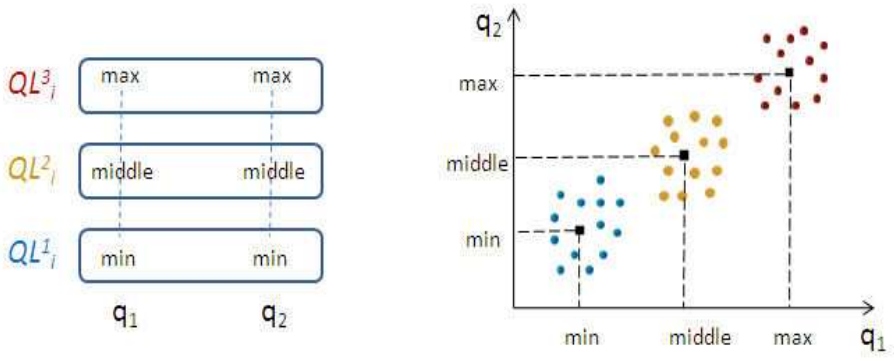

FIGURE 5.1: Example of 3 clusters in two-dimensional space using K-means

Hence, based on the values of QoS levels, the initial set of centroids can be defined. Let's denote by $QL_i = \langle QL_i^1, QL_i^2, ..., QL_i^K \rangle$ the set of centroids of the task A_i with $QL_i^z = \langle QL_{i1}^z, QL_{i2}^z, ..., QL_{im}^z \rangle$ denotes the z^{th} centroid of A_i for each $1 \leq z \leq K$.

Once all centroids are defined, we assign each candidate service to the closest centroid using the Euclidean distance (as defined in Section 5.13). Since the domains of the values of the different quality attributes can be different, QoS values of candidate services and centroids are normalized. The new value of each candidate service $S_{ijk} \in \mathcal{S}_{Presi}$ for each quality attribute q_y with $1 \leq y \leq m$ is computed based on the equation below:

$$Q(S_{ijk}, q_y)' = \begin{cases} \frac{Q(A_i,q_y)^{max} - Q(S_{ijk},q_y)}{Q(A_i,q_y)^{max} - Q(A_i,q_y)^{min}} & \text{if} \quad Q(A_i, q_y)^{max} - Q(A_i, q_y)^{min} \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.15}$$

By the same manner, the scaling is applied also to the values of centroids (i.e., $QL_{iy}^{z'}, \forall 1 \leq y \leq m, \forall A_i \in \mathcal{A}, \forall 1 \leq z \leq K$). Hence, the Euclidean distance between a centroid QL_i^z and a service S_{ijk} is as follows:

$$D(S_{ijk}, QL_i^z) = \sqrt{\sum_{y=1}^{m} (QL_{iy}^{z'} - Q(S_{ijk}, q_y)')^2} \tag{5.16}$$

With $QL_{iy}^{z'}$ denotes the new value of the centroid $QL_{iy}^{z'}$ after the scaling phase $\forall 1 \leq y \leq m, \forall A_i \in \mathcal{A}, \forall 1 \leq z \leq K$.

An example of a set of clusters for abstract business tasks is depicted in Figure 5.2.

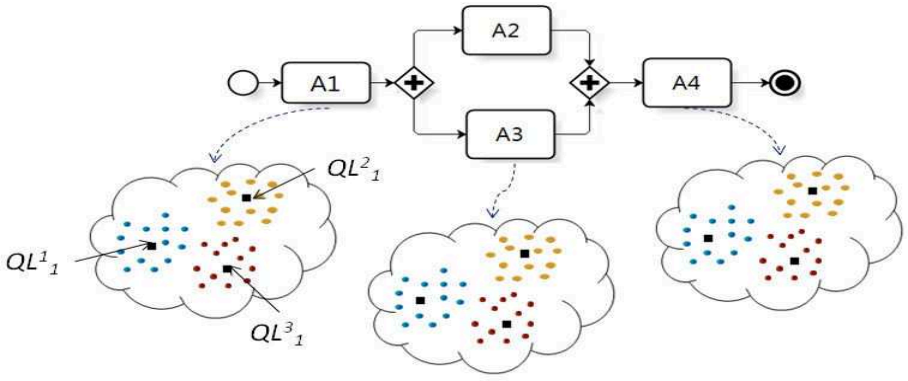

FIGURE 5.2: Example of a set of clusters

5.3.2 Local QoS Constraints Specification

This phase aims to compute local QoS constraints for each business task based on the set of centroids. It proceeds through two main steps: centroid utilities and the selection of the best centroids.

5.3.2.1 Centroid Utilities

To identify local QoS constraints, first, we assign each centroid QL_i^z an utility value (i.e., $U(QL_i^z)$) between 0 and 1, which estimates the benefit of using the QoS values of this centroid as local QoS constraints. The utility value of each centroid is computed as follows:

$$U(QL_i^z) = U_q(QL_i^z) * \frac{r(QL_i^z)}{c_i}, \forall A_i \in \mathcal{A}, \forall 1 \leq z \leq K \tag{5.17}$$

Where:

$$U_q(QL_i^z) = \sum_{y=1}^{m} W_y * \frac{Q(A_i, q_y)^{max} - QL_{iy}^z}{Q(A_i, q_y)^{max} - Q(A_i, q_y)^{min}} \tag{5.18}$$

The first part (i.e., $U_q(QL_i^z)$) specifies the utility of the centroid based on its QoS values. The second part (i.e., $\frac{r(QL_i^z)}{c_i}$) allows giving better utility value to the centroid that has more candidate services where $r(QL_i^z)$ is the number of candidate services of the centroid QL_i^z and c_i is the total number of services of the task A_i.

5.3.2.2 The Selection of the Best Centroids

The second step allows to identify the best centroid of each business task. QoS values of the selected centroids will be considered as local QoS constraints in the selection process. We propose a constraint optimization model to find the best centroids such that all global QoS constraints are satisfied. To select only one centroid for each task, we use a binary decision variable x_i^z for each centroid such that $x_i^z = 1$ if the centroid QL_i^z is selected for the abstract task A_i and $x_i^z = 0$ otherwise, which is expressed in Constraint (5.19).

$$\sum_{z=1}^{K} x_i^z = 1, \forall A_i \in \mathcal{A}, x_i^z \in \{0, 1\} \tag{5.19}$$

The goal of the objective function (5.20) is to maximize the utility value of the set of the selected centroids in order to reduce the number of discarded services.

$$\text{maximize} \quad \sum_{A_i \in \mathcal{A}} \sum_{z=1}^{K} U(QL_i^z) * x_i^z \tag{5.20}$$

To guarantee that the QoS values of the selected centroids ensure that the global QoS constraints will be satisfied, we add Constraint (5.21).

$$Agg_{A_i \in \mathcal{A}} \ (\sum_{A_i \in \mathcal{A}} \sum_{z=1}^{K} QL_{iy}^z * x_i^z) \leq Q(q_y), \forall 1 \leq y \leq m \tag{5.21}$$

To select only one centroid for each choice structure, we add constraints (5.6) and (5.7). Additionally, the selected centroids must ensure that the overall deadline is fulfilled. To do so, we add Constraint (5.22), which guarantees that the sum of the minimum start time of the first task and the aggregated duration value of the selected centroids satisfies the required deadline. In this constraint, $t_{A_1}^{min}$ indicates the minimum possible start time of the first business task A_1 (See Section 3.4.2) and y refers to the execution duration attribute.

$$t_{A_1}^{min} + Agg_{A_i \in \mathcal{A}} \ (\sum_{A_i \in \mathcal{A}} \sum_{z=1}^{K} QL_{iy}^z * x_i^z) \leq deadline \tag{5.22}$$

5.3.3 Deadline Decomposition

Unlike existing approaches, which handle static QoS values [16, 74], specifying local QoS constraints does not guarantee that all services that satisfy these constraints can collaborate when considering time-dependent QoS values. For instance, suppose that the business process has two abstract tasks A_1 and A_2, which will be executed in sequence with A_1 precedes A_2. Let's denote by s_1^* and s_2^* the best services that satisfy all local QoS constraints of the two tasks A_1 and A_2, respectively. Suppose now that the service s_2^* is available in a time span before that of the service s_1^*. In this case, the two services can not be part of a feasible solution even though they satisfy all local QoS constraints.

To this end, temporal properties have to be considered also to identify local temporal constraints that have to be satisfied by the selected services to guarantee that all selected services can collaborate together. To do so, we identify four variables for each task: earliest start time est^m, earliest finish time eft^m, latest start time lst^M and latest finish time lft^M. The values of these variables are defined based on the minimum and the

maximum duration values of each task and should guarantee that all intra and inter-task temporal constraints are satisfied and the overall deadline is respected.

Example 5.1. Let's take the example presented in Figure 5.3. In this example, we consider two abstract business tasks A_1 and A_2. For each task, we compute the largest time interval based on both minimum and maximum duration values as well as the minimum start time and maximum finish time of its candidate services. In this example, we suppose that the deadline is equal to 38 time units. The local temporal values of each task are depicted in Figure 5.3.

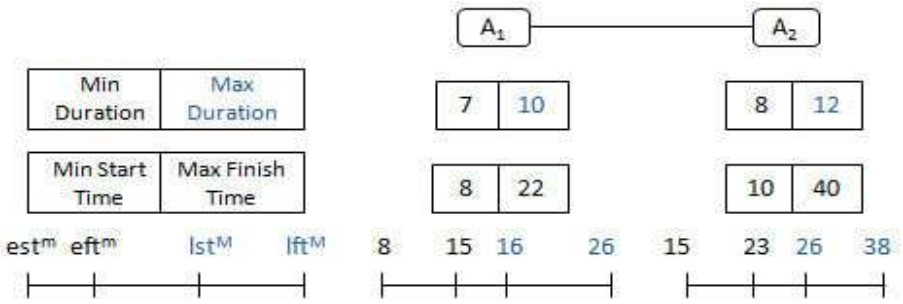

FIGURE 5.3: Example of deadline decomposition

The specification of the local temporal constraints of each activity is not a trivial task when handling several temporal constraints and complex business structures. To deal with this, time intervals are identified based on constraint optimization model. This model can be applied in parallel for each task $A_i \in \mathcal{A}$.

The objective function (5.23) allows for maximizing the value of lft_i^M while minimizing the value of est_i^m.

$$\text{maximize} \quad lft_i^M - est_i^m \tag{5.23}$$

To specify the dependencies between the start and the finish time of each task, we add Constraints from (5.24) to (5.27). Here, we consider that the difference between est_i^m and eft_i^m is equal to the minimum duration value for the task A_i (i.e., $Q(A_i, dur)^{min}$) whereas the difference between lst_i^M and lft_i^M is equal to the duration of the selected centroid for the task A_i (i.e., QL_{iy}^z), which is considered as the maximum duration value of A_i, with z refers to the selected centroid of the task A_i and y refers to the duration attribute.

$$eft_i^m = est_i^m + Q(A_i, dur)^{min}, \forall A_i \notin \mathcal{L} \tag{5.24}$$

$$lft_i^M = lst_i^M + QL_{iy}^z, \forall A_i \notin \mathcal{L} \tag{5.25}$$

$$eft_i^m = est_i^m + \alpha_i * Q(A_i, dur)^{min}, \forall A_i \in \mathcal{L} \tag{5.26}$$

$$lft_i^M = lst_i^M + \alpha_i * QL_{iy}^z, \forall A_i \in \mathcal{L} \tag{5.27}$$

To satisfy precedence dependencies, we add Constraints (5.28) and (5.29).

$$eft_i^m \leq est_v^m, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v) \tag{5.28}$$

$$lft_i^M \leq lst_v^M, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v) \tag{5.29}$$

Besides, to guarantee that local temporal constraints are fulfilled, we add Constraints (5.30) and (5.31) while considering only the temporal constraints (SNLT and FNLT).

$$lst_i^M \leq T, \forall tc_i(SNLT, T) \in \mathcal{TC} \tag{5.30}$$

$$lft_i^M \leq T, \forall tc_i(FNLT, T) \in \mathcal{TC} \tag{5.31}$$

Finally, Constraints (5.32) and (5.33) deal with temporal constraints. For simplicity, in this model, we only consider finish-to-start inter-task temporal constraints. Thus, the following constraints have to be satisfied:

$$eft_i^m + D_{iv}^{min} \leq est_v^m \leq eft_i^m + D_{iv}^{max}, \forall\ td_{i,v}(FS, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD} \tag{5.32}$$

$$lft_i^M + D_{iv}^{min} \leq lst_v^M \leq lft_i^M + D_{iv}^{max}, \forall\ td_{i,v}(FS, D_{iv}^{min}, D_{iv}^{max}) \in \mathcal{TD} \tag{5.33}$$

with,

$$est_i^m, eft_i^m, lst_i^M, lft_i^M \in [t_{A_i}^{min}, t_{A_i}^{max}], \forall A_i \in \mathcal{A} \tag{5.34}$$

The start and finish time of each task will be considered as local temporal constraints and thus, used in the next step to find a candidate service for each abstract task.

5.3.4 Local Selection

After defining local QoS and temporal constraints, the last step of our approach is to select a close-to-optimal solution. The local selection aims to find the best service for each business task such that all local QoS and temporal constraints are fulfilled. The best service is the service that has the best utility value amongst all the candidate

services of the corresponding service class. The local selection can be applied in parallel for each task. The selection process is presented in Algorithm 4.

Algorithm 4 Local Service Selection for a Task A_i

```
1:  for each S_ijk ∈ S_Presi do
2:      ComputeUtility(S_ijk)
3:  RankServices(A_i)
4:  for each S_ijk ∈ S_Presi do
5:      for each q_y ∈ QS do
6:          if Q(S_ijk, q_y) ≥ Q_L(A_i, q_y) then
7:              break
8:      if t^min_{S_ijk} ≤ est^m_i and t^max_{S_ijk} ≥ lft^M_i then
9:          SelectService(S_ijk)
10:         DefineStartTimeInterval(S_ijk)
11:         break
```

First, we rank candidate services of each business task according to their utilities (lines 1 to 3). The utility of each service is quantified by the utility function declared in (5.35). To avoid local optimums, we compare the distance between the quality value of the service and the minimum quality value of its corresponding task with the distance between the minimum and maximum aggregated values for each quality attribute.

$$U(S_{ijk}) = \sum_{y=1}^{m} W_y * \frac{Q(A_i, q_y)^{max} - Q(S_{ijk}, q_y)}{Q(q_y)^{max} - Q(q_y)^{min}} \tag{5.35}$$

For each task, we select the candidate service with the best utility value and that meets all local QoS (lines 4 to 7) and temporal (lines 8 to 9) constraints of its corresponding task. Local QoS constraints are considered as upper bounds for QoS values of the different candidate services where $Q_L(A_i, q_y)$ denotes the local QoS constraint of the attribute q_y for the task A_i. To deal with temporal constraints, a candidate service is selected if its time span is larger than the interval $[est^m, lft^M]$. Here, we note that the time span of each service instance covers its execution duration (i.e., $t^{max}_{S_{ijk}} - t^{min}_{S_{ijk}} \geq Q(S_{ijk}, dur)$). For each selected service, we identify the time span in which it can start execution so that it can collaborate with other selected services (line 10). To do so, we add the following constraint:

$$max(t^{min}_{S_{ijk}}, est^m_i) \leq st_i \leq lft^M_i - Q(S_{ijk}, dur) \tag{5.36}$$

With st_i and ft_i are the start and the finish time of the selected service of the task A_i. Hence, if this constraint is satisfied, we can guarantee that the selected services can form a feasible solution. The start time of each service is then defined at run time

with respect to the Constraint (5.36) and the start and the finish time of its precedent services. In the following, we demonstrate that the selected services after a local selection can collaborate with each other while satisfying all constraints.

Lemma 5.1. *If all the selected services satisfy local QoS constraints and time spans (identified by the deadline decomposition step) and guarantee Constraint* (5.36)*, they can collaborate with each other and all local and global constraints are fulfilled.*

The proof of Lemma 5.1 is given in Appendix A.4.

5.4 Conclusion

In this chapter, we have presented exact and approximate approaches to solve the service selection problem. The proposed approaches allow handling both small and large selection problems. The exact approach is based on constraint optimization model to select the best solution. Although this approach allows selecting the best combination of services, it can not be applied in large service selection problems where the number of services, tasks and constraints may be very high. For this reason, we have proposed an approximate service selection approach. This approach is based on clustering and constraints decomposition techniques to identify local QoS and temporal constraints from the global ones.

The proposed approaches can only be applied on request time before executing the selected services. Service selection at request time does not guarantee that the selected services offer the estimated values during execution. Hence, a dynamic service selection approach at run-time is required to guarantee that the selected solution is still satisfactory during execution. This approach is discussed in the next chapter.

Chapter 6

Dynamic Service Selection at Run-time

Contents

6.1 Introduction

The successful execution of business processes according to user needs requires that the values of all selected services are compliant with their corresponding constraints. Static service selection approaches are based on estimated values to obtain an ex-ante service composition. Service oriented systems often operate in highly uncertain and dynamic environments. Thus, during execution, actual values of candidate services may deviate from the estimated ones, which may cause the violation of business constraints. For example, due to a high overhead, the response time of a selected service may be higher than estimated. Such deviations might affect the successful execution of the service combinations during execution and thus, the satisfaction of end-to-end constraints. In addition, during the execution of services, several changes may occur due to the dynamic

nature of service systems. Indeed, services can join the system, become unavailable, or their offerings can change.

To ensure a reliable execution of service compositions, the selected combination of services is monitored during execution so that re-selection actions can be triggered each time new events that may affect the process execution, arise. The aim of the dynamic selection is to adjust the selected service combination so that all constraints can be guaranteed and the optimality of the selected solution remains reasonable. This is a complex task especially when handling several constraints and dependencies between services mainly the temporal constraints. Existing approaches that deal with service failures, constraints violations and environment changes do not consider specific characteristics related to the presence of temporal properties. Moreover, they usually delay the adaptation until a violation of a global constraint occurs, which may lead to undesired effects such as the inability to find a feasible solution.

In this chapter, we propose a proactive dynamic service selection approach to handle several changes and violations that can be observed at different stages of the execution while ensuring that the new services still meet the original end-to-end global constraints. We assume that a *primary service combination* is already identified using static selection approaches presented in Chapter 5. The proposed approach reacts to changes and violations as soon as they occur to guarantee the successful execution of the service combination and reduce the occurrence of violations during execution. In contrast to existing approaches, in addition to QoS offerings, we consider temporal constraints and their dependencies. The remainder of this chapter is organized as follows. In Section 6.2, we describe a motivating scenario. In Section 6.3, we detail our service selection approach to deal with both service violations and environment changes during service execution. Finally, we conclude the chapter in Section 6.4.

6.2 Motivating Scenario

To better illustrate the usefulness of the proactive service selection, let's consider the example presented in Section 1.3. Candidate service instances of each task are presented in Figure 3.2. Suppose that the initial selected combination of services is $CS^* = (S_{111}, S_{211}, S_{311}, S_{431}, S_{531}, S_{611})$. In the following, we give some examples of violations and changes that might occur during the execution of the selected solution.

- *Scenario 1*: Suppose that while the service S_{111} is executing, S_{311} changes its quality values to 7 units for the execution duration and 10 units for the cost attribute. In this case, the solution is no more feasible since the global execution

duration constraint is violated. Delaying the substitution of services until the service S_{311} starts the execution will lead to the inability to find a feasible solution since all service combinations including the services S_{111}, S_{211} and S_{311} violate global constraints. Hence, the service S_{311} should be substituted by S_{331} during the execution of S_{111} to obtain a satisfactory solution.

- *Scenario 2*: Suppose that during its execution, the service S_{211} takes 2 time units rather than 1. In this case, the global execution duration is not exceeded and the deadline is respected. Suppose now that another violation occurs when the service S_{611} is executing. This service takes 2 time units rather than 1. In this case, the service can not offer the expected cost and the overall deadline will be violated. Here, replacing the service S_{531} by S_{512} after the violation of the service S_{211} will result in a satisfactory solution even if the violation occurs on the service S_{611}. Thus, reacting to changes as soon as they occur allows for minimizing the interruption time while increasing the chance to find a feasible solution after violation.

- *Scenario 3*: Suppose that while executing the service S_{211}, the service S_{512} performs better with a cost equal to 10 rather than 14. In this case, replacing the service S_{531} by the service S_{512} will lead to a better optimality for the selected combination of services.

6.3 Proactive Service Selection

In this section, we detail our time-aware proactive service selection approach. This approach relies on computed maximum and intermediary thresholds, which are used at run time to trigger corresponding re-selection actions (Section 6.3.1). Based on these thresholds, a set of the most relevant candidate services is selected for each business task to enhance the efficiency of the re-selection approach (Section 6.3.2). Finally, according to classes of changes that may arise at run time, re-selection actions are performed (Section 6.3.3). The re-selection process relies on a local and a region approaches (Section 6.3.4). We propose an iterative method to define re-selection regions to include a small set of tasks to be considered rather than considering all non-executed tasks.

6.3.1 Specification of Re-selection Thresholds

To get control over the execution of the process, for each task we define *maximum* and *intermediary* thresholds for each task. If at least one maximum threshold of one task is

violated, the selected solution is no more feasible and thus, it should be modified. Moreover, intermediary thresholds are specified for each abstract task so that if the values of a service exceed at least one of these thresholds, re-selection actions are triggered in order to minimize the likelihood of possible violations.

