COST MANAGEMENT OF SERVICE COMPOSITION

Robson Wagner Albuquerque de Medeiros
COST MANAGEMENT
OF
SERVICE COMPOSITION

Robson Wagner Albuquerque de Medeiros
COST MANAGEMENT
OF
SERVICE COMPOSITION

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof.dr. T.T.M. Palstra,
on account of the decision of the graduation committee,
to be publicly defended
on Friday the 8th of December 2017 at 10.45

by

Robson Wagner Albuquerque de Medeiros

born on the 9th of August 1979
in Vitória de Santo Antão, Pernambuco, Brazil

This doctoral degree program was undertaken jointly with the Universidade Federal de Pernambuco (Brazil) and the University of Twente (the Netherlands)
This dissertation has been approved by:

Supervisors: prof. dr. R.J. Wieringa

prof. N. Souto Rosa

© 2017: Robson Wagner Albuquerque de Medeiros, the Netherlands

Cover: designed by Antônio Fernando de Medeiros


DOI: 10.3990/1.9789036544368

URL: https://doi.org/10.3990/1.9789036544368

Printed by Ipskamp, Enschede, the Netherlands
Graduation Committee

Chairman/secretary: prof.dr. P.M.G. Apers (Universiteit Twente)

Supervisor(s): prof.dr. R.J. Wieringa (Universiteit Twente)
prof. N. Souto Rosa (Universidade Federal de Pernambuco)

Referee: dr. D. Quartel (Bizzdesign)

Committee Members: prof.dr. J. van Hillegersberg (Universiteit Twente)
dr.ir. L. Ferreira Pires (Universiteit Twente)
prof. S. Rinderle (Universität Wien)
prof.dr.ir. M. Aiello (Rijksuniversiteit Groningen)
Abstract

Many organisations across the world have adopted Service-Oriented Architecture (SOA) to interconnect their computing infrastructures (Business-to-Business) and offer interfaces to their customers (Business-to-Customer). SOA can help these organisations access the market more quickly, respond to changes in a business environment, improve business processes and customers services and even reduce costs. In SOA, service composition has emerged as an important strategy to enable collaboration of applications provided by different organisations (Business-to-Business). With the increasing number of Web services having similar functionality but different pricing schemes, choosing the most appropriate set of services with the lowest cost has been a challenge in service compositions. In our exploratory literature review to identify the challenges