6.3.1.1 Computing Maximum Thresholds

The maximum thresholds are computed considering the values of the selected services (i.e., services of the selected combination of services CS^*) and the global constraints. For instance, if we consider the example presented in Section 1.3, maximum thresholds for the cost attribute of the tasks A_1, A_2, A_3, A_4, A_5 and A_6 are equal to 15, 13, 15, 15, 15 and 10, respectively, considering the selected combination of services $CS^* = (S_{111}, S_{211}, S_{311}, S_{431}, S_{531}, S_{611})$. Consequently, if for instance the service S_{111} of the first task changes its cost value to 16 rather than 12, the global execution duration will be violated and the selected solution should be modified.

The set of the maximum execution thresholds of each business task $A_i \in \mathcal{A}$ is denoted by $T_i^M \langle T_{i1}^M, .., T_{iy}^M, .., T_{im+2}^M \rangle$ with $1 \leq y \leq m+2$. The maximum threshold for a quality attribute q_y except the duration attribute for a task A_b (i.e., T_{by}^M) is computed using the following constraint optimization model.

$$\text{maximize} \quad Q(A_b, q_y) \tag{6.1}$$

$$Q(A_i, q_y) = Q(s_i^s, q_y), \forall A_i \in \mathcal{A}, i \neq b \tag{6.2}$$

$$Agg_{A_i \in \mathcal{A}}(Q(A_i, q_y)) \leq Q(q_y), \forall A_i \in \mathcal{A} \tag{6.3}$$

$$Q(A_b, q_y) \in [Q(s_b^s, q_y), Q(q_y)] \tag{6.4}$$

Maximum QoS threshold of the task A_b is equal to its maximum allowed quality value (i.e., $Q(A_b, q_y)$) while considering the quality values of the selected services of all other tasks (i.e., $Q(s_i^s, q_y)$ $\forall A_i \in \mathcal{A} \mid i \neq b$) (Constraints (6.1) and (6.2)). Constraint (6.3) guarantees the satisfaction of the global QoS constraint. To guarantee that the maximum threshold fulfills the intra-task QoS constraints, Constraints (4.8) and (4.9) can be used. Here, we note that intra-task constraints are applied only for the task for which we compute the maximum threshold (i.e., A_b) since the values of the quality attribute for all other tasks are fixed to the values of their corresponding selected services. Finally, Constraint (6.4) indicates the domain of the maximum quality value of the task A_b. We note that for choice structure, we suppose that paths that will be executed are identified in the selection phase.

To compute maximum thresholds for the duration attribute as well as maximum temporal threshold, we propose the model from (6.5) to (6.10).

$$\text{maximize} \quad Q(A_b, dur) \tag{6.5}$$

$$Agg_{A_i \in \mathcal{A}}(Q(A_i, dur)) \leq Q(dur), \forall A_i \in \mathcal{A} \tag{6.6}$$

$$ft_i \leq st_v, \forall A_v \in \mathcal{A}, A_i \in \mathcal{P}d(A_v), A_i \notin \mathcal{A} \tag{6.7}$$

$$Q(A_i, dur) = Q(s_i^s, dur), \forall A_i \in \mathcal{A}, i \neq b \tag{6.8}$$

$$Q(A_b, dur) \in [Q(s_b^s, dur), Q(dur)] \tag{6.9}$$

$$st_i, ft_i \in [t_{s_i^s}^{min}, t_{s_i^s}^{max}], \forall A_i \in \mathcal{A} \tag{6.10}$$

Maximum threshold of each task A_b for the duration attribute is equal to its maximum allowed quality value (i.e., $Q(A_b, dur)$). The objective function (6.5) allows for maximizing the duration value of the task A_b. Constraint (6.6) accounts for the global duration constraint that has to be fulfilled. Constraint (6.7) deals with dependencies between tasks. The duration of each task $A_i \in \mathcal{A}, i \neq b$ is equal to the duration of its selected service (Constraint (6.8)). The domain of the maximum threshold is presented in Constraint (6.9). Finally, the start and the finish time of each task should be in the time interval of the selected service (Constraint (6.10)). To guarantee the satisfaction of the deadline, we use Constraint (4.14). To deal with dependencies between the start and the finish time of the different tasks, we use Constraints form (4.15) to (4.16). To deal with intra and inter-task constraints, we use Constraints from (4.22) to (4.27). Again, intra-task constraints are applied only for the task A_b. Maximum temporal thresholds for a task A_b (i.e., T_{bm+1}^M and T_{bm+2}^M) represent its latest start time and its latest finish time, respectively. These thresholds should not be exceeded during execution for each task. The latest finish time of the task A_b is equal to its finish time after applying the constraint optimization model (i.e., ft_b). Whereas its latest start time is equal to $ft_b - Q(s_b^s, dur)$.

6.3.1.2 Computing Intermediary Thresholds

The intermediary thresholds are used to trigger a proactive re-selection actions prior to a global constraint violation. The aim is to avoid delaying the re-selection until a violation of a global constraint occurs on the one hand, and, on the other hand avoid triggering re-selection actions each time a violation is observed, which can decrease the efficiency of the proposed approach. The set of intermediary thresholds for each task

$A_i \in \mathcal{A}$ is denoted by $T_i^I \langle T_{i1}^I, .., T_{iy}^I, .., T_{im+2}^I \rangle$ with $1 \leq y \leq m+2$. These thresholds are computed as follows:

$$T_{iy}^I = \begin{cases} \frac{T_{iy}^M + p_y^{s_i^s}}{2} & if \quad 1 \leq y \leq m \\ \frac{T_{iy}^M + st_i}{2} & if \quad y = m+1 \\ \frac{T_{iy}^M + ft_i}{2} & if \quad y = m+2 \end{cases} \tag{6.11}$$

6.3.2 Pertinent Services

Pertinent services are used as backup during the re-selection. The aim is to avoid taking into consideration all candidate services of each task, which can negatively influence the performance of the re-selection step. The set of *pertinent services* for each task $A_i \in \mathcal{A}$ is denoted by $\mathcal{S}_{Peri}$. In the following, we explain how these services are identified and ranked to be used at run-time. For simplicity, in the rest of this chapter, we use s_i to indicate a service instance of the task A_i.

6.3.2.1 Identifying Pertinent Services

The first step is to prune all services that do not violate the maximum thresholds. Algorithm 5 summarizes the process of selecting pertinent services for each abstract task $A_i \in \mathcal{A}$.

Algorithm 5 Identifying Pertinent Services for a Task A_i

1: **Input:** The set of candidate services $\mathcal{S}_i \setminus \{s_i^s\}$ of each task A_i
2: **Output:** The set of pertinent services $\mathcal{S}_{Peri}$
3: $\mathcal{S}_{Peri} = \emptyset$
4: **for** each $s_i \in \mathcal{S}_i \setminus \{s_i^s\}$ **do**
5: **if** *isPertinentService*(s_i, T_i^M) **then**
6: $\mathcal{S}_{Peri} = \mathcal{S}_{Peri} \cup \{s_i\}$

The proposed algorithm takes as input the set of all candidate services $\mathcal{S}_i \setminus \{s_i^s\}$ of each task A_i and returns the set of pertinent services $\mathcal{S}_{Peri}$. A service s_i is considered pertinent if $\forall\ 1 \leq y \leq m+2$, $p_y^{s_i}$ respects T_{iy}^M, hereafter denoted by ($p_y^{s_i}$ res T_{iy}^M), with:

$$p_y^{s_i} \ \ res \ \ T_{iy}^M \ \text{ iff } \begin{cases} p_y^{s_i} \leq T_{iy}^M & \text{if} \quad 1 \leq y \leq m \\ [X, Y] \geq Q(s_i, dur) & \text{otherwise} \end{cases} \tag{6.12}$$

With $[X, Y] = [t_{s_i}^{min}, t_{s_i}^{max}] \cap [st_i, ft_i]$ and st_i and ft_i are the time values computed using the model from (6.5) to (6.10). In other words, the maximum quality threshold

is respected if the quality value of the service is less than or equal to the threshold $\forall 1 \leq y \leq m$ since we consider only quality attributes with decreasing value direction. The maximum temporal thresholds (i.e., T^M_{im+1} and T^M_{im+2}) are respected by a service s_i if the intersection between its time span and the interval $[st_i, T^M_{im+2}]$ covers its execution duration value (i.e., $Q(s_i, dur)$).

6.3.2.2 Ranking Pertinent Services

Once the pertinent services are identified for each task, we rank them according to their scores. Consequently, in case of re-selection, the best pertinent service (i.e., the first service) will be selected. This allows to reduce the delay of the substitution of failed services since these latter can be substituted by the first pertinent service in most cases without the need to re-select services for all non-executed tasks and waiting for the result of the re-selection.

The main idea is to give a better score to the service that has the largest distance compared to the set of its corresponding thresholds so that if a violation is observed at run-time, the best service has more chance to be selected giving its quality values without violating the required constraints. To do so, we compute the distance between each service and the maximum thresholds of its corresponding task based on its quality values for all quality parameters. The score of each service $s_i \in \mathcal{S}_{Peri}$, denoted by $sr(s_i)$, is computed as follows:

$$sr(s_i) = \sum_{l=1}^{m} W_l * D_l(s_i) \tag{6.13}$$

With,

$$D_l(s_i) = \begin{cases} \frac{T^M_{il} - Q(s_i, q_l)}{T^M_{il} - Q(A_i, q_l)^{min}} & \text{if} \quad T^M_{il} \neq Q(A_i, q_l)^{min} \\ 1 & \text{otherwise} \end{cases} \tag{6.14}$$

The service that has the utmost score will be best ranked. Here, we point out that the set of pertinent services is updated during execution to deal with potential changes (e.g., new services can be added, existing services can be removed).

6.3.3 Event Classes and Recovery Actions

Usually, adaptation approaches handle possible changes only after a violation of global constraints is observed or when the service that has to be executed is no more available.

In other words, even when changes, that do not affect end-to-end constraints occur at an early stage, they will be considered only if the selected solution is no more feasible, which leads to undesired effects such as the inability to find a satisfactory solution or a significant interruption time due to for instance the unavailability of the service that has to be invoked. Moreover, most of the current approaches propose to re-select all non-executed services from scratch, which causes a significant delay on the execution of the selected services.

To deal with these limitations, we propose a proactive service re-selection approach that deals with possible changes during execution at early stages in order to prevent possible violations. The main idea is to react to changes as soon as they occur during the process execution in order to prevent the interruption time of the execution when searching for a feasible solution. To avoid possible violations, we propose to proactively update the selected services during execution based on the information of the already executed parts of the process while minimizing the number of re-selected services. In addition, the proposed approach allows enhancing the selected solution in response to not only changes in the values of the selected services but also changes in the environment (e.g., a new better service is added) so that the optimality of the selected solution during execution is maintained.

6.3.3.1 Categories of Changes

During execution, each abstract task $A_i \in \mathcal{A}$ has an execution state denoted by es_i, which can be assigned to one of the following states: *ae* to denote that the task is already executed and its execution is completed, *ce* to denote a partial execution (i.e., the task is currently executed and its execution is not yet completed) and *ne* to denote a task that is not yet executed. In Table 6.1, we give some notations used in this chapter.

Changes occurring during execution can be classified into two main categories: *changes in the service that is currently in execution* (i.e., $s_i^s \in CS^* \mid es_i = ce$) and *changes in the environments* (i.e., changes in the selected and not executed services and changes in the set of pertinent services). These latter can be an *addition* of a new service, a *deletion* of a service or a *modification* in the values of a service. Each of these categories can be further classified into two sub-categories: ignored changes, which are *without effects* and changes *with potential effects* and that have to be considered.

- **Changes without effects**: these changes affect neither the set of pertinent services $\mathcal{S}_{Per}$ nor the selected solution CS^* (i.e., $\mathcal{S}^n_{Peri} = \mathcal{S}^o_{Peri}, \forall A_i \in \mathcal{A}$ and $CS^{*n} = CS^{*o}$).

- **Changes with potential effects**: these changes affect the set of pertinent services $\mathcal{S}_{Per}$ and can be further divided into changes that *do not affect* the selected solution CS^*

(i.e., the selected services are the same identified by the static selection approach) and changes that *affect* the selected solution. Here we note that if a selected service changes its values but it is preserved in the selected solution, these changes are considered as *not affecting* changes. A change is considered as *affecting* if at least one selected service has to be substituted by another candidate service.

In the next section, we give a detailed presentation of the different cases when each of the aforementioned categories is identified as well as their corresponding re-selection actions.

TABLE 6.1: Some notations.

s_i^n	The new added service for the task A_i
s_i^d	The deleted service of the task A_i
s_i^o	The old service of the task A_i before modification
s_i^m	The new service of the task A_i after modification
s_i^1	The first ranked service in $\mathcal{S}_{Peri}$
$sr'(s_i)$	The new score of s_i considering the new values of thresholds of A_i
$\mathcal{S}_{Peri}^o$	The old set of supplementary services of the task A_i
$\mathcal{S}_{Peri}^n$	The new set of supplementary services of the task A_i
CS^{*o}	The old selected combination of services
CS^{*n}	The new selected combination of services
$\neg(p_y^{s_i}$ sat $T_{iy})$	The value of the attribute p_y does not satisfy the threshold T_{iy}

6.3.3.2 Recovery Actions

Changes occurring during run time can either affect the set of the pertinent services or both pertinent services and the selected solution. Depending on the type of the changes, several actions can be triggered (See Table 6.2). For this reason, we define four main actions that have to be applied in the different cases we consider.

Action$_1$: *Update pertinent services of a concerned non-executed task A_i (i.e., $\mathcal{S}_{Peri}$)*

This action takes place when for a given task A_i : (i) a new service is added to the set of pertinent services or (ii) an existing service changes its values but still satisfies maximum thresholds. In these two cases, the score of the concerned service has to be computed based on Constraint (6.13) and the service will be ranked according to the value of its score.

Action$_2$: *Update maximum and intermediary thresholds and pertinent services of all non-executed tasks (i.e., T_i^I, T_i^M and $\mathcal{S}_{Peri}$ $\forall A_i \in \mathcal{A} \mid es_i = ne$)*

Each time a change occurs on the selected solution, our approach proceeds on two main steps: (1) As the values of thresholds are computed based on the values of the selected

services, they should be updated for all non-executed tasks (i.e., $A_i \in \mathcal{A} \mid es_i = ne$) taking into account the values of the already executed services (Equations from (6.1) to (6.11)). We note that when updating temporal thresholds, the start and the finish time of the already executed services are considered in the model from (6.5) to (6.10). (2) Considering the new values of thresholds, the set of pertinent services of all non-executed tasks will be updated according to Constraint (6.12). Services that no longer meet the new values of thresholds are removed. In the case where a better change occurs, some services may be added to the set of pertinent services if they satisfy the new thresholds. After that, the scores of pertinent services of all non-executed tasks will be recomputed based on Constraint (6.13) and the ranking of services is updated.

Action$_3$: *Update the optimal solution and thresholds and pertinent services of all non-executed tasks (i.e.,* CS^*, T_i^I, T_i^M *and* $\mathcal{S}_{Peri}$ $\forall A_i \in \mathcal{A} \mid es_i = ne$*)*

When global constraints are guaranteed to be violated (i.e., at least one maximum threshold is exceeded) or the service currently in execution is no more available, the optimal solution is no more feasible and should be changed. In such cases, the execution might be interrupted until the re-selection is completed. To speed up the re-selection process, we propose to select services locally for each non-executed task. The aim of the *dynamic local selection* is to avoid comparing several combinations of services, which decreases the efficiency of the selection process. Given the already executed services and the occurred violation, the main idea is to search for a service for at least one non-executed task such that all constraints are fulfilled. If a solution is found, $Action_2$ is applied, otherwise, we proceed to the *region based selection approach*. If a solution is found, $Action_2$ will be applied, otherwise, there is no solution to the selection problem. Both the *dynamic local selection* and *region based selection* approaches are detailed in Section 6.3.4.

Action$_4$: *Update thresholds and pertinent services of an interrupted task*

If a service of a task A_i currently in execution, is no more available, it should be substituted. In contrast to existing approaches that change the values of global constraints according to already executed services, we do not change the global constraints. In fact, this can not be easily applied when handling complex business structures and several QoS and temporal constraints. On the other side, changing global constraints is not always allowed. To deal with this, first, we add a temporally virtual abstract task A'_i in sequence with the task A_i with $es_i = ce$. This virtual task has the same characteristics as A_i (i.e., it has the same set of pertinent services and the same temporal constraints) and it will be used only to search for a new solution while considering the time spent by the unavailable service of the task A_i. The best ranked service in $\mathcal{S}_{Peri}$ will be considered as the selected service for the task A'_i.

- Changes in Services Currently in Execution: When a deviation occurs in a service during its execution, it should be considered. Here, we note that we assume that once a service starts execution, it can not be interrupted even if it violates the thresholds.

To maintain the efficiency of the proposed algorithm and avoid unnecessary re-selection actions, we consider that if the violation is smaller than the intermediary thresholds, it will not affect the selected solution (Case (1).a. in Table 6.2). If there is a high deviation between the estimated and the actual values (i.e., the violation exceeds the intermediary threshold), the likelihood to violate global constraints will be high. If the violated value is between the intermediary and the maximum thresholds, $Action_2$ is applied considering the new values of the service currently executed. Then, we compare the new scores of all non-executed services in CS^* (i.e., $sr'(s_v^s)\ \forall\ es_v = ne$) with those of the best selected services (i.e., $sr'(s_v^1)\ \forall\ es_v = ne$) according to the new values of thresholds when considering the violation in the executed service. If all selected services have the best scores, the violation is considered as non affecting and the selected solution will not change (Case (1).b.). Otherwise, selected services with lower scores will be substituted by the best ranked services of their corresponding tasks (Case (1).c.). In such a case, $Action_2$ will be applied another time to take into account the values of the new services.

If the violation of the service currently in execution exceeds at least one maximum threshold (i.e., $\exists 1 \leq y \leq m+2, \neg(p_y^{s_i^m}\ \text{res}\ T_{iy}^M)$), we apply $Action_3$ to update the optimal solution and search for a new feasible solution (Case (1).d.). If the service currently in execution is no more available, we apply $Action_4$ and then, we apply the same steps of $Action_3$ (Case (1).e.).

- Changes in the Environment: These changes are related to the cases when a new service is added or when a service that is not currently in execution is deleted or changes its values. In all cases, if a change occurs in candidate services of an already executed task (i.e., $es_i = ae$), it will not be considered (Cases (2).a., (3).a. and (4).a.). Hereafter, we summarize changes that should be considered as well as actions that should be triggered in each case:

- *Addition*: if a new added service does not satisfy thresholds, it will not be considered (Case (2).b.). Otherwise, if its score is less than the score of the selected service (Case (2).c.) or it is better but the selected service is currently in execution (Case (2).d.), the service will be added to the set of pertinent services and $Action_1$ is applied. Otherwise, the selected service will be substituted by the new service and then $Action_2$ is applied (Case (2).e.).

- *Deletion*: if the deleted service does not belong neither to the set of pertinent services nor to CS^*, it will not be considered (Case (3).b.). If it belongs to $\mathcal{S}_{Per}$ it will be removed (Case (3).c.), otherwise, it will be substituted by the first ranked service (Case (3).d.) and $Action_2$ is applied.

- *Modification*: if the modified service does not belong to $\mathcal{S}_{Per}$ and CS^* and does not satisfy at least one maximum threshold, it will not be considered (Case (4).b.). If the modified service belongs to $\mathcal{S}_{Per}$ and does not satisfy at least one maximum threshold, it will be removed from the set of pertinent services (Case (4).c.). Otherwise, we distinguish between two cases: (1) If the modified service does not belong to CS^*, thus, it will be considered as a new added service (Cases (4).d., (4).e. and (4).h.). (2) If the modified service belongs to CS^*, then, if it satisfies all intermediary thresholds (Case (4).f.) or it satisfies all maximum thresholds and its score is better than that of the best service (i.e., s_i^1) (Case (4).g.), it will be preserved in CS^*. Otherwise, if it satisfies maximum thresholds but its score is less than that of the best service (Case (4).i.) or if it does not satisfy at least one maximum threshold (Case (4).j.), then it will be substituted by the service s_i^1 in CS^*. In all these cases, $Action_2$ should be applied. The aim of this step is to enhance the selected solution according to the occurred modification in order to reduce the number of possible violations in the remaining tasks of the process. We note that if $s_i^o \in CS^*$, then, the change is in a task that is not yet executed (i.e., $es_i = ne$) since if $es_i = ce$, the change is considered as a change in a service currently in execution, which is treated previously.

6.3.4 Local and Region based Service Selection

As discussed in Section 6.3.3, when global constraints are violated or the service currently in execution is no more available, we proceed to the local service selection approach. In case their is no solution, a region based service selection approach is applied. These two approaches take as input the first non-executed task A_i and return the new solution if it exists. We point out, that in case of global constraint violation, the first non-executed task is the one that succeeds the task that violates the maximum threshold. In the case where the executed service is no more available, the first non-executed task is the added virtual task (i.e., A'_i). In the following, we detail each of these approaches.

TABLE 6.2: Classification of changes and re-selection actions.

		Not to be Considered	To be Considered	
			Not Affecting	**Affecting**
Changes in Services	$s_i^s \in CS^* \mid es_i = ce$		- **(1).a.** $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^I - **(1).b.** $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^M and $\exists 1 \leq y \leq m+2$ $\mid \neg(p_y^{s_i^m}$ sat $T_{iy}^I)$ and $sr'(s_v^s) \geq sr'(s_v^1), \forall es_v = ne$ $\Rightarrow CS^{*n} = CS^{*o} \setminus \{s_i^s\} \cup \{s_i^m\}$ and **Action₂**	- **(1).c.** $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^M and $\exists 1 \leq y \leq m+2 \mid$ $\neg(p_y^{s_i^m}$ sat $T_{iy}^I)$ and $\exists s_v^s \in CS^* \mid sr'(s_v^s) < sr'(s_v^1)$ and $es_v = ne$ $\Rightarrow CS^{*n} = CS^{*o} \setminus \{s_i^s, s_v^s \mid es_v = ne$ and $sr'(s_v^s) < sr'(s_v^1)\}$ $\cup \{s_i^m, s_v^1 \mid es_v = ne$ and $sr'(s_v^s) < sr'(s_v^1)\}$ and **Action₂** - **(1).d.** $\exists 1 \leq y \leq m+2, \neg(p_y^{s_i^m}$ sat $T_{iy}^M) \Rightarrow$ **Action₃** - **(1).e.** s_i^s is no more available $\Rightarrow$ **Action₄** and **Action₃**
Changes in Environment	**Addition**	- **(2).a.** $es_i = ae$ - **(2).b.** $\exists 1 \leq y \leq m+2$ $\mid \neg(p_y^{s_i^n}$ sat $T_{iy}^M)$	- **(2).c.** $\forall 1 \leq y \leq m+2, p_y^{s_i^n}$ sat T_{iy}^M and $sr(s_i^n) \leq sr(s_i^s)$ - **(2).d.** $\forall 1 \leq y \leq m+2, p_y^{s_i^n}$ sat T_{iy}^M and $sr(s_i^n) > sr(s_i)$ and $es_i = ce$ $\Rightarrow \mathcal{S}_{Peri}^n = \mathcal{S}_{Peri}^o \cup \{s_i^n\}$ and **Action₁**	- **(2).e.** $\forall 1 \leq y \leq m+2, p_y^{s_i^n}$ sat T_{iy}^M and $sr(s_i^n) > sr(s_i^s)$ and $es_i = ne$ $\Rightarrow CS^{*n} = CS^{*o} \setminus \{s_i^s\} \cup \{s_i^n\}$ and **Action₂**
	Deletion	- **(3).a.** $es_i = ae$ - **(3).b.** $s_i^d \notin \mathcal{S}_{Peri}$ and $s_i^d \notin CS^*$	- **(3).c.** $s_i^d \in \mathcal{S}_{Peri}$ $\Rightarrow \mathcal{S}_{Peri}^n = \mathcal{S}_{Peri}^o \setminus \{s_i^d\}$	- **(3).d.** $s_i^d \in CS^*$ $\Rightarrow CS^{*n} = CS^{*o} \setminus \{s_i^s\} \cup \{s_i^1\}$ and **Action₂**
	Modification	- **(4).a.** $es_i = ae$ - **(4).b.** $s_i^o \notin \mathcal{S}_{Peri}$ and $s_i^o \notin CS^*$ and $\exists 1 \leq y \leq m+2 \mid$ $\neg(p_y^{s_i^m}$ sat $T_{iy}^M)$	- **(4).c.** $s_i^o \in \mathcal{S}_{Peri}$ and $\exists 1 \leq y \leq m+2 \mid \neg(p_y^{s_i^m}$ sat $T_{iy}^M)$ $\Rightarrow \mathcal{S}_{Peri}^n = \mathcal{S}_{Peri}^o \setminus \{s_i^o\}$ - **(4).d.** $s_i^o \notin CS^*$ and $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^M and $sr(s_i^m) \leq sr(s_i^s)$ - **(4).e.** $s_i^o \notin CS^*$ and $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^M and $sr(s_i^m) > sr(s_i^s)$ and $es_i = ce$ $\Rightarrow \mathcal{S}_{Peri}^n = \mathcal{S}_{Peri}^o \cup \{s_i^m\}$ and **Action₁** if $s_i^o \notin \mathcal{S}_{Peri}$ $\Rightarrow \mathcal{S}_{Peri}^n = \mathcal{S}_{Peri}^o \setminus \{s_i^o\} \cup \{s_i^m\}$ and **Action₁** if $s_i^o \in \mathcal{S}_{Peri}$ - **(4).f.** $s_i^o \in CS^*$ and $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^I - **(4).g.** $s_i^o \in CS^*$ and $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^M and $\exists 1 \leq y \leq m+2 \mid \neg(p_y^{s_i^m}$ sat $T_{iy}^I)$ and $sr(s_i^m) \geq sr(s_i^1)$ $\Rightarrow CS^{*n} = CS^{*o} \setminus \{s_i^o\} \cup \{s_i^m\}$ and **Action₂**	- **(4).h.** $s_i^o \notin CS^*$ and $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^M and $sr(s_i^m) > sr(s_i^s)$ and $es_i = ne$ $\Rightarrow CS^{*n} = CS^{*o} \setminus \{s_i^s\} \cup \{s_i^m\}$ and **Action₂** - **(4).i.** $s_i^o \in CS^*$ and $\forall 1 \leq y \leq m+2, p_y^{s_i^m}$ sat T_{iy}^M and $\exists 1 \leq y \leq m+2 \mid \neg(p_y^{s_i^m}$ sat $T_{iy}^I)$ and $sr(s_i^m) < sr(s_i^1)$ $\Rightarrow CS^{*n} = CS^{*o} \setminus \{s_i^o\} \cup \{s_i^1\}$ and **Action₂** - **(4).j.** $s_i^o \in CS^*$ and $\exists 1 \leq y \leq m+2 \mid \neg(p_y^{s_i^m}$ sat $T_{iy}^M)$ $\Rightarrow CS^{*n} = CS^{*o} \setminus \{s_i^o\} \cup \{s_i^1\}$ and **Action₂**

6.3.4.1 Dynamic Local Service Selection Approach

When a deviation exceeds the maximum threshold or a service currently in execution is no more available, we proceed to the dynamic local service selection approach presented in Algorithm 6. First, we compute thresholds for non-executed tasks (i.e., $\forall A_v \in A_{ne} \mid A_{ne} = \{A_v \in \mathcal{A}, es_v = ne\}$) based on models presented in Section 6.3.1 while taking into account the violation occurred on the service currently in execution (lines from 4 to 5). We note that the computation of the new thresholds can be done in parallel for all tasks since it is independent form one task to another. In this step, if the service violates the estimated values, only thresholds for the violated attributes will be recomputed. If the executed service is no more available, thresholds will be recomputed for all QoS and temporal attributes. The set of pertinent services of each non-executed task is then updated by removing all services that do not meet the new thresholds and recomputing the score of each service based on the new values of thresholds (line 6). If there is at least one service in the set of pertinent services of an non-executed task A_i considering the new values of thresholds, the selected service of this task (i.e., s_i^s) will be substituted by the best ranked service (i.e., s_i^1) in the optimal solution CS^{*n} (lines from 7 to 12). In fact, since the best service satisfies maximum thresholds, its integration with the already selected services in CS^{*o} maintains the satisfaction of all global constraints. If all non-executed tasks have empty sets of pertinent services (i.e. $\mathcal{S}^n_{Perv} = \emptyset$, $\forall A_v \in A_{ne}$), we apply the region based service selection approach described in the next subsection (lines 13 and 14).

Algorithm 6 Dynamic Local Service Selection Algorithm

1: **Input:** The first non-executed task A_i
2: **Output:** The new solution of the selection problem Sol
3: $Sol = \emptyset$
4: **for** each $A_v \in A_{ne}$ **do**
5: $computeThresholds(A_v)$
6: $\mathcal{S}^n_{Perv} \leftarrow updatePertinentServices(A_v, \mathcal{S}^o_{Perv})$
7: **while** $Sol = \emptyset$ and $i \leq n$ **do**
8: **if** $\mathcal{S}^n_{Peri} \neq \emptyset$ **then**
9: $CS^{*n} = CS^{*o} \setminus \{s_i^s\} \cup \{s_i^1\}$
10: $Sol = CS^{*n}$
11: **else**
12: $i = i + 1$
13: **if** $Sol = \emptyset$ **then**
14: $Sol \leftarrow regionBasedServiceSelection()$

6.3.4.2 Region based Service Selection Approach

When there is no solution using the dynamic local selection, we propose to re-select services for a specified set of tasks. To do this, we propose an algorithm that iteratively identifies a re-selection region $\mathcal{R}$ that has to be considered. The aim is to avoid including all the set of non-executed tasks in the selection process and include only a minimum number of services that have to be replaced in the re-selection region. The main idea of this step is to increment the number of tasks included in the region until a feasible solution can be found. Figure 6.1 shows an example of re-selection regions.

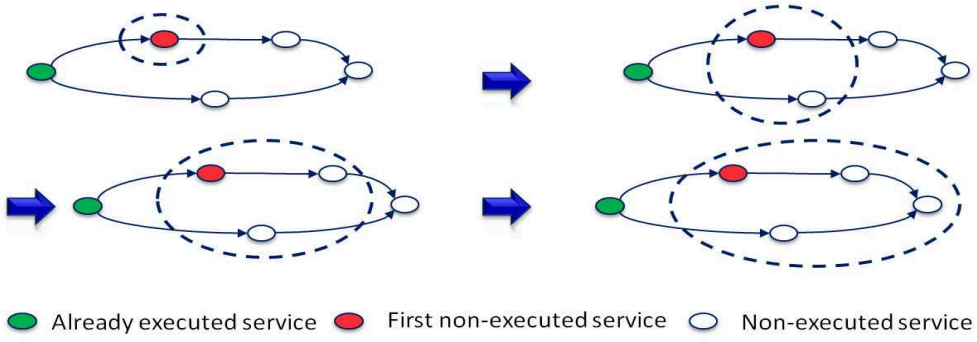

FIGURE 6.1: Re-selection regions after a violation or an unavailable service

The region based service selection approach is presented in Algorithm 7. First, we assign the first non-executed task A_i to the region set $\mathcal{R}$ (line 3). While there is no solution and the number of tasks included in $\mathcal{R}$ is less than the number of non-executed tasks, we expand the re-selection region $\mathcal{R}$ by adding the next task (i.e., A_{i+1}) (lines 4 and 5). We suppose that the tasks are ordered according to their ids, which represent the occurrence of the task in the business process. Moreover, in the case where the failed service is the last service (i.e., $A_i = A_n$), if there is no solution using the local selection approach, we conclude that there is no solution to the problem and it is not worth to apply the region based selection approach. Then, we apply the selection algorithm to the set of tasks in the specified region (line 6). The selection problem is presented as a constraint optimization problem defined in the model from (6.15) to (6.26)). In this step, only a set of pertinent services is considered for each task in order to deal with scalability issues. In case there is a solution to the constraint optimization model, all selected services for tasks in the selection region will substitute the already selected services in CS^* (i.e., $CS^{*n} = CS^{*o} \setminus \{s_i^s, \forall A_i \in \mathcal{R}\} \cup \{s_i, \forall A_i \in \mathcal{R}\}$ with s_i denotes the new selected service for A_i using the proposed model) and $Sol = CS^{*n}$ (lines from 7 to 9). In case there is no solution (i.e., $Sol = \emptyset$), we increment the counter i and repeat the same steps (lines 10 and 11). The algorithm ends if a solution to the problem is found or if all non-executed tasks are included in the region and there is no solution. In this latter case, we conclude that the problem has no feasible solution (lines from 12 to 13).

Algorithm 7 Region Based Service Selection Algorithm

Input: The first non-executed task A_i
Output: The new solution of the selection problem Sol
$\mathcal{R} = \{A_i\}$, $Sol = \emptyset$
while $Sol = \emptyset$ and $\mid \mathcal{R} \mid < \mid A_{ne} \mid$ **do**
  Add task A_{i+1} to $\mathcal{R}$
  $Sol \leftarrow$ *Re-selectServices*($\mathcal{R}$)
  if $Sol \neq \emptyset$ **then**
    $CS^{*n} = CS^{*o} \setminus \{s_i^s, \forall A_i \in \mathcal{R}\} \cup \{s_i, \forall A_i \in \mathcal{R}\}$
    $Sol = CS^{*n}$
  else
    $i = i + 1$
if $Sol = \emptyset$ **then**
  There is no solution to the selection problem

The proposed selection model is as presented in the following (Constraints from (6.15) to (6.26)). The objective function (6.15) allows selecting the best solution. To avoid considering all candidate services for the tasks in $\mathcal{R}$, only the set of pertinent services of each task before the violation will be considered (i.e., $\mathcal{S}_{Per}$). To guarantee that only one service will be selected for each task we define Constraint (6.17) where $a_{ijk} = 1$ if the service S_{ijk} is selected and 0 otherwise. In addition, to avoid searching for the corresponding global constraints for each region, which can be time consuming and can not give the best values in some cases, we apply the selection algorithm for all tasks while assigning to each task that does not belong to $\mathcal{R}$, the originally selected service before the violation. Constraints (6.18) and (6.19) allow maintaining the selected services for all tasks that do not belong to the selection region. Since all global constraints have to be satisfied when selecting the optimal solution, we add Constraint (6.20). In addition, we should ensure that the start and finish time of each non-executed task (i.e., $\forall A_i \in \mathcal{A} \mid es_i = ne$) belong to the time span of its selected service instance, which is presented in Constraints from (6.21) to (6.23). The start and the finish times of each executed task are equal to the times of its already executed service. Constraint (6.24) identifies the domains of the start and the finish time of each task in $\mathcal{R}$. The start and the finish time of an non-executed task that does not belong to the selection region should belong to the time span of its already selected service before the violation (Constraint (6.25)). Finally, Constraint (6.26) guarantees that the overall deadline is satisfied. For simplicity, in this model, we do not detail intra and inter task temporal constraints as well as structural constraints. Constraints presented in Section 5.2 can be used.

$$\text{maximize} \sum_{y=1}^{m} W_y * \frac{Q(q_y)^{max} - Q(CS, q_y)}{Q(q_y)^{max} - Q(q_y)^{min}} \tag{6.15}$$

$$Q(CS, q_y) = Agg_{A_i \in \mathcal{A}}(\sum_{S_{ij} \in \mathcal{S}_{Peri}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * Q(S_{ijk}, q_y)), \forall q_y \in \mathcal{QS} \tag{6.16}$$

$$\sum_{S_{ij} \in \mathcal{S}_{Peri}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} = 1, \forall A_i \in \mathcal{A}, a_{ijk} \in \{0, 1\} \tag{6.17}$$

$$a_{ijk} = 1, \forall A_i \notin \mathcal{R}, S_{ijk} = s_i^s \tag{6.18}$$

$$a_{ijk} = 0, \forall A_i \notin \mathcal{R}, S_{ijk} \neq s_i^s \tag{6.19}$$

$$Q(CS, q_y) \leq Q(q_y), \forall q_y \in \mathcal{QS} \tag{6.20}$$

$$\sum_{S_{ij} \in \mathcal{S}_{Peri}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * t_{S_{ijk}}^{min} \leq st_i, \forall A_i \in \mathcal{A} \mid es_i = ne \tag{6.21}$$

$$st_i \leq \sum_{S_{ij} \in \mathcal{S}_{Peri}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * (t_{S_{ijk}}^{max} - Q(S_{ijk}, dur)), \forall A_i \in \mathcal{A} \mid es_i = ne \tag{6.22}$$

$$ft_i = st_i + \sum_{S_{ij} \in \mathcal{S}_{Peri}} \sum_{T_{S_{ijk}} \in \mathcal{T}_{ij}} a_{ijk} * Q(S_{ijk}, dur), \forall A_i \in \mathcal{A} \mid es_i = ne \tag{6.23}$$

$$st_i, ft_i \in [t_{A_i}^{min}, t_{A_i}^{max}], \forall A_i \in \mathcal{R} \tag{6.24}$$

$$st_i, ft_i \in [t_{s_i^s}^{min}, t_{s_i^s}^{max}], \forall A_i \notin \mathcal{R} \mid es_i = ne \tag{6.25}$$

$$ft_n \leq deadline \tag{6.26}$$

6.4 Conclusion

In this chapter, we have proposed a dynamic service selection approach that reacts to changes as soon as these changes occur in order to repair the selected solution during execution. First, to avoid unnecessary adaptations, we compute a set of thresholds for each business task. Second, the most pertinent services for each task are identified to speed up the re-selection of services at run-time. Finally, we categorize the different changes and specify actions that have to be taken in each case. To allow the substitution of a minimal number of services without interrupting the execution of services, unless necessary, and reduce the computation time, we propose a dynamic local selection algorithm. In case, there is no solution, we apply a region based service selection to avoid the need to perform re-selection from scratch. Our approach differs from existing ones by the fact that thresholds and alternative services are updated during execution based on the values of the already executed services. Moreover, reactions to changes are made as soon as they occur in order to avoid execution interruption and increase the likelihood of finding a satisfactory solution. In addition, unlike existing work, our approach allows not only dealing with run-time deviations, but also it enables to enhance

the selected service composition while considering time-dependent QoS associated with temporal constraints. In the next chapter, we evaluate the different contributions of our thesis.

Chapter 7

Evaluation

Contents

7.1 Introduction

This chapter studies the performance of the different contributions of this thesis and presents the prototype we have developed. The different algorithms are implemented and evaluated analytically and empirically. In Section 7.2, we present our simulation tool (*TQCoS*) to generate the different test cases. The evaluation of the performance of the different contributions is detailed in Section 7.3. The time complexity has been studied and experiments based evaluations have been conducted. Finally, Section 7.4 concludes the chapter.

7.2 The TQCoS Simulation Tool

To model the business process as well as the different constraints between tasks, we adopted the work of [120]. In this work, authors propose an extension of the Business Process Modeling Notation (BPMN) standard [135] to support the specification of temporal constraints. BPMN offers a standard notation for modeling and specifying business processes. It provides a graphical notation that allows a better understanding of business models. Figure 7.1 shows a screenshot of the modeling of the business process of our motivating example (Section 1.3) using the TOPE (Time-aware inter-Organisational business Process vErification) tool [120, 136]. A description of the BPMN file of our motivating scenario is provided in Appendix B.

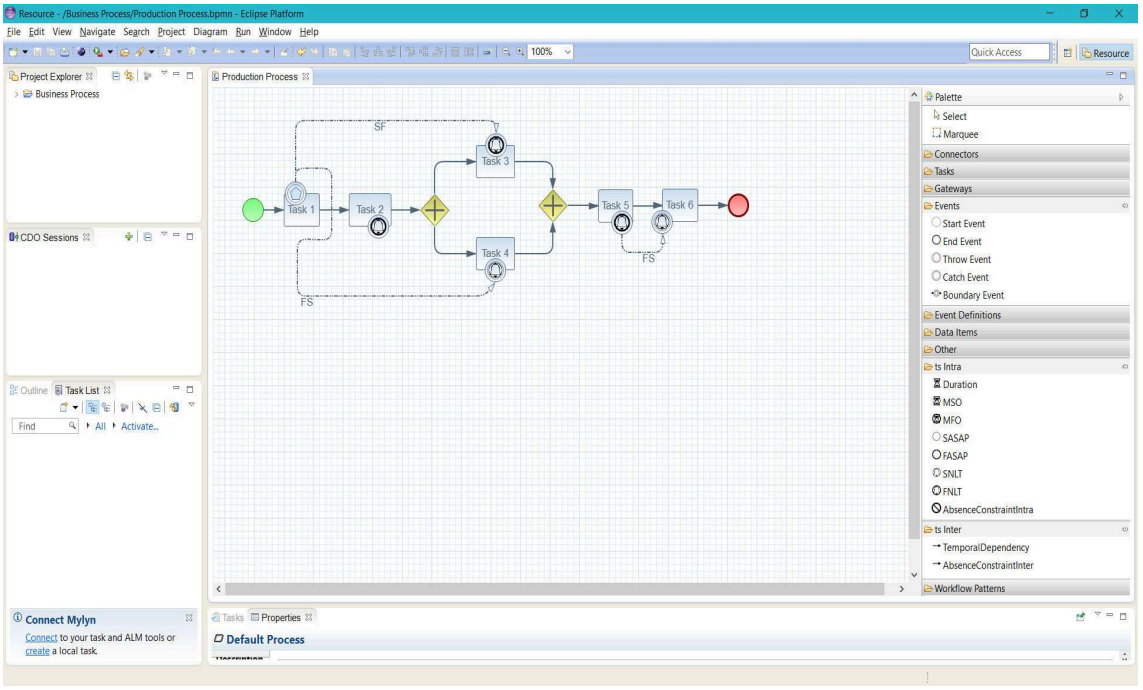

FIGURE 7.1: Modeling the business process using TOPE tool

To evaluate the different contributions of our approach and perform the different test cases, we have designed and developed a simulation tool. The graphical user interface is implemented with Java AWT and Java Swing. Hereafter, we present some screenshots that show the main steps of our approach.

Figure 7.2 depicts the first graphical interface of our tool. It enables the specification of the set of QoS attributes, the service selection approach (i.e., exact or approximate selection with or without pruning), the distribution of QoS values presented in Section 3.3.2.1 (i.e., correlated, independent or anti-correlated) and the number of services per abstract business task. The user should also specify the business process model. To

extract constraints at the business level (i.e., structural, QoS and temporal constraints) and parse the BPMN documents, we use the DOM[1] Java API for XML.

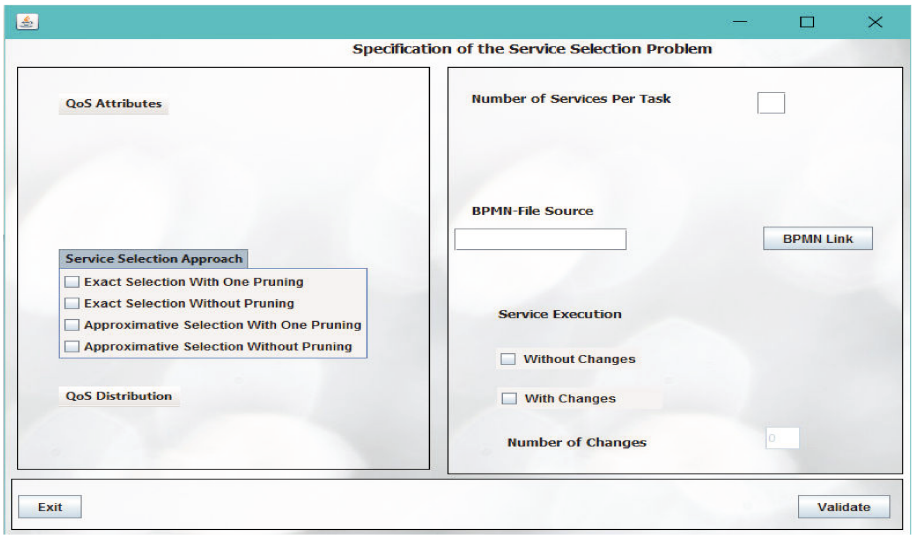

FIGURE 7.2: Setting the parameters of the selection problem

If the user wants to evaluate the dynamic selection process, the number of changes has to be stated. Our tool simulates and manages several faults and changes during execution according to specified parameters. Indeed, the user may specify the category of the change to simulate (i.e., addition, modification or deletion of services) (the right side of Figure 7.3).

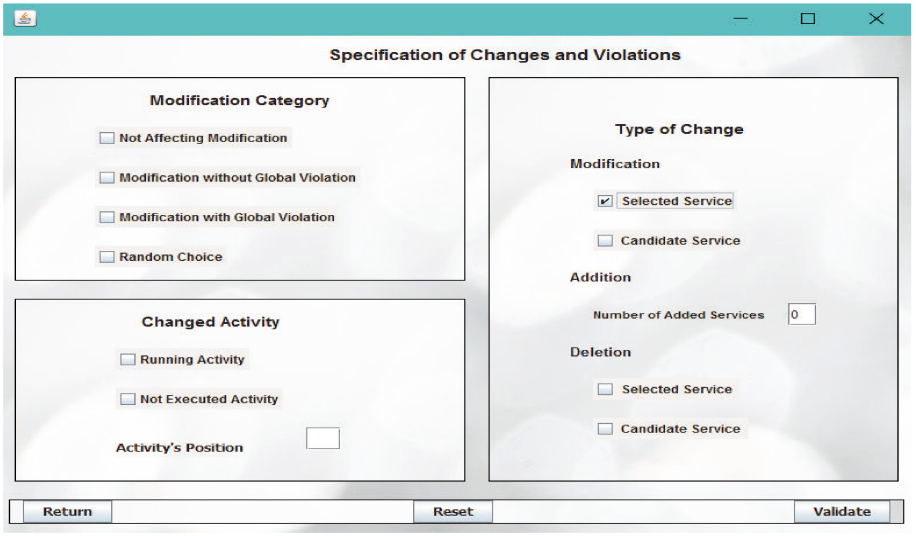

FIGURE 7.3: Setting the simulated changes and violations

In case of a modification or a deletion of a service, the user should indicate if the change occurs in a service already selected in the optimal solution or in a candidate service. When a fluctuation in a selected (under execution or not) or a candidate service

[1] https://www.w3.org/DOM/

is checked to be simulated, the user can indicate the category of the fluctuation (the top left side of Figure 7.3). This latter can be less than the intermediary threshold (i.e., not affecting modification), between the intermediary and the maximum threshold (i.e., modification without global violation) or a violation that exceeds the maximum threshold (i.e., modification with global violation). If the violation occurs on the selected service, the user may also specify if the service is currently in execution or it is not yet executed as well as the position of the violation (the bottom left side of Figure 7.3).

The final step is the visualization of the execution of tasks as well as the different changes that might occur at run-time. Figure 7.4 shows a screenshot where the execution of the process presented in Section 1.3 is depicted. This figure indicates that a change occurs in task A_3 (task in red) during the execution of the task A_1 (task in green). At the end of the execution, the optimality of the obtained solution and the re-selection time will be displayed.

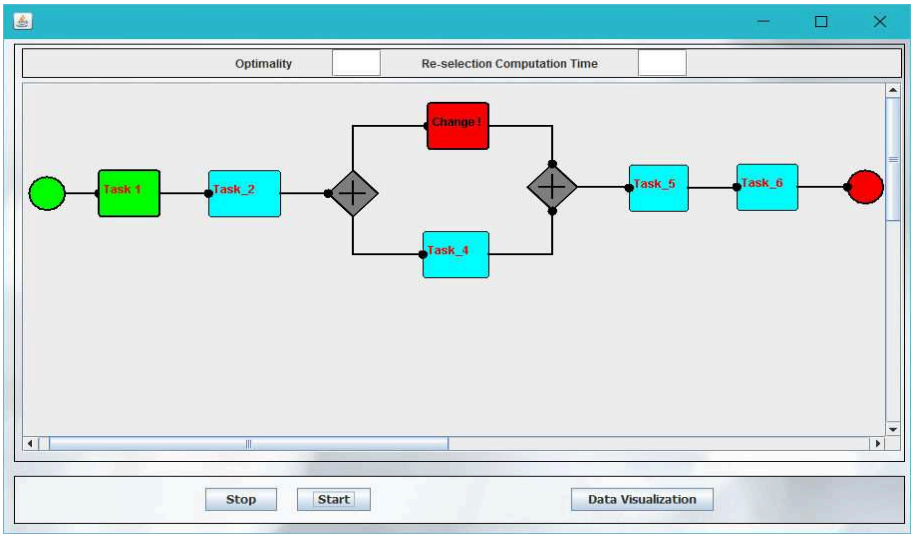

FIGURE 7.4: Visualization of the business process execution

7.3 Evaluative Study

In this section, we evaluate the performance of the different contributions of this thesis presented in the previous chapters.

7.3.1 Experiment Settings

Experiments have been performed on a laptop with a 32 bit Intel Core 2.20 GHz CPU and 4GB RAM and ubuntu 14.04 as operating system. To implement the proposed

constraint optimization models, we used the open source constraint solver Choco 2.1.5[2] and Java 1.7. Candidate services for each abstract task were generated randomly based on given minimum and maximum values for QoS attributes and time availabilities (i.e., start and finish time of each service instance). The minimum and the maximum values of QoS attributes are inspired from the QWS dataset[3]. This dataset consists of 365 Web services, each with a set of 9 quality attributes measured using commercial benchmark tools [137, 138]. Further, in this study, all constraints were randomly chosen. Time-dependent QoS attributes for each service instance were generated with a uniform distribution for a time horizon with 150 time points and distributed in the range between 1 and 150. The number of QoS and temporal constraints at the business level is fixed to 2 and 3, respectively. The structure of the business process as well as constraints at the business level (i.e., QoS and temporal constraints) are generated randomly for each test case. Different test cases have been studied and solved several times and results are averaged over several runs. In these experiments, all QoS attributes are assumed to follow an independent distribution (See Figure 3.3 in Section 3.3.2.1).

7.3.2 Evaluation of The Service Pruning Approach

We first evaluate the performance of the service pruning approach presented in Chapter 4 in terms of the computation time. In these experiments, we study only the performance of the constraint-based pruning approach. The computation time includes the time required to compute local QoS and temporal thresholds and to eliminate uninteresting services before selecting the best combination of services (See Section 4.3). It is worth noting that the computation time required to eliminate inadequate services is approximately negligible compared to the computation time of the thresholds computation process and thus, it is ignored in these experiments. Hereafter, we only present the time required to compute local thresholds.

Since local QoS and temporal thresholds (i.e., the minimum start and the maximum finish time) of each task can be computed independently from each other, all thresholds can be computed in parallel in order to reduce the computation time. This latter can be measured by the maximum time value to compute all thresholds. We note that the computation time does not change when the number of services changes. This is an obvious result since the computation of the local thresholds does not depend on the number of candidate services per task. In fact, it only depends on the domains of their QoS and temporal values (i.e., minimum and maximum values of each attribute), which can be identified at design time. Furthermore, the computation time does not depend

[2]http://www.emn.fr/z-info/choco-solver/
[3]http://www.uoguelph.ca/~qmahmoud/qws/

on the number of global constraints. This is due to the fact that local thresholds are computed in parallel for all QoS attributes. Hence, the pruning time depends only on the values of global constraints and the number of abstract tasks.

- *Performance vs the values of global constraints and the number of abstract tasks*

In Figure 7.5, we present the computation time of the pruning process in response to the values of global constraints and the number of abstract tasks. For each case, we present the average time required to compute QoS and temporal thresholds and the computation time of the pruning process (i.e., the maximum time required to measure all thresholds).

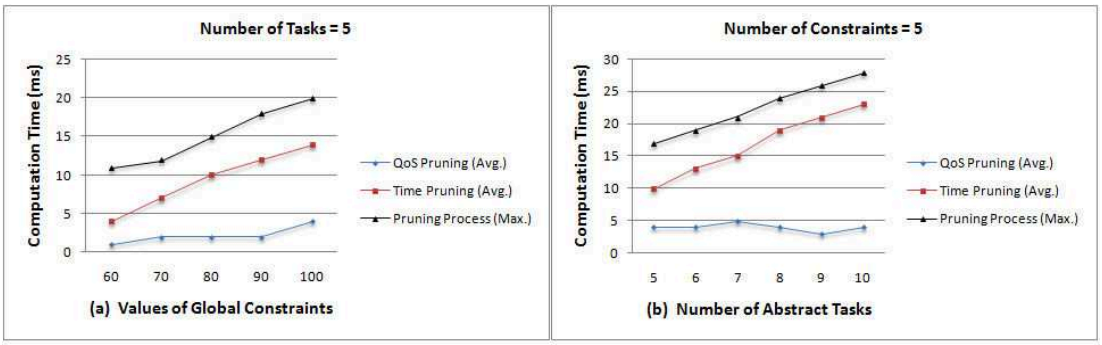

FIGURE 7.5: Evaluation of the computation time of the pruning process

First, in Figure 7.5(a), the values of global constraints vary from 60 to 100 and the number of abstract tasks is fixed to 5. The computation time increases when the values of global constraints increase. This is explained by the fact that the number of possible values of local thresholds becomes higher. Second, Figure 7.5(b) presents the computation time when the number of business tasks varies from 5 to 10. The number of global constraints is equal to 5 and their values vary from 80 to 120. Results indicate that the computation time increases very slowly when the number of tasks increases because the number of decision variables increases as well in this case.

- *Discussion*

Results displayed in Figure 7.5 show that in all cases the time required to compute local thresholds is dominated by the computation of local temporal thresholds. This is due to the fact that the number of decision variables is larger than that used to compute QoS thresholds. Experiments show that in both cases, the time required to compute local thresholds is very small (28 ms in the worst case) while dealing with large domains for all variables. This allows our approach to be used in large problems where QoS and

temporal values may be set in very large intervals with a negligible overhead that does not affect the efficiency of the selection process.

7.3.3 Evaluation of The Exact Service Selection Approach

In what follows, we evaluate the computation time of the optimal service selection approach presented in Section 5.2 with respect to the number of services, global constraints and abstract business tasks in two cases: *with pruning* and *without pruning*. We note that in the first case (i.e., with pruning), the considered computation time is equal to the sum of the computation time of both the pruning and the selection steps.

- *Performance vs the number of services*

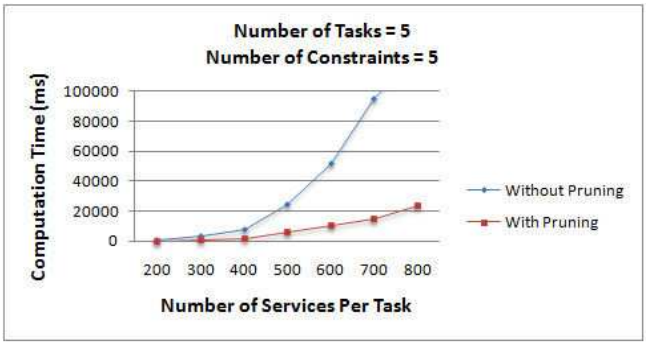

FIGURE 7.6: Evaluation of the computation time of the exact selection approach with respect to the number of candidate services per task

First, experiments were conducted in relation to the number of candidate service instances per task that varies from 200 to 800 with 5 business tasks and 5 global constraints. The results provided in Figure 7.6 indicate that applying the pruning process significantly outperforms the basic algorithm. Although the computation time of the basic algorithm is close to the time of our approach when the number of services is small (around 200 service instances), the computation time of our approach increases very slowly as compared to the basic one by increasing the number of candidate services per task.

- *Performance vs the number of global constraints*

Figure 7.7(a) shows the computation time of the selection algorithms when the number of global constraints varies between 5 and 10 and the number of tasks is fixed to 5 with 200 candidate service instances for each task. As before, while the computation time of the basic algorithm significantly increases due to the fact of the growing number of

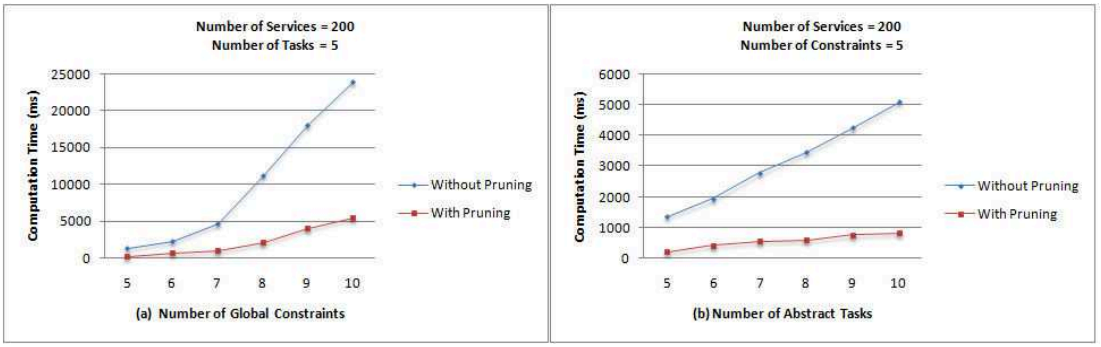

FIGURE 7.7: Evaluation of the computation time of the exact selection approach with respect to the number of global constraints and business tasks

optimal instances that should be compared, the time of our algorithm increases very slowly and leads to better performance even when the number of constraints increases. This is an expected behavior since by considering several global constraints, the number of feasible solutions decreases. Hence, more services are likely to violate one or more constraints and thus, they can be pruned prior to the selection process.

- *Performance vs the number of abstract tasks*

To further evaluate the performance of our pruning approach, Figure 7.7(b) depicts experimental results based on the computation time with respect to the number of business tasks. The number of candidate service instances for each task is fixed to 200 and the number of global constraints is fixed to 5. Business tasks depend on each other with several complex structures including different composition patterns. Again, results are the same as previously and the optimal solution can be found rather fast when applying our pruning approach.

- *Discussion*

In all the test cases, the results show a considerable gain in performance of the selection process when applying the pruning approach, which scales better than the traditionally selection algorithm where all candidate services are considered. The performance of our approach is significant particularly in the context of complex selection problems where the number of candidate services, tasks and constraints can be large.

7.3.4 Evaluation of The Approximate Service Selection Approach

In this section, we evaluate the performance of the heuristic based service selection approach, presented in Section 5.3, by studying its time complexity and analyzing experimental results based on simulation studies.

7.3.4.1 Complexity Evaluation

The size of the selection problem is defined by three main factors: the number of abstract business tasks n, the number of candidate service instances per task c and the number of global constraints m. The performance of the heuristic service selection approach is affected by the time complexity of the different steps.

- *Service clustering*: This phase includes the specification of the centroids and the classification of services. First, for each task, the number of QoS levels for each quality attribute is K. Hence, the complexity of the first step is $O(n * K * m)$ with K is the number of centroids. Since the classification of services can be done in parallel for all tasks, the complexity of this step is $O(c * K * m * it)$ with it is the number of iterations of the clustering step. Hence, the complexity of the service clustering phase is $O(n * K * m + c * K * m * it)$.

- *Local QoS constraints specification*: The number of decision variables of the constraint optimization model to find the best local QoS constraints (i.e., the best centroids) is $n * K$. Consequently, the complexity of this phase is $O(2^{n*K})$.

- *Deadline decomposition*: This step relies on the use of four variables (earliest start, earliest finish, latest start, and latest finish time) and it can be applied in parallel for all tasks. The complexity of this step is $O(4 * no)$ with no is the operation number of the equation resolution.

- *Local selection*: The best service for each task can be selected locally with a complexity of $O(c)$.

Based on the above analysis, the overall complexity of our approach is: $O(n * K * m + c * K * m * it + 2^{n*K} + 4 * no + c) = O(2^{n*K})$. Therefore, the time complexity of the proposed approach is dominated by the local QoS constraints specification phase whose complexity does not depend on the number of candidate services, which enhances its scalability. Let us now compare our approach with existing approaches [16, 66, 67, 72]. If $K < h$ with h is the number of promising services per task in [72], our approach achieves better performance than the existing work, which does not deal with time-dependent QoS and temporal constraints. Furthermore, if $K << m * d$ with d is the number of QoS levels in [16] and $K << c$, we can ensure that the size of our model is much smaller than those of the models proposed in [16], [67], [66] even though these approaches consider only static QoS values.

7.3.4.2 Experimental Results

To evaluate the effectiveness of the heuristic selection approach, we study its performance in terms of the computation time, the optimality and the success rate with respect to the number of centroids. We compare the results of the different tests in two cases: *heuristic approach without pruning (HWO)* and *heuristic approach with pruning (HW).*

- *Performance in terms of the computation time*

First, we analyze the computation time of the different test cases with respect to the number of candidate services per task and the number of abstract business tasks (Figure 7.8).

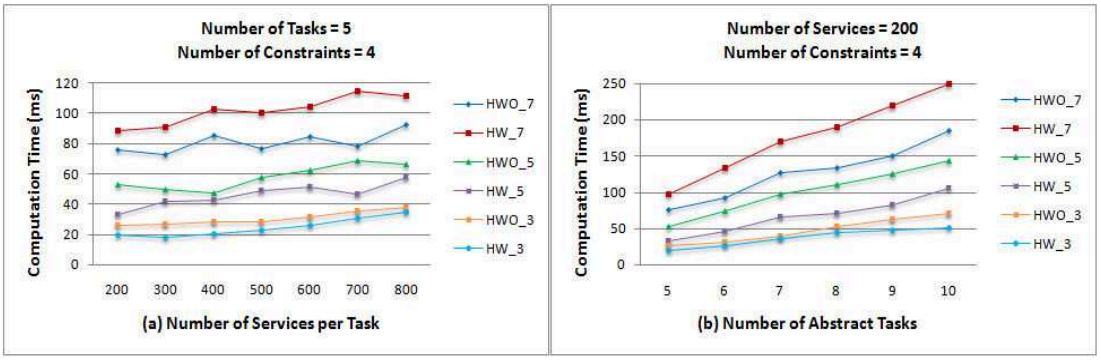

FIGURE 7.8: Evaluation of the computation time of the approximate selection approach

Figure 7.8(a) shows the computation time of the different approaches. The number of business tasks and global constraints is fixed to 5 and 4, respectively. The number of centroids varies between 3 and 7. The number of clustering iterations is fixed to 3. The results indicate that the computation time of the heuristic approach is very small comparing to the optimal approach with and without pruning presented in Figure 7.6(a). In addition, the computation time is smaller when applying our pruning techniques before the selection process in most cases. This is explained by the fact that the number of services is reduced when applying our pruning approach. Moreover, the domains of the QoS values of the centroids are more smaller due to the consideration of local thresholds. Hence, a near-to-optimal solution can be found more quickly. As depicted in Figure 7.8(a), we can observe also that the computation time of the heuristic approach without pruning is a little bit better than the time of the heuristic approach with pruning when the number of centroids increases. This is due to the fact that in this latter, the combinations of centroids that should be compared are more relevant and thus, more computation time is required to compare all possible combinations. Experiments also show that although the computation time increases exponentially when the number of services rises in the exact approaches, it is relatively stable in our heuristic approach.

However, the computation time increases slowly when the number of centroids increases. This is obvious since the time of the local QoS constraints specification phase does not depend on the number of services per task but mainly depends on the number of centroids.

Figure 7.8(b) shows the computation time of the selection approaches when the number of tasks varies between 5 and 10 and the number of global constraints is fixed to 4 with 200 candidate services for each task. Again, experiments show that our heuristic approach outperforms the exact approaches. In fact, the computation time of our approach increases slowly along with the number of tasks especially when the number of centroids is very small.

- *Performance in terms of the optimality*

In addition to the computation time, we evaluate the efficiency of the heuristic selection approach in terms of its optimality with respect to the number of services and abstract tasks (Figure 7.9). The optimality evaluates the quality of the solution obtained by the different selection approaches. It is computed by comparing the overall utility value of the solution found by each selection approach (U_{sel}) (See equation 3.5) to the utility value of the optimal solution obtained by the optimal selection approach (U_{opt}) i.e.,:

$$optimality = U_{sel}/U_{opt}$$

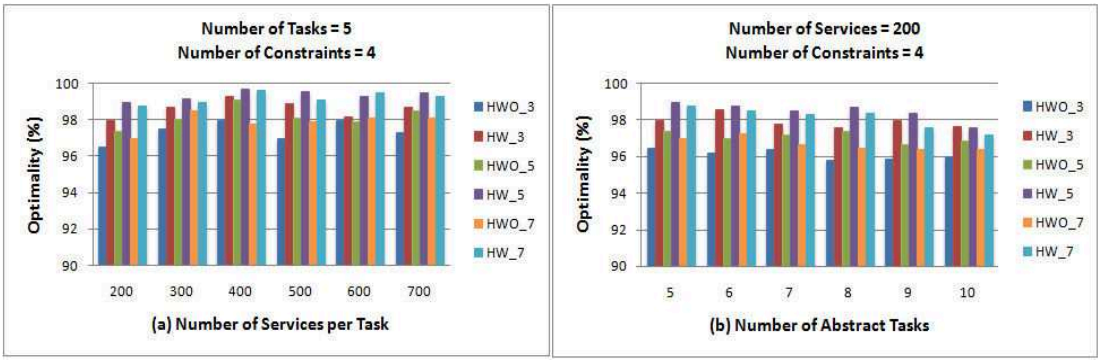

FIGURE 7.9: Evaluation of the optimality of the approximate selection approach

Figure 7.9(a) shows that by increasing the number of centroids, the optimality of the selected solution increases as well. However, if the number of centroids increases more than necessary, the optimality does not change in most cases and can decrease in some cases. This is due to the fact that when the number of centroids is higher than necessary, the classification of services is no more efficient since candidate services will be distributed in a very large set of QoS levels, which may affect the utility values of the selected

centroids. Moreover, the optimality of the heuristic approach with pruning is better than that of the approach without pruning. This is due to the fact that the computed centroids are more accurate after the pruning process. Additionally, Figure 7.9(b) also shows that our approach can produce a satisfactory optimality (i.e., more than 96 %) in most cases. Again, applying our pruning techniques before the selection process allows reaching better results since the selection algorithm is applied on the set of the most significant services. Moreover, in contrast to the selection approach without pruning, which can reach a reasonable optimality when increasing the number of centroids, the heuristic approach with pruning can obtain better optimality even when the number of centroids is very small. This is because when considering all candidate services, several inadequate instances can affect the values and the utilities of the centroids.

- *Performance in terms of the success rate*

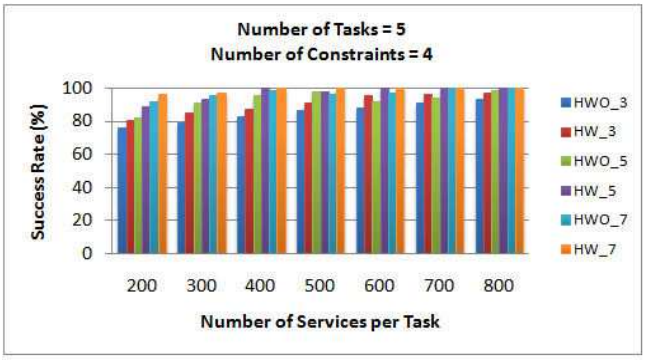

FIGURE 7.10: Evaluation of the success rate of the approximate selection approach

The success rate presents the percentage of scenarios where a feasible solution is found compared to the total number of scenarios where a solution exists:

$$success = Nb_{succ-scen}/Nb_{total-scen}$$

The results displayed in Figure 7.10 indicate that the success rate of the heuristic approach increases along with the number of candidate services per task. Figure 7.10 also shows that the heuristic approach after the pruning phase achieves better success rate than the heuristic approach that does not consider preselected candidate services. The reason of this behavior is that the pruning process allows eliminating uninteresting services and consider only the most important ones. Hence, selected centroids are more relevant and thus, it is more likely to find a feasible solution.

- *Discussion*

We can conclude that in all test cases, our heuristic approach can achieve a satisfactory optimality with a very small computation time. However, there is a significant trade-off between the number of the centroids and the computation time, the optimality and the success rate of the proposed approach. In fact, when the number of centroids increases, the computation time of the selection process increases and the optimality and the success rate increase as well and inversely. Hence, the choice of the number of centroids is of great importance. It should not be very high to reduce the computation time and find an efficient classification of services. Additionally, it should not be very small so that we can find the most adequate centroids and thus, a close-to-optimal solution. Results also show the advantage of the pruning process before the selection phase. In fact, although the pruning step can increase the computation time, this latter is still very negligible as compared to the time of the selection approach without pruning. This is mainly due to the fact that local thresholds of the pruning process can be computed in parallel for all tasks and for each QoS attribute in contrast to the local QoS constraints that can not be computed in parallel and that requires more computation time when no pruning techniques are applied. Thus, the pruning process allows: (i) reducing the computation time of the selection process and (ii) achieving more accurate results with a very small number of centroids.

7.3.5 Evaluation of The Dynamic Service Selection Approach

In this section, we report the evaluation results of our dynamic service selection approach presented in Chapter 6. First, we analyze the time complexity of the selection algorithms. Then, we present the analysis of experimental results focusing on the efficiency in terms of the computation time, the optimality and the success rate.

7.3.5.1 Complexity Evaluation

The size of the dynamic selection problem is defined by two main factors: the number of abstract tasks considered in the dynamic selection phase and the number of candidate services considered for each abstract task. When a change or a violation occurs without affecting the satisfaction of global constraints (i.e., when the actions $Action_1$ and $Action_2$ are applied (See Section 6.3.3.2)), the enhancement process is usually performed in parallel to the execution of the services and thus, it does not affect the complexity of the proposed approach. In addition, the complexity of the action $Action_4$ is equal to $O(s)$ with s is the number of pertinent services per task. Hence, in this subsection, we analyze the complexity of the local and the region based service selection approaches presented in Section 6.3.4 (i.e., $Action_3$).

- *Local service selection*: the time complexity of the local selection phase depends on the complexity of the following three steps: (i) computing local thresholds, (ii) updating the set of pertinent services and (iii) locally selecting new services. The computation of local thresholds is independent of the number of services per tasks and the number of business tasks since QoS and temporal thresholds for all tasks can be computed in parallel. Thus, the complexity of this step is $O(no)$ with no denotes the operation number of equation resolution of the proposed models (from (6.1) to (6.10)). (ii) The updating of the set of pertinent services can be done in parallel for all non-executed tasks. It consists on browsing the set of pertinent services of each task, selecting and computing the scores of those that satisfy the new thresholds, which has a complexity of $O(s)$ and the ranking of the new set of pertinent services, which has a complexity of $O(np)$ in the best case and $O(np * log(np))$ in the worst case with np is the number of the new pertinent services. (iii) The local selection of the best services has a complexity of $O(1)$ in the best case (i.e., the first non-executed task has a non empty set of services) and a complexity of $O(e)$ in the worst case (i.e., only the last non-executed task has a non empty set of services) with e denotes the number of non-executed tasks (i.e., $e =\mid A_{ne} \mid$). Thus, the complexity of the local selection phase is equal to $O(no + s + np * log(np) + e) = O(np * log(np))$ in the worst case.

- *Region based service selection*: the complexity of this step is equal to $O(2^{r*s})$ with r is the number of tasks in the selection region $\mathcal{R}$. More specifically, it is equal to $O(2^{2*s})$ in the best case (i.e., in the case where a solution to the selection problem is found when only two non-executed tasks are included in $\mathcal{R}$) and is equal to $O(2^{e*s})$ in the worst case (i.e., all non-executed tasks are included in $\mathcal{R}$).

In conclusion, the complexity of our approach is equal to $O(np*log(np))$ in the worst case when a solution is found based on local selection and is equal to $O(2^{e*s})$ in the worst case when a solution is found using the region based selection approach. Consequently, our approach achieves better performance than existing dynamic service selection approaches [67, 101] that re-select services from scratch each time a violation is observed and thus, they have a complexity of $O(2^{e*c})$ in all cases with c denotes the number of all candidate services per task.

7.3.5.2 Experimental Results

To evaluate the effectiveness of our approach, we study its performance in terms of the computation time, the optimality and the success rate. We compare our approach that enhances the selected solution progressively during execution with the global dynamic selection approach (global approach for short) in which changes are considered only when

global constraints are violated and a selection from scratch for all non-executed tasks is applied [67]. Here, we suppose that the time required to identify the category of change is very negligible compared to the selection time and thus, it is ignored. Experiments were conducted using a complex business process composed of 10 abstract tasks. The structure of the process is generated randomly. The number of candidate services and QoS constraints is fixed to 500 and 5, respectively.

- *Performance of the dynamic selection process in response to a violation in a selected service*

First, we compare the computation time (i.e., the time required to find a solution) of the two approaches in response to a deletion or a violation of a selected but not yet executed service (Figure 7.11(a)) and a deletion or a violation of a service currently in execution (Figure 7.11(b)). In these two cases, we suppose that the violation exceeds the maximum thresholds and thus, a re-selection is mandatory to guarantee the satisfaction of global constraints.

In Figure 7.11(a), we assume that the violation or the deletion of the selected service occurs after two tasks from the one being executed (e.g., while executing the first task, the violation occurs on the third task). Experiments reveal that our approach outperforms the basic approach in all cases. In fact, the computation time of our approach is negligible compared to that of the global approach. This is mainly due to the fact that in our approach, a solution can be found by simply replacing the failed service by the first pertinent service. However, the global approach delays the reaction to changes until the execution of the failed service. Experiments also show that the global approach takes more time when the change is observed at an early stage of the execution due to fact that there are several non-executed services and thus, more possible combinations have to be compared. We note that whereas, the interruption time (i.e., the time in which the execution is stopped to find a feasible solution) is equal to the computation time for the global approach (since this latter takes selection actions only after executing the failed service), our approach does not cause the interruption of the execution since the selection actions are taken as soon as the change occurs in parallel to the execution.

In Figure 7.11(b), the change can not be observed before the execution of the erroneous service and thus, the interruption of execution is required. In such a case, the computation time is equal to the interruption time for both approaches. Again, our approach has a very small interruption time even when the change occurs at earlier stages. In fact, in some cases, our approach can find a solution based on local selection. Moreover, selecting services only for a specific region while considering a set of pertinent services for each task allows enhancing the performance of our approach since it decreases the

number of possible combinations that have to be compared. The computation time of both approaches decreases when the position of the executed task get closer to the end of the process. This is due to the fact that the number of tasks that are considered in the selection process becomes smaller and the number of possible solutions decreases.

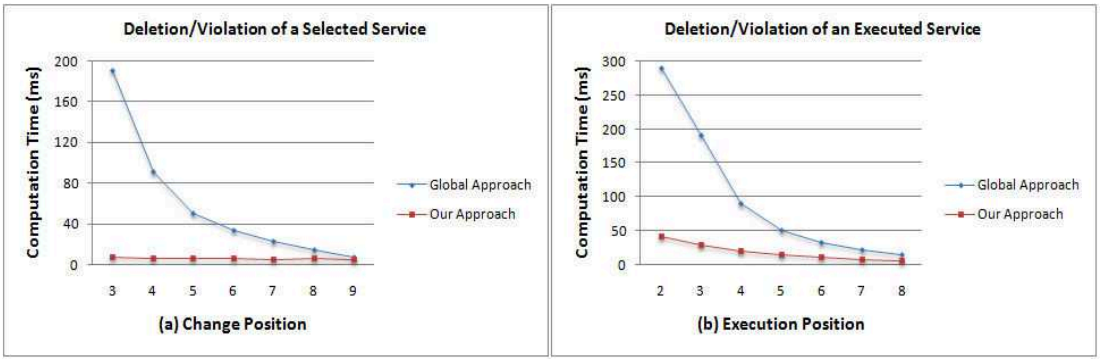

FIGURE 7.11: Evaluation of the computation time of the dynamic selection approach in response to a service violation

To evaluate the gain of utility of our approach, we compare its optimality with that of the global approach (Figures 7.12(a) and 7.12(b)) considering both cases as previously (i.e., a deletion or a violation of a selected but not yet executed service and a deletion or a violation of a service currently in execution). The optimality of each approach is computed by dividing the utility value of the obtained solution after changes by the utility value of the primary combination of services. As can be seen in Figure 7.12(a), our approach reaches better optimality in all cases since it relies on early reaction to changes. Results also show that although the optimality of the global approach decreases by increasing the change position, the optimality of our approach is almost stable. Figure 7.12(b) indicates that in the case where the erroneous behavior is observed in the service currently in execution, our approach can produce a satisfactory optimality. Here, the global approach has better optimality since it considers all possible combinations of services in contrast to our approach that can find a solution by local selection.

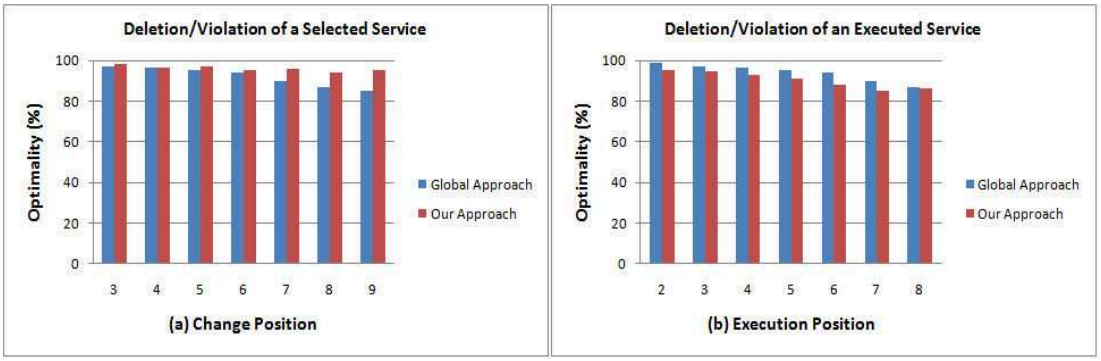

FIGURE 7.12: Evaluation of the optimality of the dynamic selection approach in response to a service violation

FIGURE 7.13: Evaluation of the success rate of the dynamic selection approach in response to a service violation

To further illustrate the performance of our approach, we compare the success rate achieved by the different test cases (Figure 7.13).

Figure 7.13 indicates that our approach has better success rate since it reacts to changes as soon as they occur, which increases the likelihood to find a solution. Whereas, when using the global approach, it might be the case where no solution is found after a global violation. This is due to the fact of delaying the reaction to changes after the execution of the failed service. We note that if the erroneous behavior occurs in the executed service, the two approaches have the same success rate since our approach acts as the global approach in the worst case.

- *Performance of the enhancement process in response to a random change*

To further evaluate the efficiency of our approach, we study its performance in response to the number of changes, which are added randomly at run-time for tasks that are currently in execution or not yet executed. These changes include the addition, the modification and the deletion of non-selected candidate services as well as violations in the selected services. All violations in selected services are assumed to be less than the maximum thresholds. The positions of changes and violations are generated randomly in all cases.

First, Figure 7.14 depicts the performance of our approach in terms of the computation time required to consider the different violations and changes and in terms of the interruption time. The computation time includes the time required to deal with all changes. Figure 7.14(a) shows that although the computation time of the global approach is almost negligible when the number of changes is very small, it increases by increasing the number of changes. In contrast, the computation time of our approach (i.e., the time required to enhance the solution) is very small in most cases. This is explained by the fact that the global approach usually does not react to changes if the global constraints

are not violated. If a global violation occurs, the execution is interrupted until a solution is found and usually a re-selection for all the non-executed portion of the business process is required, which can be time consuming. However, our approach takes adaptation actions and enhances the selected solution during execution as soon as a change occurs even before the occurrence of a global violation. Figure 7.14(b) indicates that the interruption time is basically equal to 0 using our approach since the enhancement process is done in parallel to the execution of services without the need to interrupt the execution. In contrast, since the global approach delays the treatment of the different changes and violations until a global violation occurs, it requires to interrupt the execution until a solution is found.

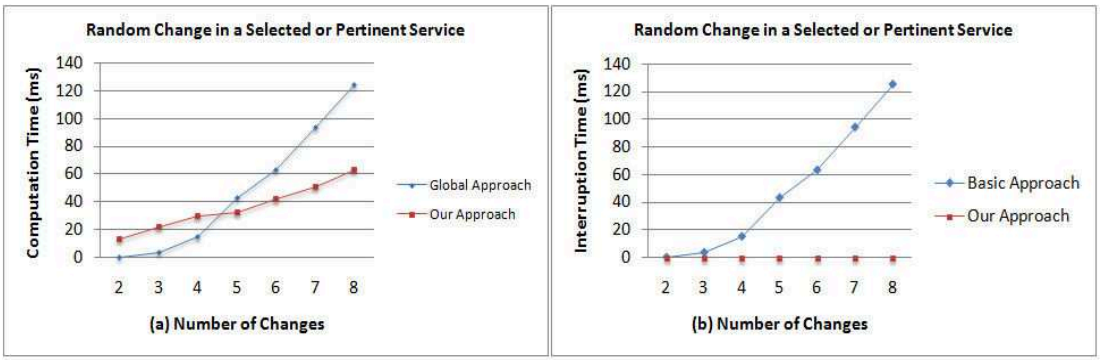

FIGURE 7.14: Evaluation of the computation time of the dynamic selection approach in response to environment changes

Second, we study the optimality and the success rate of our approach compared to those of the global approach according to the number of changes. The optimality of each approach is computed by dividing the utility value of the obtained solution after changes by the utility value of the primary combination of services. Figure 7.15(a) shows that although the optimality of the global approach decreases by increasing the number of changes and violations, the optimality of our approach is almost stable in most cases. This is mainly due to the fact that our approach allows enhancing the selected solution and taking re-selection actions during execution before executing the failed service. Moreover, in contrast to the global approach, our approach considers not only the non-executed successors of the failed service but also its non-executed predecessors, which allows obtaining better solutions. Figure 7.15(b) indicates that our approach has better success rate since it proactively reacts to changes, which increases the likelihood to find a solution. Whereas, when using the global approach, it might be the case where no solution is found after a global violation.

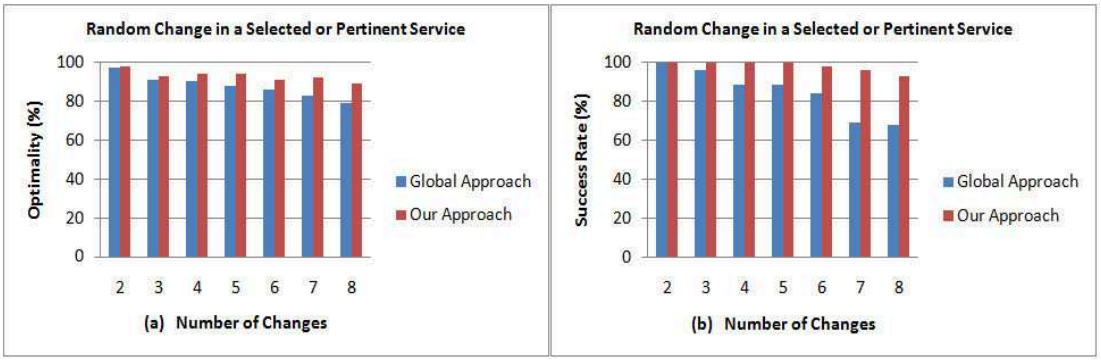

FIGURE 7.15: Evaluation of the optimality and the success rate of the dynamic selection approach in response to environment changes

- *Discussion*

The results show that our approach achieves quick and efficient selection in response to environment changes and service failures and violations. In fact, it can find a close-to-optimal solution in dynamic and uncertain environments with a small overhead. Our approach has better performance than the global approach since it allows enhancing the selected solution and taking preventive actions before a violation of global constraints may occur. In addition, when global constraints are violated or a service currently in execution is no more available, our local and region based service selection approaches have better performance than that of the global selection approach. Moreover, our approach can find a solution in most cases with a very small computation time and a satisfactory optimality. This is due to the fact that the local selection approach involves only the re-selection of one service as opposed to the global selection approach that requires the re-selection of all non-executed services in order to select the optimal solution. Moreover, the global approach can not find a feasible solution in some cases, which is mainly due to the accumulation of several violations.

7.4 Conclusion

In this chapter, we have demonstrated the usefulness of the proposed approach through experimental results. We have studied the performance of all contributions from analytical and empirical points of view. Experiments are performed based on several test cases to evaluate the proposed contributions in response to the number of business tasks, constraints and candidate services per task. The performance of our algorithms are evaluated based on the required computation time, the optimality and the success rate. The evaluation results demonstrate the effectiveness of the proposed approach compared to

existing approaches while still dealing with both small and large selection problems and considering QoS and temporal properties.

Chapter 8

Conclusions and Future Work

Contents

In service oriented systems, the development of complex applications relies on the composition of elementary services. Usually, besides functional requirements, the composition must fulfill also non functional requirements, namely QoS and temporal constraints. In real-world scenarios, QoS offers may not be static and can change over time. Moreover, several dependencies may exist between involved services namely structural and temporal dependencies. On the other side, services can evolve in highly dynamic environments, which might cause QoS values fluctuations during the execution.

Selecting the adequate combination (composition) of services and ensuring that the selected solution remains satisfactory during execution in response to QoS fluctuations and environment changes while considering temporal properties represents a major challenge in service oriented systems and, despite several researches, still not adequately treated. Towards addressing these challenges, this thesis has provided several novel contributions. These contributions are summarized in Section 8.1, followed by a list of possible short and long term future extensions of our research work in Section 8.2.

8.1 Contributions and Research Summary

In this section, we summarize the different contributions of our research work. Our thesis has four main contributions:

- Specification of a constraint-based service selection model

- A dominance and constraint-based service pruning approach

- Exact and approximate service selection approaches at design time

- A proactive dynamic service selection approach at run-time

8.1.1 Specification of a constraint-based service selection model

In this thesis, we have proposed a rich constraint model which caters for several constraints and dependencies that may be defined at the business level namely structural, QoS and temporal constraints. Four main structural patterns have been considered: sequence, parallel, choice and loop. Based on these patterns, we have proposed a detailed presentation of the aggregation function while handling several categories of QoS attributes.

In contrast to existing work, our model allows for specifying not only global QoS constraints but also intra and inter-task QoS and temporal constraints at the business level to get control over the execution of the process and to cater for temporal dependencies between services. Moreover, despite their impact on the quality of the delivered services, time-dependent QoS have been ignored by most of the service selection approaches. To deal with this limitation, we have provided a presentation of timed service instances, which may deliver different QoS values with respect to time. These instances have been considered in the different steps of our approach.

8.1.2 A dominance and constraint-based service pruning approach

Selecting the adequate solution while achieving time efficiency and guaranteeing a satisfactory optimality is of great importance especially in heavily constrained service selection problems. To ensure the time efficiency of the selection process, we have presented dominance and constraint-based service pruning techniques. The proposed techniques aim to reduce the number of candidate services prior to the selection process based on both QoS and temporal properties while maintaining the optimal solution. A set of constraint optimization models have been proposed to compute the different local

QoS and temporal thresholds required to remove inadequate services. The proposed models are generalized to accommodate challenging problems where several constraints and dependencies are identified between services. The computation of local thresholds is independent from one task to another and can be applied in parallel for all QoS attributes to increase the efficiency of the pruning phase. Through the pruning process, improvement techniques have been provided to enhance the service selection problem in case there is no feasible solution.

The proposed service pruning approach differs from existing ones by the fact that it does not pre-select relevant services based on only QoS values but also based on temporal properties while considering several dependencies between services and this, without removing the optimal solution. Moreover, our pruning strategies enable identifying the source of failure in case there is no solution to the selection problem so that improvement strategies can be carried out to find a feasible solution. Additionally, the pruning process allows enhancing the performance of the service selection process with a very small overhead since the set of local thresholds can be computed in parallel and only a set of the most accurate services is considered during the selection phase.

8.1.3 Exact and approximate service selection approaches at design time

In this thesis, we have proposed exact and approximate service selection approaches. The exact approach is adequate in small selection problems to select the optimal solution (i.e., the best combination of services). The approximate approach is suitable for large selection problems to select a close-to-optimal solution.

To avoid greedy decomposition of global constraints and consider correlations between the values of each candidate service, we have adopted clustering techniques to identify a set of clusters for each abstract task. The most relevant centroids are then selected (one centroid for each task) such that all constraints are satisfied. The quality values of the selected centroids will be further considered as local QoS constraints. Moreover, in contrast to existing approaches that handle static QoS values, in addition to the global QoS decomposition, we have proposed a decomposition technique to identify local temporal constraints that have to be fulfilled during local selection. The proposed heuristic approach allows for reaching a satisfactory optimality in most cases with a very small computation time.

8.1.4 A proactive dynamic service selection approach at run-time

To deal with uncertainties and dynamism of service oriented systems, appropriate adaptation actions have to be taken in response to possible violations and environment changes. Most of existing approaches consider only violations in the selected services and do not cater for changes that might occur on the environment during service execution. In this thesis, we have proposed a dynamic service selection approach to adapt the selected combination of services at run-time in order to support the development of reliable business processes.

Different from existing approaches that take recovery actions only for corrective purposes, our dynamic selection approach allows handling several changes and enhancing the selected solution during execution in response to the values of the already executed services and the current state of the environment while handling QoS and temporal properties. The main advantages of our approach can be summarized as follows:

- *Classification of changes and re-selection thresholds*

  To avoid unnecessary re-selection, we have identified intermediary thresholds for each abstract task. Moreover, a set of maximum thresholds that must not be exceeded during execution has been identified so that if at least one of these thresholds is exceeded, re-selection actions have to be taken. To define the importance of each change during execution, we have provided a classification of the different changes. Thus, according to the type of the change, actions are triggered either in parallel to the execution or an interruption of the execution of services is required.

- *Limited set of candidate services*

  A set of pertinent services is identified for each task to speed up the re-selection of services during execution. Pertinent services selected for each abstract task are identified such that all constraints are verified and they are ranked according to their distance to the maximum execution thresholds. This allows a reliable service re-selection during execution that guarantees the fulfillment of all constraints. Unlike existing approaches that consider static set of backup solutions, in our approach, we have proposed to update the set of pertinent services according to the new information in the selected solution without interrupting the execution of services.

- *Limited set of abstract tasks*

  To enhance the efficiency of the proposed approach, first, a local selection is applied to substitute the failed service by simply selecting the first ranked pertinent service.

If no solution is found, we identify a re-selection region to avoid re-selecting services for all non-executed tasks.

- *Re-selection in parallel to execution*

  In our approach, adaptation actions are triggered as soon as a change occurs in order to minimize the execution interruption and prevent possible violations due to the accumulation of several violations. Moreover, if the adaptation actions triggered in response to the different changes and violations are not completed before the complete execution of the service currently in execution, they will be continued in parallel to the execution of the next service. This allows maximizing the likelihood of finding a feasible solution while guaranteeing a good optimality of the selected solution since re-selection actions are triggered before reaching the erroneous service where no solution can be found or the solution would be of lower quality.

8.2 Future Directions

Besides the aforementioned contributions, several perspectives are still to be investigated towards improving and extending our work. In what follows, we list short and long-term perspectives to present potential enhancements of our work and future research directions in service composition selection.

8.2.1 Short-term Perspectives

In order to improve the proposed approach, five short-term perspectives can be investigated:

- *Implementing the dominance-based pruning*

  In this thesis, the dominance-based pruning phase presented in Section 4.2 was not implemented. As a first step towards enhancing the proposed work, we plan to implement this phase to further enhance the performance of the pruning process. To do so, multi-objective optimization algorithms can be applied to identify all non-dominated candidate service instances for each abstract task (e.g., SPEA [45], NSGA [47], PAES [49]).

- *Considering different distributions and patterns of QoS values*

  In this thesis, all experiments were conducted assuming an independent distribution of quality values for the different candidate service instances. The third

perspective consists in studying the performance of the different contributions of our work through experimental studies while taking into account different distributions and patterns of QoS values (See Section 3.3.2).

- *Considering real-data*

  In this thesis, all experiments were conducted on simulation studies and randomly-generated data. Although these experiments have demonstrated the effectiveness of the different proposed contributions and algorithms, it is important to support the obtained results by demonstrating the usefulness of the different contributions in practical scenarios through real-data.

- *Developing the improvement process*

  In section 4.4, we have presented pruning-based improvement process to try to find a solution when the selection process cannot be fulfilled. The second short-term perspective is the development and the validation of this improvement process. This can be achieved by adopting appropriate negotiation strategies guided by our improvement techniques.

- *Prioritizing changes and violations*

  Finally, we are interested in extending our dynamic service selection approach to handle the case where multiple changes might occur at the same time and multiple solutions, which can be conflicting, can be found. In this case, we shall propose to classify the different changes and violations according to their importance during execution.

8.2.2 Long-term Perspectives

In what follows, we list a number of possible long-term perspectives that we believe are interesting to our work.

- *Inter-service correlations*

  In our work, we have concentrated on structural and inter-task temporal dependencies between services and we suppose that services are independent with respect to quality attributes. However, further correlations can also be considered in the selection problem. In fact, services may behave differently in different composition scenarios with respect to quality attributes [119, 139–141]. This can be due to for example business correlations among competitive enterprises. For example, a service provider might offer discount policies if a service of another provider is selected to implement one task in the same service composition. Thus, quality values

of a service may depend on the other services selected in the composition. Inter-service QoS dependencies is a very interesting feature that have to be considered in service selection problems.

Another category of correlations among services is the compatibility correlation, which indicates whether two or more services can be selected in the same composition [139, 142]. QoS and compatibility dependencies awareness raises additional challenges to the service selection problems. Tackling with such dependencies in the different steps of our approach combined with structural and temporal dependencies is a very interesting future direction of our research work.

- *Granularity heterogeneity*

  In our work, we assume that the composite service is modeled using a business process with elementary activities. That means, for each activity, we try to select a concrete service. This assumption may prevent the discovery and the selection of pertinent services that do not meet abstract tasks identified in the business process. For example, a provider that offers manufacturing and assembling facilities as one service (i.e., it must perform the two functionalities together and not only one) can not be selected if these two tasks are modeled separately in the composition. Similarly, if these two services are modeled as one abstract task in the composition, individual manufacturing and assembling services will be neglected.

  Thus, a coarse-grained service can better implement two abstract tasks by combining their functionalities rather than two individual finer-grained candidate services and inversely [17, 70, 143]. Addressing services available at different granularity levels when pruning and selecting services is a target area for future work.

- *Behavioral adaptation*

  The improvement techniques proposed in this thesis, in case no solution can be found after the pruning process (See Section 4.2) are mainly concentrated in the improvements required by users and service providers. However, in some cases, to further relax the selection problem, improvements and adaptations can be made at the business level (e.g., by modifying some intra or inter-task constraints or by adapting the structure of the business process). Moreover, in our work, we assume that the structure of the service composition is fixed at design time and that during execution, only dynamic service selection is considered in case of violations or environment changes. Behavioral adaptation can be another alternative to face deviations at run-time. In case of failure, the execution of the composition of services according to another alternative behavior can be considered [99, 104].

Alternative behaviors can be obtained by changing the structure of the composition. Thus, an adaptive system that is capable to modify the process structures on the fly is another important future direction of our work.

Appendix A

Proofs of Theorems and Lemmas

A.1 Proof of Theorem 4.1

Proof. Consider a service combination CS' that is derived from a service combination $CS = \{s_1, ..., s_i, ..., s_n\}$ that satisfies all constraints by substituting s_i by s_j with s_j is better than s_i for at least one attribute and it is better or equal to s_i for all other attributes.

- Suppose that s_j is better than s_i in one QoS attribute and it is better or equal in other parameters. Thus, since QoS aggregation functions are monotone, a better value in one QoS produces a better aggregated value (and thus, a better utility value) and maintains the satisfaction of all structural and QoS constraints. In addition, since s_j is better or equal to s_i in its temporal properties, all temporal constraints are fulfilled. In fact, a constraint that is satisfied by a time interval $\mathcal{T}_1$, will be satisfied by a larger time interval $\mathcal{T}_2$.
- Suppose now that s_j is better than s_i in its time interval and has the same or better QoS values than s_i. In this case, a combination CS' that has the same services than CS while substituting s_i by s_j, has the same or a better utility value and satisfies all temporal constraints since a combination derived from a larger time interval fulfills all constraints as a combination derived from a smaller time interval.

□

A.2 Proof of Lemma 4.1

Proof. Suppose that after the dominance pruning process there is a service s_i in a service class S_i that belongs to the optimal solution CS^* but it does not belong to the set of skyline services. Thus, according to the dominance definition presented previously, there is another service $s_j \in S_i$ that dominates s_i (i.e., s_j is better than or equal to s_i in all QoS and temporal parameters and is strictly better in at least one parameter). Suppose that s_j is better than s_i in one QoS or temporal parameter and it is better or equal to s_i in other attributes, thus, if we consider a service combination CS' that is derived from CS^* by substituting s_i by s_j, this combination also satisfies business constraints and has a better or the same utility value according to Theorem 4.1. Consequently, CS' is better than CS^* and thus, it should be selected as the optimal solution, which implies a contradiction. □

A.3 Proof of Lemma 4.2

Proof. We prove Lemma 4.2 by reduction to absurdity. Assume that there exists an optimal solution $CS^* = \{s_1, ..., s_i, ..., s_n\}$ that has the best utility value and satisfies all the constraints but has not been selected. Thus, CS^* has at least one service $s_i \in \mathcal{S}_i$ that is eliminated after the constraint-based pruning process. Hence, the service s_i is pruned either because of its QoS attributes or its time interval.

- First, suppose that the service s_i is pruned due to its QoS attributes, i.e., there is at least one QoS parameter $q_y \in \mathcal{QS}$ s.t., $Q(s_i, q_y) > Q_{LT}(A_i, q_y)$. Since $s_i \in CS^*$, (1) the aggregation of the quality value of this service for the attribute q_y (i.e., $Q(s_i, q_y)$) with the quality values of the other services in CS^* satisfies the global QoS constraint $Q(q_y)$ and (2) all intra and inter-task constraints are satisfied. These two conditions are respected if we consider the combination of services composed of the service s_i and the services that have the minimum quality values for the attribute q_y of all the remaining tasks according to Theorem 4.1. Hence, s_i has a better quality value than the local QoS threshold $Q_{LT}(A_i, q_y)$ and the aggregation of its QoS value with all the minimum QoS values of the other tasks satisfies the global constraint $Q(q_y)$ such that all business constraints are fulfilled. Consequently, $Q(s_i, q_y)$ has to be selected as the local threshold of the task A_i for the quality attribute q_y so, $Q(s_i, q_y) \leq Q_{LT}(A_i, q_y)$, which is a contradictory result.
- Suppose now that the service s_i is pruned based on its time interval so either of the following two cases is satisfied:

- $[t^{min}_{S_{ijk}}, t^{max}_{S_{ijk}}] \cap [est_i, lft_i] = \emptyset$. Let's start by the first case (i.e., $t^{min}_{S_{ijk}} > lft_i$). Given that s_i is a candidate service in the optimal solution, (1) the aggregation of its values with the values of the remaining services of the combination satisfies the global execution duration constraint and (2) all intra and inter-task temporal constraints are fulfilled. From Theorem 4.1., we can conclude that these two conditions are also fulfilled if we select the service s_i with other services that offer the minimum execution duration values. Hence, the largest time interval of the task A_i should be $[est_i, ft_i]$ with $ft_i \in [t^{min}_{S_{ijk}}, t^{max}_{S_{ijk}}]$. Thus, the latest finish time has to be greater than ft_i (i.e., $lft_i \geq ft_i$) and so, $lft_i \geq t^{min}_{S_{ijk}}$, which leads to a contradiction. In the same way, we can demonstrate the second case (i.e., $t^{max}_{S_{ijk}} < est_i$).
- $[t^{min}_{S_{ijk}}, t^{max}_{S_{ijk}}] \cap [est_i, lft_i] = [X, Y]$ and $Y - X < Q(s_i, dur)$. Here, three cases arise: (a) $t^{min}_{S_{ijk}} \geq est_i$ and $t^{max}_{S_{ijk}} \leq lft_i$. In this case, $t^{max}_{S_{ijk}} - t^{min}_{S_{ijk}} < Q(s_i, dur)$, which is impossible. (b) $est_i \leq t^{min}_{S_{ijk}} \leq lft_i$ and $t^{max}_{S_{ijk}} \geq lft_i$. As previously, we can demonstrate that in this case the largest time interval of the task A_i should be $[est_i, ft_i]$ with $ft_i \geq t^{min}_{S_{ijk}} + Q(s_i, dur)$. Hence, $ft_i \geq lft_i$ and thus, it should be considered as the latest finish time for the task A_i, which is not the case. In the same way, we can demonstrate the latest case (c) $t^{min}_{S_{ijk}} \leq est_i$ and $est_i \leq t^{max}_{S_{ijk}} \leq lft_i$.

We conclude that a service that can be part of the optimal solution can not be pruned by our constraint-based pruning techniques. □

A.4 Proof of Lemma 5.1

Proof. First, since all selected services fulfill local QoS constraints, global QoS constraints will be satisfied according to (5.21). In other hand, suppose that the selected service of the first task (i.e., $A_i \in \mathcal{A}$) meets Constraint (5.36). Consequently, in the best case (i.e., its start time is equal to est^m_i and it has the minimum duration value), its finish time is equal to eft^m_i. In the worst case, its finish time is equal to lft^M_i according to Constraint (5.36). Therefore, the service that will implement the successor task (i.e., A_v with $A_i \in \mathcal{P}d(A_v)$) can verify $est^m_v \leq st_v \leq lst^M_v$ according to (5.28) and (5.29). Hence, in the worst case (i.e., the duration of the selected service is equal to the duration of the selected centroid for the task A_v), this service can start in its allowed time span and it also satisfies local temporal constraints. We note that in all cases, intra and inter-task temporal constraints and the overall deadline are satisfied since they are considered in the deadline decomposition phase (Constraints from (5.30) to (5.33)).

Moreover, the maximum duration values of the selected services guarantee the overall deadline according to (5.22). □

Appendix B

Description of The BPMN File

The structure of the BPMN file is as following.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn2:definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL" xmlns:bpmndi="http:
    //www.omg.org/spec/BPMN/20100524/DI" xmlns:dc="http://www.omg.org/spec/DD
    /20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI" xmlns:tc="
    http://org.eclipse.bpmn2.modeler.examples.ts" id="Definitions_1"
    targetNamespace="http://org.eclipse.bpmn2.modeler.examples.ts">
  <bpmn2:process id="Production_Process" name="Production Process">
    <bpmn2:startEvent id="StartEvent_1">
      <bpmn2:outgoing>tc:SequenceFlow_2</bpmn2:outgoing>
    </bpmn2:startEvent>
    <bpmn2:sequenceFlow id="SequenceFlow_2" sourceRef="StartEvent_1" targetRef="
    Task_1"/>
    <bpmn2:task id="Task_2" name="Task 2">
      <bpmn2:incoming>tc:SequenceFlow_3</bpmn2:incoming>
      <bpmn2:outgoing>tc:SequenceFlow_4</bpmn2:outgoing>
    </bpmn2:task>
    <bpmn2:sequenceFlow id="SequenceFlow_4" sourceRef="Task_2" targetRef="
    ParallelGateway_1"/>
    <bpmn2:parallelGateway id="ParallelGateway_1">
      <bpmn2:incoming>tc:SequenceFlow_4</bpmn2:incoming>
      <bpmn2:outgoing>tc:SequenceFlow_5</bpmn2:outgoing>
      <bpmn2:outgoing>tc:SequenceFlow_6</bpmn2:outgoing>
    </bpmn2:parallelGateway>
    <bpmn2:sequenceFlow id="SequenceFlow_5" sourceRef="ParallelGateway_1"
    targetRef="Task_3"/>
    <bpmn2:sequenceFlow id="SequenceFlow_6" sourceRef="ParallelGateway_1"
    targetRef="Task_4"/>
    <bpmn2:task id="Task_3" name="Task 3">
      <bpmn2:incoming>tc:SequenceFlow_5</bpmn2:incoming>
      <bpmn2:outgoing>tc:SequenceFlow_7</bpmn2:outgoing>
    </bpmn2:task>
    <bpmn2:sequenceFlow id="SequenceFlow_7" sourceRef="Task_3" targetRef="
    ParallelGateway_2"/>
    <bpmn2:task id="Task_4" name="Task 4">
```

```
    <bpmn2:incoming>tc:SequenceFlow_6</bpmn2:incoming>
    <bpmn2:outgoing>tc:SequenceFlow_8</bpmn2:outgoing>
  </bpmn2:task>
  <bpmn2:sequenceFlow id="SequenceFlow_8" sourceRef="Task_4" targetRef="
  ParallelGateway_2" />
  <bpmn2:parallelGateway id="ParallelGateway_2">
    <bpmn2:incoming>tc:SequenceFlow_7</bpmn2:incoming>
    <bpmn2:incoming>tc:SequenceFlow_8</bpmn2:incoming>
    <bpmn2:outgoing>tc:SequenceFlow_9</bpmn2:outgoing>
  </bpmn2:parallelGateway>
  <bpmn2:sequenceFlow id="SequenceFlow_9" sourceRef="ParallelGateway_2"
  targetRef="Task_5" />
  <bpmn2:task id="Task_5" name="Task 5">
    <bpmn2:incoming>tc:SequenceFlow_9</bpmn2:incoming>
    <bpmn2:outgoing>tc:SequenceFlow_10</bpmn2:outgoing>
  </bpmn2:task>
  <bpmn2:sequenceFlow id="SequenceFlow_10" sourceRef="Task_5" targetRef="Task_6
  " />
  <bpmn2:task id="Task_6" name="Task 6">
    <bpmn2:incoming>tc:SequenceFlow_10</bpmn2:incoming>
    <bpmn2:outgoing>tc:SequenceFlow_11</bpmn2:outgoing>
  </bpmn2:task>
  <bpmn2:sequenceFlow id="SequenceFlow_11" sourceRef="Task_6" targetRef="
  EndEvent_2" />
  <bpmn2:task id="Task_1" name="Task 1">
    <bpmn2:incoming>tc:SequenceFlow_2</bpmn2:incoming>
    <bpmn2:outgoing>tc:SequenceFlow_3</bpmn2:outgoing>
  </bpmn2:task>
  <bpmn2:sequenceFlow id="SequenceFlow_3" sourceRef="Task_1" targetRef="Task_2"
  />
  <bpmn2:endEvent id="EndEvent_2">
    <bpmn2:incoming>tc:SequenceFlow_11</bpmn2:incoming>
  </bpmn2:endEvent>
  <bpmn2:boundaryEvent id="BoundaryEvent_3" name="" attachedToRef="tc:Task_1">
    <bpmn2:eventDefinition xsi:type="tc:SNLT" id="SNLT_1" tc:type="SNLT" value=
  "0" />
    <bpmn2:eventDefinition xsi:type="tc:FNLT" id="FNLT_5" tc:type="FNLT" value=
  "0" />
  </bpmn2:boundaryEvent>
  <bpmn2:boundaryEvent id="BoundaryEvent_4" name="" attachedToRef="tc:Task_3">
    <bpmn2:eventDefinition xsi:type="tc:FNLT" id="FNLT_3" tc:type="FNLT" value=
  "0" />
  </bpmn2:boundaryEvent>
  <bpmn2:flowElement xsi:type="tc:TemporalDependency" id="TemporalDependency_3"
   name="SF" fromValue="1" toValue="12" sourceRef="BoundaryEvent_3" targetRef="
  BoundaryEvent_4" />
  <bpmn2:boundaryEvent id="BoundaryEvent_5" name="" attachedToRef="tc:Task_4">
    <bpmn2:eventDefinition xsi:type="tc:SNLT" id="SNLT_2" tc:type="SNLT" value=
  "0" />
  </bpmn2:boundaryEvent>
  <bpmn2:flowElement xsi:type="tc:TemporalDependency" id="TemporalDependency_4"
   name="FS" fromValue="0" toValue="3" sourceRef="BoundaryEvent_3" targetRef="
  BoundaryEvent_5" />
  <bpmn2:boundaryEvent id="BoundaryEvent_6" name="" attachedToRef="tc:Task_5">
```

```
      <bpmn2:eventDefinition xsi:type="tc:FNLT" id="FNLT_6" tc:type="FNLT" value=
    "0"/>
    </bpmn2:boundaryEvent>
    <bpmn2:boundaryEvent id="BoundaryEvent_7" name="" attachedToRef="tc:Task_6">
      <bpmn2:eventDefinition xsi:type="tc:SNLT" id="SNLT_3" tc:type="SNLT" value=
    "0"/>
    </bpmn2:boundaryEvent>
    <bpmn2:flowElement xsi:type="tc:TemporalDependency" id="TemporalDependency_5"
     name="FS" fromValue="1" toValue="2" sourceRef="BoundaryEvent_6" targetRef="
    BoundaryEvent_7"/>
    <bpmn2:boundaryEvent id="BoundaryEvent_9" name="FNLT" attachedToRef="
    tc:Task_2">
      <bpmn2:eventDefinition xsi:type="tc:FNLT" id="FNLT_7" tc:type="FNLT" value=
    "15"/>
    </bpmn2:boundaryEvent>
  </bpmn2:process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_1" name="Production Process">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="process_4">
      <bpmndi:BPMNShape id="BPMNShape_1" bpmnElement="StartEvent_1">
        <dc:Bounds height="36.0" width="36.0" x="30.0" y="117.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_ParallelGateway_1" bpmnElement="
    ParallelGateway_1">
        <dc:Bounds height="50.0" width="50.0" x="330.0" y="110.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_ParallelGateway_2" bpmnElement="
    ParallelGateway_2">
        <dc:Bounds height="50.0" width="50.0" x="530.0" y="103.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_Task_2" bpmnElement="Task_2">
        <dc:Bounds height="50.0" width="72.0" x="210.0" y="110.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_Task_3" bpmnElement="Task_3">
        <dc:Bounds height="50.0" width="66.0" x="425.0" y="37.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_Task_4" bpmnElement="Task_4">
        <dc:Bounds height="50.0" width="66.0" x="425.0" y="176.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_Task_5" bpmnElement="Task_5">
        <dc:Bounds height="50.0" width="59.0" x="632.0" y="104.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_Task_6" bpmnElement="Task_6">
        <dc:Bounds height="50.0" width="61.0" x="740.0" y="103.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_EndEvent_1" bpmnElement="EndEvent_2">
        <dc:Bounds height="36.0" width="36.0" x="851.0" y="110.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_Task_1" bpmnElement="Task_1">
        <dc:Bounds height="50.0" width="61.0" x="100.0" y="110.0"/>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="BPMNShape_BoundaryEvent_3" bpmnElement="
    BoundaryEvent_3">
        <dc:Bounds height="36.0" width="36.0" x="102.0" y="92.0"/>
      </bpmndi:BPMNShape>
```

```
  <bpmndi:BPMNShape id="BPMNShape_BoundaryEvent_5" bpmnElement="
BoundaryEvent_5">
    <dc:Bounds height="36.0" width="36.0" x="440.0" y="208.0" />
  </bpmndi:BPMNShape>
  <bpmndi:BPMNShape id="BPMNShape_BoundaryEvent_4" bpmnElement="
BoundaryEvent_4">
    <dc:Bounds height="36.0" width="36.0" x="441.0" y="19.0" />
  </bpmndi:BPMNShape>
  <bpmndi:BPMNShape id="BPMNShape_BoundaryEvent_6" bpmnElement="
BoundaryEvent_6">
    <dc:Bounds height="36.0" width="36.0" x="654.0" y="136.0" />
  </bpmndi:BPMNShape>
  <bpmndi:BPMNShape id="BPMNShape_BoundaryEvent_7" bpmnElement="
BoundaryEvent_7">
    <dc:Bounds height="36.0" width="36.0" x="722.0" y="135.0" />
  </bpmndi:BPMNShape>
  <bpmndi:BPMNShape id="BPMNShape_BoundaryEvent_9" bpmnElement="
BoundaryEvent_9">
    <dc:Bounds height="36.0" width="36.0" x="242.0" y="142.0" />
  </bpmndi:BPMNShape>
  <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_2" bpmnElement="SequenceFlow_2"
sourceElement="BPMNShape_1" targetElement="BPMNShape_Task_1">
    <di:waypoint xsi:type="dc:Point" x="66.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="84.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="84.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="100.0" y="135.0" />
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_3" bpmnElement="SequenceFlow_3"
sourceElement="BPMNShape_Task_1" targetElement="BPMNShape_Task_2">
    <di:waypoint xsi:type="dc:Point" x="161.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="190.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="190.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="210.0" y="135.0" />
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_4" bpmnElement="SequenceFlow_4"
sourceElement="BPMNShape_Task_2" targetElement="BPMNShape_ParallelGateway_1">
    <di:waypoint xsi:type="dc:Point" x="282.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="326.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="326.0" y="135.0" />
    <di:waypoint xsi:type="dc:Point" x="330.0" y="135.0" />
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_5" bpmnElement="SequenceFlow_5"
sourceElement="BPMNShape_ParallelGateway_1" targetElement="BPMNShape_Task_3">
    <di:waypoint xsi:type="dc:Point" x="355.0" y="110.0" />
    <di:waypoint xsi:type="dc:Point" x="355.0" y="62.0" />
    <di:waypoint xsi:type="dc:Point" x="425.0" y="62.0" />
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_6" bpmnElement="SequenceFlow_6"
sourceElement="BPMNShape_ParallelGateway_1" targetElement="BPMNShape_Task_4">
    <di:waypoint xsi:type="dc:Point" x="355.0" y="161.0" />
    <di:waypoint xsi:type="dc:Point" x="355.0" y="201.0" />
    <di:waypoint xsi:type="dc:Point" x="425.0" y="201.0" />
  </bpmndi:BPMNEdge>
  <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_7" bpmnElement="SequenceFlow_7"
sourceElement="BPMNShape_Task_3" targetElement="BPMNShape_ParallelGateway_2">
```

```
      <di:waypoint xsi:type="dc:Point" x="491.0" y="62.0" />
      <di:waypoint xsi:type="dc:Point" x="555.0" y="62.0" />
      <di:waypoint xsi:type="dc:Point" x="555.0" y="103.0" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_8" bpmnElement="SequenceFlow_8"
sourceElement="BPMNShape_Task_4" targetElement="BPMNShape_ParallelGateway_2">
      <di:waypoint xsi:type="dc:Point" x="491.0" y="201.0" />
      <di:waypoint xsi:type="dc:Point" x="555.0" y="201.0" />
      <di:waypoint xsi:type="dc:Point" x="555.0" y="154.0" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_9" bpmnElement="SequenceFlow_9"
sourceElement="BPMNShape_ParallelGateway_2" targetElement="BPMNShape_Task_5">
      <di:waypoint xsi:type="dc:Point" x="581.0" y="128.0" />
      <di:waypoint xsi:type="dc:Point" x="630.0" y="128.0" />
      <di:waypoint xsi:type="dc:Point" x="630.0" y="129.0" />
      <di:waypoint xsi:type="dc:Point" x="632.0" y="129.0" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_10" bpmnElement="SequenceFlow_10
" sourceElement="BPMNShape_Task_5" targetElement="BPMNShape_Task_6">
      <di:waypoint xsi:type="dc:Point" x="691.0" y="129.0" />
      <di:waypoint xsi:type="dc:Point" x="740.0" y="129.0" />
      <di:waypoint xsi:type="dc:Point" x="740.0" y="128.0" />
      <di:waypoint xsi:type="dc:Point" x="740.0" y="128.0" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_11" bpmnElement="SequenceFlow_11
" sourceElement="BPMNShape_Task_6" targetElement="BPMNShape_EndEvent_1">
      <di:waypoint xsi:type="dc:Point" x="801.0" y="128.0" />
      <di:waypoint xsi:type="dc:Point" x="839.0" y="128.0" />
      <di:waypoint xsi:type="dc:Point" x="839.0" y="128.0" />
      <di:waypoint xsi:type="dc:Point" x="851.0" y="128.0" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_TemporalDependency_3" bpmnElement="
TemporalDependency_3" sourceElement="BPMNShape_BoundaryEvent_3" targetElement
="BPMNShape_BoundaryEvent_4">
      <di:waypoint xsi:type="dc:Point" x="120.0" y="92.0" />
      <di:waypoint xsi:type="dc:Point" x="120.0" y="0.0" />
      <di:waypoint xsi:type="dc:Point" x="459.0" y="0.0" />
      <di:waypoint xsi:type="dc:Point" x="459.0" y="20.0" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_TemporalDependency_4" bpmnElement="
TemporalDependency_4" sourceElement="BPMNShape_BoundaryEvent_3" targetElement
="BPMNShape_BoundaryEvent_5">
      <di:waypoint xsi:type="dc:Point" x="120.0" y="92.0" />
      <di:waypoint xsi:type="dc:Point" x="120.0" y="72.0" />
      <di:waypoint xsi:type="dc:Point" x="181.0" y="72.0" />
      <di:waypoint xsi:type="dc:Point" x="181.0" y="180.0" />
      <di:waypoint xsi:type="dc:Point" x="123.0" y="179.0" />
      <di:waypoint xsi:type="dc:Point" x="123.0" y="265.0" />
      <di:waypoint xsi:type="dc:Point" x="440.0" y="265.0" />
      <di:waypoint xsi:type="dc:Point" x="440.0" y="245.0" />
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_TemporalDependency_5" bpmnElement="
TemporalDependency_5" sourceElement="BPMNShape_BoundaryEvent_6" targetElement
="BPMNShape_BoundaryEvent_7">
      <di:waypoint xsi:type="dc:Point" x="672.0" y="172.0" />
```

```
        <di:waypoint xsi:type="dc:Point" x="672.0" y="199.0"/>
        <di:waypoint xsi:type="dc:Point" x="741.0" y="199.0"/>
        <di:waypoint xsi:type="dc:Point" x="741.0" y="172.0"/>
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</bpmn2:definitions>
```

Listing B.1: Example of a BPMN File Description

Appendix C

Temporal-Aware Template and Offer using WS-Agreement*

Figure C.2 shows an example of temporal-aware agreement and offer using WS-Agreement* proposed in [129]. WS-Agreement* allows describing local and global validity time periods. It also allows describing periodical disjoint and non-disjoint intervals. Period definitions are presented in Figure C.1.

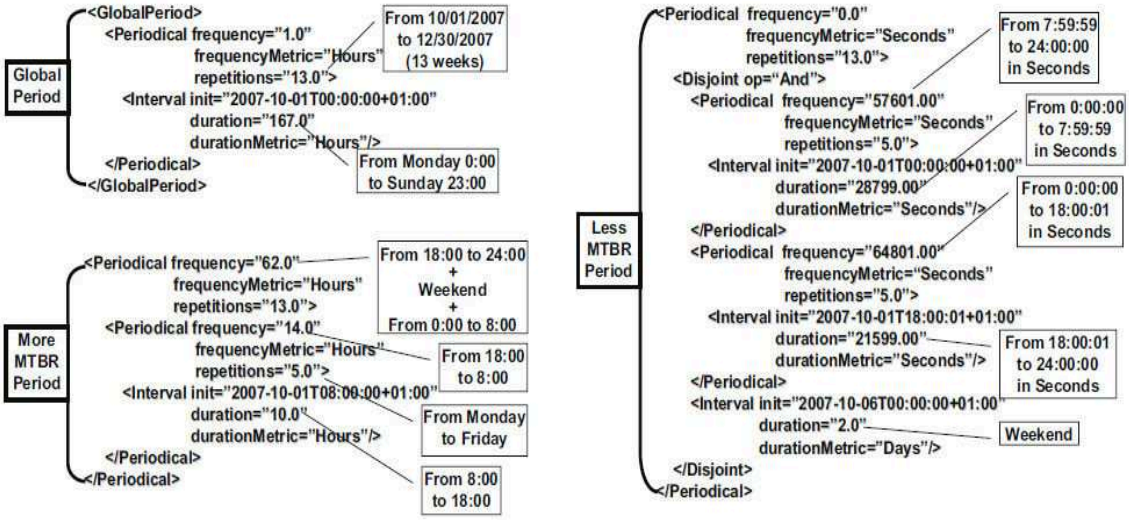

FIGURE C.1: Period Definitions in WS-Agreement*

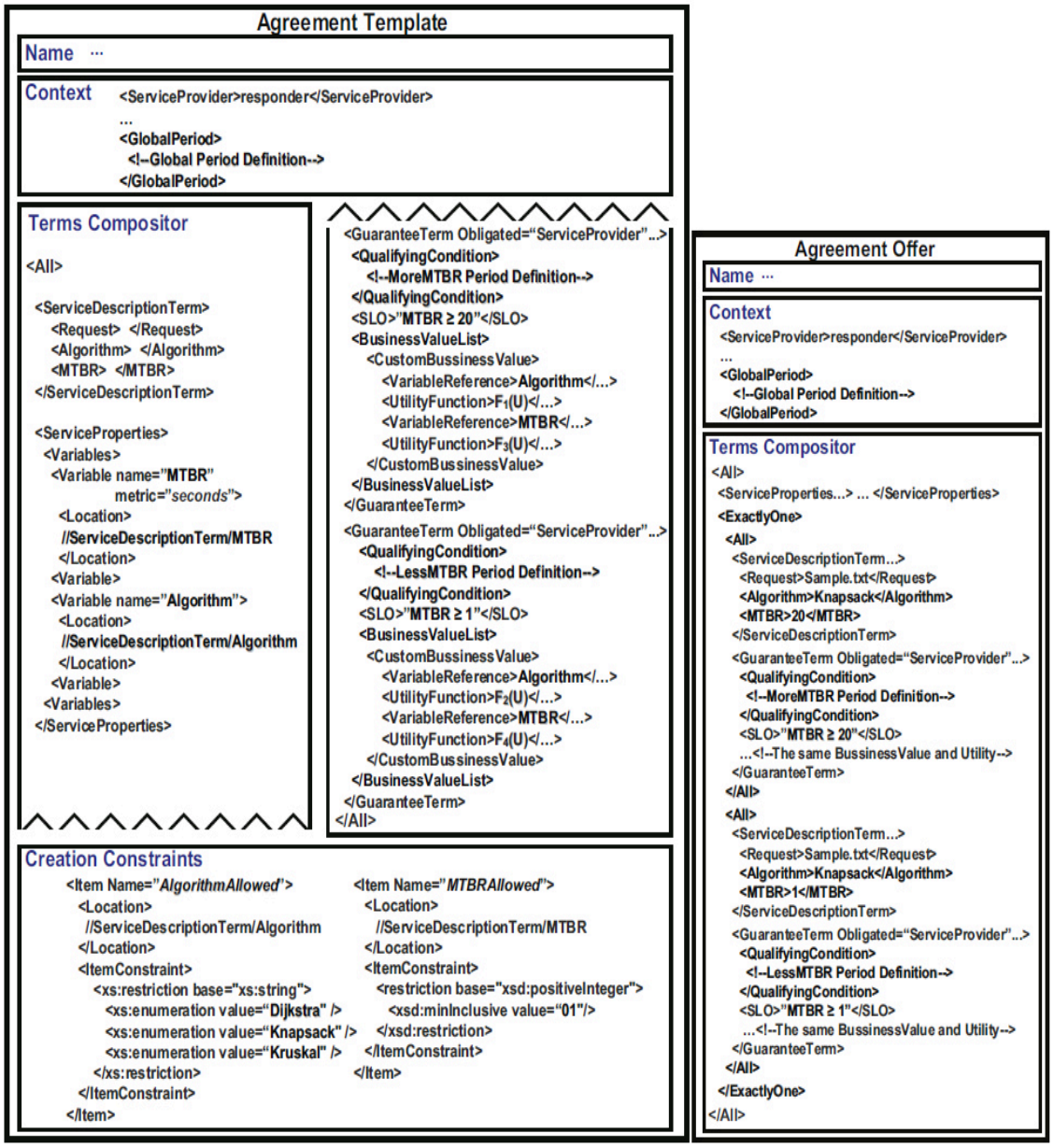

FIGURE C.2: Temporal-Aware Agreement Template and Offer using WS-Agreement*

Bibliography

[1] Florian Wagner, Adrian Klein, Benjamin Klöpper, Fuyuki Ishikawa, and Shinichi Honiden. Multi-objective service composition with time- and input-dependent qos. In *2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, June 24-29, 2012*, pages 234–241, 2012.

[2] Adrian Klein, Florian Wagner, Fuyuki Ishikawa, and Shinichi Honiden. A probabilistic approach for long-term B2B service compositions. In *2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, June 24-29, 2012*, pages 259–266, 2012.

[3] Boualem Benatallah and Hamid R. Motahari Nezhad. Service oriented architecture: Overview and directions. In *Advances in Software Engineering, Lipari Summer School 2007, Lipari Island, Italy, July 8-21, 2007, Revised Tutorial Lectures*, pages 116–130, 2007.

[4] Mike P. Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *VLDB J.*, 16(3):389–415, 2007.

[5] Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, 2007.

[6] Surya Nepal and Athman Bouguettaya. Issues and challenges in web service management systems. *IJNGC*, 3(2), 2012.

[7] Andreas Lanz, Jens Kolb, and Manfred Reichert. Enabling personalized process schedules with time-aware process views. In *Advanced Information Systems Engineering Workshops - CAiSE 2013 International Workshops, Valencia, Spain, June 17-21, 2013. Proceedings*, pages 205–216, 2013.

[8] Nawal Guermouche and Claude Godart. Composition of web services based on timed mediation. *IJNGC*, 5(1), 2014.

[9] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*, pages 191–205, 2013.

[10] Helan Liang, Yanhua Du, and Sujian Li. An improved genetic algorithm for service selection under temporal constraints in cloud computing. In *Web Information Systems Engineering - WISE 2013 - 14th International Conference, Nanjing, China, October 13-15, 2013, Proceedings, Part II*, pages 309–318, 2013.

[11] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005*, pages 1069–1075, 2005.

[12] J. Cardoso, A.P. Sheth, J.A. Miller, J. Arnold, and K. Kochut. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, 1(3):281–308, 2004.

[13] Leilei Chen, Jian Yang, and Liang Zhang. Time based qos modeling and prediction for web services. In *Service-Oriented Computing - 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings*, pages 532–540, 2011.

[14] Benjamin Klöpper, Fuyuki Ishikawa, and Shinichi Honiden. Service composition with pareto-optimality of time-dependent qos attributes. In *Service-Oriented Computing - 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10, 2010. Proceedings*, pages 635–640, 2010.

[15] Octavio Martín-Díaz, Antonio Ruiz Cortés, Amador Durán, and Carlos Müller. An approach to temporal-aware procurement of web services. In *Service-Oriented Computing - ICSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005, Proceedings*, pages 170–184, 2005.

[16] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 881–890, 2009.

[17] Lina Barakat, Simon Miles, Iman Poernomo, and Michael Luck. Efficient multi-granularity service composition. In *2011 IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, July 4-9, 2011*, pages 227–234, 2011.

[18] Ikbel Guidara, Tarak Chaari, and Mohamed Jmaiel. An efficient service selection approach with time-dependent qos. In *2014 IEEE 23rd International WETICE Conference, WETICE 2014, Parma, Italy, 23-25 June, 2014*, pages 320–325, 2014.

[19] Ikbel Guidara, Nawal Guermouche, Tarak Chaari, Saïd Tazi, and Mohamed Jmaiel. Pruning based service selection approach under qos and temporal constraints. In *2014 IEEE International Conference on Web Services, ICWS 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 9–16, 2014.

[20] Ikbel Guidara, Nawal Guermouche, Tarak Chaari, Mohamed Jmaiel, and Saïd Tazi. Time-dependent qos aware best service combination selection. *Int. J. Web Service Res.*, 12(2):1–25, 2015.

[21] Ikbel Guidara, Nawal Guermouche, Tarak Chaari, Saïd Tazi, and Mohamed Jmaiel. Heuristic based time-aware service selection approach. In *2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, June 27 - July 2, 2015*, pages 65–72, 2015.

[22] Ikbel Guidara, Imane Al Jaouhari, and Nawal Guermouche. Dynamic selection for service composition based on temporal and qos constraints. In *2016 IEEE International Conference on Services Computing, SCC 2016, San Francisco, SF, USA, June 27 - July 2, 2016*, 2016.

[23] Ikbel Guidara, Nawal Guermouche, Tarak Chaari, Mohamed Jmaiel, and Saïd Tazi. Proactive service selection in dynamic service oriented systems (submitted to an international journal). 2016.

[24] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *4th International Conference on Web Information Systems Engineering, WISE 2003, Rome, Italy, December 10-12, 2003*, pages 3–12, 2003.

[25] Mahesh H. Dodani. From objects to services: A journey in search of component reuse nirvana. *Journal of Object Technology*, 3(8):49–54, 2004.

[26] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. W3c note, March 2001.

[27] Kyriakos Kritikos and Dimitris Plexousakis. Semantic qos metric matching. In *Fourth IEEE European Conference on Web Services (ECOWS 2006), 4-6 December 2006, Zürich, Switzerland*, pages 265–274, 2006.

[28] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. Uddi version 3.0.2, October 2004.

[29] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. Soap version 1.2 part 1: Messaging framework, June 2003.

[30] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC 2004, San Diego, CA, USA, July 6, 2004, Revised Selected Papers*, pages 43–54, 2004.

[31] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau. HTN planning for web service composition using SHOP2. *J. Web Sem.*, 1(4):377–396, 2004.

[32] Paolo Traverso and Marco Pistore. Automated composition of semantic web services into executable processes. In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, pages 380–394, 2004.

[33] Girish Chafle, Koustuv Dasgupta, Arun Kumar, Sumit Mittal, and Biplav Srivastava. Adaptation inweb service composition and execution. In *2006 IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA*, pages 549–557, 2006.

[34] Danilo Ardagna, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani. PAWS: A framework for executing adaptive web-service processes. *IEEE Software*, 24(6):39–46, 2007.

[35] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *TWEB*, 1(1), 2007.

[36] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1): 5–51, 2003.

[37] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. Qos aggregation for web service composition using workflow patterns. In *8th International Enterprise Distributed Object Computing Conference (EDOC 2004), 20-24 September 2004, Monterey, California, USA, Proceedings*, pages 149–159, 2004.

[38] Fabio Casati, Ski Ilnicki, Li-jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and dynamic service composition in *eFlow*. In *Advanced Information Systems Engineering, 12th International Conference CAiSE 2000, Stockholm, Sweden, June 5-9, 2000, Proceedings*, pages 13–31, 2000.

[39] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[40] Martin Alt, Andreas Hoheisel, Hans Werner Pohl, and Sergei Gorlatch. A grid workflow language using high-level petri nets. In *Parallel Processing and Applied Mathematics, 6th International Conference, PPAM 2005, Poznan, Poland, September 11-14, 2005, Revised Selected Papers*, pages 715–722, 2005.

[41] Michael C. Jaeger, Gero Mühl, and Sebastian Golze. Qos-aware composition of web services: A look at selection algorithms. In *2005 IEEE International Conference on Web Services (ICWS 2005), 11-15 July 2005, Orlando, FL, USA*, pages 807–808, 2005.

[42] Carlos A. Coello Coello, Clarisse Dhaenens, and Laetitia Jourdan. Multi-objective combinatorial optimization: Problematic and context. In *Advances in Multi-Objective Nature Inspired Computing*, pages 1–21. 2010.

[43] Immanuel Trummer, Boi Faltings, and Walter Binder. Multi-objective quality-driven service selection - A fully polynomial time approximation scheme. *IEEE Trans. Software Eng.*, 40(2):167–191, 2014.

[44] Kwangsun Yoon Paul K. Yoon, Ching-Lai Hwang. *Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences)*. Sage Publications, 1995.

[45] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evolutionary Computation*, 3(4):257–271, 1999.

[46] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, editors, *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2001. International Center for Numerical Methods in Engineering.

[47] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[48] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation*, 6(2):182–197, 2002.

[49] J. Knowles and D. Corne. The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, 1999.

[50] D.W. Corne, J.D. Knowles, and M.J. Oates. The Pareto envelope-based selection algorithm for multiobjective optimization. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, 2000.

[51] Qi Yu and Athman Bouguettaya. Computing service skyline from uncertain qows. *IEEE Trans. Services Computing*, 3(1):16–29, 2010.

[52] Hui Ma, Favyen Bastani, I-Ling Yen, and Hong Mei. Qos-driven service composition with reconfigurable services. *IEEE T. Services Computing*, 6(1):20–34, 2013.

[53] Mohammad Alrifai, Dimitrios Skoutas, and Thomas Risse. Selecting skyline services for qos-based web service composition. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 11–20, 2010.

[54] Ying Chen, Jiwei Huang, and Chuang Lin. Partial selection: An efficient approach for qos-aware web service composition. In *2014 IEEE International Conference on Web Services, ICWS, 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 1–8, 2014.

[55] Rosario Giuseppe Garroppo, Stefano Giordano, and Luca Tavanti. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, 54(17):3081–3107, 2010.

[56] Krzysztof R. Apt. *Principles of constraint programming.* Cambridge University Press, 2003. ISBN 978-0-521-82583-2.

[57] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[58] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Inf. Sci.*, 237:82–117, 2013.

[59] Bart G. W. Craenen, A. E. Eiben, and Jano I. van Hemert. Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Trans. Evolutionary Computation*, 7(5):424–444, 2003.

[60] Boualem Benatallah, Quan Z. Sheng, Anne H. H. Ngu, and Marlon Dumas. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 297–308, 2002.

[61] Fei Li, Fangchun Yang, Kai Shuang, and Sen Su. Q-peer: A decentralized qos registry architecture for web services. In *Service-Oriented Computing - ICSOC 2007, Fifth International Conference, Vienna, Austria, September 17-20, 2007, Proceedings*, pages 145–156, 2007.

[62] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma. A qos-aware selection model for semantic web services. In *Service-Oriented Computing - ICSOC 2006, 4th International Conference, Chicago, IL, USA, December 4-7, 2006, Proceedings*, pages 390–401, 2006.

[63] Daniel A. Menascé. Qos issues in web services. *IEEE Internet Computing*, 6(6): 72–75, 2002.

[64] Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng. Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 66–73, 2004.

[65] Kun Yang, Alex Galis, and Hsiao-Hwa Chen. Qos-aware service selection algorithms for pervasive service composition in mobile wireless environments. *MONET*, 15(4):488–501, 2010.

[66] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004.

[67] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Software Eng.*, 33(6):369–384, 2007.

[68] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, pages 411–421, 2003.

[69] Danilo Ardagna and Barbara Pernici. Global and local qos constraints guarantee in web service selection. In *2005 IEEE International Conference on Web Services (ICWS 2005), 11-15 July 2005, Orlando, FL, USA*, pages 805–806, 2005.

[70] Lina Barakat. *Efficient Adaptive Multi-Granularity Service Composition.* PhD thesis, School of Natural & Mathematical Sciences, 2013.

[71] Nebil Ben Mabrouk. *QoS-aware Service-Oriented Middleware for Pervasive Environments.* PhD thesis, University of Paris 6, 2012.

[72] Lianyong Qi, Ying Tang, Wanchun Dou, and Jinjun Chen. Combining local optimization and enumeration for qos-aware web service composition. In *IEEE International Conference on Web Services, ICWS 2010, Miami, Florida, USA, July 5-10, 2010*, pages 34–41, 2010.

[73] Sherry X. Sun and Jing Zhao. A decomposition-based approach for service composition with global qos guarantees. *Inf. Sci.*, 199:138–153, 2012.

[74] Farhad Mardukhi, Naser Nematbakhsh, Kamran Zamanifar, and Asghar Barati. Qos decomposition for service composition using genetic algorithm. *Appl. Soft Comput.*, 13:3409–3421, 2013.

[75] Qiang He, Jun Yan, Hai Jin, and Yun Yang. Quality-aware service selection for service-based systems based on iterative multi-attribute combinatorial auction. *IEEE Trans. Software Eng.*, 40(2):192–215, 2014.

[76] Ahlem Ben Hassine, Shigeo Matsubara, and Toru Ishida. A constraint-based approach to horizontal web service composition. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, pages 130–143, 2006.

[77] Florian Rosenberg, Predrag Celikovic, Anton Michlmayr, Philipp Leitner, and Schahram Dustdar. An end-to-end approach for qos-aware service composition. In *Proceedings of the 13th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2009, 1-4 September 2009, Auckland, New Zealand*, pages 151–160, 2009.

[78] Zachary J. Oster, Ganesh Ram Santhanam, and Samik Basu. Identifying optimal composite services by decomposing the service composition problem. In *IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, July 4-9, 2011*, pages 267–274, 2011.

[79] Xin Yuan. Heuristic algorithms for multiconstrained quality-of-service routing. *IEEE/ACM Trans. Netw.*, 10(2):244–256, 2002.

[80] Zhenqiu Huang, Wei Jiang, Songlin Hu, and Zhiyong Liu. Effective pruning algorithm for qos-aware service composition. In *2009 IEEE Conference on Commerce*

and Enterprise Computing, CEC 2009, Vienna, Austria, July 20-23, 2009, pages 519–522, 2009.

[81] Zhen Ye, Xiaofang Zhou, and Athman Bouguettaya. Genetic algorithm based qos-aware service compositions in cloud computing. In *Database Systems for Advanced Applications - 16th International Conference, DASFAA 2011, Hong Kong, China, April 22-25, 2011, Proceedings, Part II*, pages 321–334, 2011.

[82] Lei Cao, Minglu Li, and Jian Cao. Using genetic algorithm to implement cost-driven web service selection. *Multiagent and Grid Systems*, 3(1):9–17, 2007.

[83] Yuan-hong Shen and Xiao-hu Yang. A self-optimizing qos-aware service composition approach in a context sensitive environment. *Journal of Zhejiang University - Science C*, 12(3):221–238, 2011.

[84] Cristina Bianca Pop, Viorica R. Chifu, Ioan Salomie, Mihaela Dinsoreanu, Tudor David, and Vlad Acretoaie. Ant-inspired technique for automatic web service composition and selection. In *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2010, Timisoara, Romania, 23-26 September 2010*, pages 449–455, 2010.

[85] Jiuyun Xu and Stephan Reiff-Marganiec. Towards heuristic web services composition using immune algorithm. In *2008 IEEE International Conference on Web Services (ICWS 2008), September 23-26, 2008, Beijing, China*, pages 238–245, 2008.

[86] Ioan Salomie, Monica Vlad, Viorica R. Chifu, and Cristina Bianca Pop. Hybrid immune-inspired method for selecting the optimal or a near-optimal service composition. In *Federated Conference on Computer Science and Information Systems - FedCSIS 2011, Szczecin, Poland, 18-21 September 2011, Proceedings*, pages 997–1003, 2011.

[87] Viorica R. Chifu, Cristina Bianca Pop, Ioan Salomie, Mihaela Dinsoreanu, Alexandru Nicolae Niculici, and Dumitru Samuel Suia. Selecting the optimal web service composition based on a multi-criteria bee-inspired method. In *iiWAS'2010 - The 12th International Conference on Information Integration and Web-based Applications and Services, 8-10 November 2010, Paris, France*, pages 40–47, 2010.

[88] Tao Zhang, Jianfeng Ma, Cong Sun, Qi Li, and Ning Xi. Service composition in multi-domain environment under time constraint. In *2013 IEEE 20th International Conference on Web Services, Santa Clara, CA, USA, June 28 - July 3, 2013*, pages 227–234, 2013.

[89] H. T. Kung. On the computational complexity of finding the maxima of a set of vectors. In *15th Annual Symposium on Switching and Automata Theory, New Orleans, Louisiana, USA, October 14-16, 1974*, pages 117–121, 1974.

[90] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430, 2001.

[91] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 467–478, 2003.

[92] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient progressive skyline computation. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 301–310, 2001.

[93] Karim Benouaret, Djamal Benslimane, and Allel HadjAli. Ws-sky: An efficient and flexible framework for qos-aware web service selection. In *2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, June 24-29, 2012*, pages 146–153, 2012.

[94] Osama Sammodi, Andreas Metzger, Xavier Franch, Marc Oriol, Jordi Marco, and Klaus Pohl. Usage-based online testing for proactive adaptation of service-based applications. In *Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2011, Munich, Germany, 18-22 July 2011*, pages 582–587, 2011.

[95] Andreas Metzger, Osama Sammodi, Klaus Pohl, and Mark Rzepka. Towards pro-active adaptation with confidence: augmenting service monitoring with online testing. In *2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2010, Cape Town, South Africa, May 3-4, 2010*, pages 20–28, 2010.

[96] Nelly Delgado, Ann Q. Gates, and Steve Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Trans. Software Eng.*, 30(12):859–872, 2004.

[97] Marcos Palacios, José García-Fanjul, and Javier Tuya. Testing in service oriented architectures with dynamic binding: A mapping study. *Information & Software Technology*, 53(3):171–189, 2011.

[98] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3), 2010.

[99] Girish Chafle, Koustuv Dasgupta, Arun Kumar, Sumit Mittal, and Biplav Srivastava. Adaptation inweb service composition and execution. In *2006 IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA*, pages 549–557, 2006.

[100] Swaroop Kalasapur, Mohan Kumar, and Behrooz Shirazi. Dynamic service composition in pervasive computing. *IEEE Trans. Parallel Distrib. Syst.*, 18(7):907–918, 2007.

[101] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. Qos-aware replanning of composite web services. In *2005 IEEE International Conference on Web Services (ICWS 2005), 11-15 July 2005, Orlando, FL, USA*, pages 121–129, 2005.

[102] Tao Yu and Kwei-Jay Lin. Adaptive algorithms for finding replacement services in autonomic distributed business processes. In *2005 International Symposium on Autonomous Decentralized Systems, ISADS 2005, Chengdu, China, April 4-8, 2005, Proceedings*, pages 427–434, 2005.

[103] Rafael Aschoff and Andrea Zisman. Qos-driven proactive adaptation of service composition. In *Service-Oriented Computing - 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings*, pages 421–435, 2011.

[104] Raman Kazhamiakin, Salima Benbernou, Luciano Baresi, Pierluigi Plebani, Maike Uhlig, and Olivier Barais. Adaptation of service-based systems. In *Service Research Challenges and Solutions for the Future Internet - S-Cube - Towards Engineering, Managing and Adapting Service-Based Systems*, pages 117–156, 2010.

[105] Lei Yang, Yu Dai, and Bin Zhang. Performance prediction based ex-qos driven approach for adaptive service composition. *J. Inf. Sci. Eng.*, 25(2):345–362, 2009.

[106] Yu Dai, Lei Yang, and Bin Zhang. Qos-driven self-healing web service composition based on performance prediction. *J. Comput. Sci. Technol.*, 24(2):250–261, 2009.

[107] Rafael Aschoff and Andrea Zisman. Proactive adaptation of service composition. In *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012, Zurich, Switzerland, June 4-5, 2012*, pages 1–10, 2012.

[108] René Ramacher and Lars Mönch. Reliable service reconfiguration for time-critical service compositions. In *2013 IEEE International Conference on Services Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*, pages 184–191, 2013.

[109] Yanhua Du, Xiaofei Wang, Lifeng Ai, and Xitong Li. Dynamic selection of services under temporal constraints in cloud computing. In *IEEE International Conference on e-Business Engineering ICEBE*, pages 252–259, 2012.

[110] Kwei-Jay Lin, Jing Zhang, and Yanlong Zhai. An efficient approach for service process reconfiguration in SOA with end-to-end qos constraints. In *2009 IEEE Conference on Commerce and Enterprise Computing, CEC 2009, Vienna, Austria, July 20-23, 2009*, pages 146–153, 2009.

[111] Jing Li, Dianfu Ma, Xiupei Mei, Hailong Sun, and Zibin Zheng. Adaptive qos-aware service process reconfiguration. In *IEEE International Conference on Services Computing, SCC 2011, Washington, DC, USA, 4-9 July, 2011*, pages 282–289, 2011.

[112] Azlan Ismail, Jun Yan, and Jun Shen. Incremental service level agreements violation handling with time impact analysis. *Journal of Systems and Software*, 86(6): 1530–1544, 2013.

[113] Zeina Azmeh, Marianne Huchard, Chouki Tibermacine, Christelle Urtado, and Sylvain Vauttier. Using Concept Lattices to Support Web Service Compositions with Backup Services. In *ICIW 2010: International Conference on Internet and Web Applications and Services*, pages 363–368, Barcelona, Spain, 2010. IEEE Computer Society.

[114] Florian Wagner, Benjamin Klöpper, Fuyuki Ishikawa, and Shinichi Honiden. Towards robust service compositions in the context of functionally diverse services. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 969–978, 2012.

[115] Girish Chafle, Prashant Doshi, John Harney, Sumit Mittal, and Biplav Srivastava. Improved adaptation of web service compositions using value of changed information. In *2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA*, pages 784–791, 2007.

[116] Nebil Ben Mabrouk, Sandrine Beauche, Elena Kuznetsova, Nikolaos Georgantas, and Valérie 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*, pages 123–142, 2009.

[117] Lina Barakat, Simon Miles, and Michael Luck. Reactive service selection in dynamic service environments. In *Service-Oriented and Cloud Computing - First*

European Conference, ESOCC 2012, Bertinoro, Italy, September 19-21, 2012. Proceedings, pages 17–31, 2012.

[118] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C.K. Liu, R. Khalaf, Dieter Koenig, M. Marin, V. Mehta, S. Thatte, D. Rijn, P. Yendluri, and A. Yiu. Web Services Business Process Execution Language Version 2.0 (OASIS Standard), 2007.

[119] Lina Barakat, Simon Miles, and Michael Luck. Efficient correlation-aware service selection. In *2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, June 24-29, 2012*, pages 1–8, 2012.

[120] Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. Toward a time-centric modeling of business processes in BPMN 2.0. In *The 15th International Conference on Information Integration and Web-based Applications & Services, IIWAS '13, Vienna, Austria, December 2-4, 2013*, page 154, 2013.

[121] Camilo Flores and Marcos Sepúlveda. Temporal specification of business processes through project planning tools. In *Business Process Management Workshops - BPM 2010 International Workshops and Education Track, Hoboken, NJ, USA, September 13-15, 2010, Revised Selected Papers*, pages 85–96, 2010.

[122] Denis Gagné and André Trudel. Time-bpmn. In *2009 IEEE Conference on Commerce and Enterprise Computing, CEC 2009, Vienna, Austria, July 20-23, 2009*, pages 361–367, 2009.

[123] Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. Enhancing formal specification and verification of temporal constraints in business processes. In *IEEE International Conference on Services Computing, SCC 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 701–708, 2014.

[124] Wenjun Li, Xi Li, Xiao-jun Liang, and Xiaocong Zhou. Qos-driven service composition with multiple flow structures. In *IEEE International Conference on Services Computing, SCC 2011, Washington, DC, USA, 4-9 July, 2011*, pages 362–369, 2011.

[125] Carlos Müller, Manuel Resinas, and Antonio Ruiz Cortés. Using automated analysis of temporal-aware slas in logistics. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops - International Workshops, ICSOC/ServiceWave 2009, Stockholm, Sweden, November 23-27, 2009, Revised Selected Papers*, pages 156–164, 2009.

[126] Carlos Müller, Manuel Resinas, and Antonio Ruiz Cortés. Explaining the non-compliance between templates and agreement offers in ws-agreement. In *Service-Oriented Computing, 7th International Joint Conference, ICSOC-ServiceWave 2009, Stockholm, Sweden, November 24-27, 2009. Proceedings*, pages 237–252, 2009.

[127] Abdessalam Elhabbash, Rami Bahsoon, Peter Tiňo, and Peter R. Lewis. Self-adaptive volunteered services composition through stimulus- and time-awareness. In *2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, June 27 - July 2, 2015*, pages 57–64, 2015.

[128] Sajib Mistry, Athman Bouguettaya, Hai Dong, and A. Kai Qin. Predicting dynamic requests behavior in long-term iaas service composition. In *2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, June 27 - July 2, 2015*, pages 49–56, 2015.

[129] Carlos Müller, Octavio Martín-Díaz, Antonio Ruiz Cortés, Manuel Resinas, and Pablo Fernandez. Improving temporal-awareness of ws-agreement. In *Service-Oriented Computing - ICSOC 2007, Fifth International Conference, Vienna, Austria, September 17-20, 2007, Proceedings*, pages 193–206, 2007.

[130] Ying Chen, Jiwei Huang, Chuang Lin, and Jie Hu. A partial selection methodology for efficient qos-aware service composition. *IEEE Trans. Services Computing*, 8 (3):384–397, 2015.

[131] Marco Comuzzi and Barbara Pernici. An architecture for flexible web service qos negotiation. In *Ninth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2005), 19-23 September 2005, Enschede, The Netherlands*, pages 70–82, 2005.

[132] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–136, 1982.

[133] Glenn Milligan and Martha Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

[134] James C. Bezdek and Nikhil R. Pal. Some new indexes of cluster validity. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 28(3):301–315, 1998.

[135] *Business Process Modeling Notation (BPMN).* Object Management Group (OMG), version 1.1 edition, 2008 .

[136] Saoussen Cheikhrouhou. *Specification and Verification of Temporal Constraints in Inter-Organisational Business Processes.* PhD thesis, University of Sfax, 2015.

[137] Eyhab Al-Masri and Qusay H. Mahmoud. Qos-based discovery and ranking of web services. In *Proceedings of the 16th International Conference on Computer Communications and Networks, IEEE ICCCN 2007, Turtle Bay Resort, Honolulu, Hawaii, USA, August 13-16, 2007*, pages 529–534, 2007.

[138] Eyhab Al-Masri and Qusay H. Mahmoud. Discovering the best web service. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 1257–1258, 2007.

[139] Hua Guo, Fei Tao, Lin Zhang, Suiyi Su, and Nan Si. Correlation-aware web services composition and qos computation model in virtual enterprise. *The International Journal of Advanced Manufacturing Technology*, 51(5):817–827, 2010.

[140] Shuiguang Deng, Hongyue Wu, Daning Hu, and J. Leon Zhao. Service selection for composition with qos correlations. *IEEE Transactions on Services Computing*.

[141] Yuzhang Feng, Le Duy Ngan, and Rajaraman Kanagasabai. Dynamic service composition with service-dependent qos attributes. In *2013 IEEE 20th International Conference on Web Services, Santa Clara, CA, USA, June 28 - July 3, 2013*, pages 10–17, 2013.

[142] Quanwang Wu, Qingsheng Zhu, and Mingqiang Zhou. A correlation-driven optimal service selection approach for virtual enterprise establishment. *J. Intelligent Manufacturing*, 25(6):1441–1453, 2014.

[143] Raf Haesen, Monique Snoeck, Wilfried Lemahieu, and Stephan Poelmans. On the definition of service granularity and its architectural impact. In *Advanced Information Systems Engineering, 20th International Conference, CAiSE 2008, Montpellier, France, June 16-20, 2008, Proceedings*, pages 375–389, 2008.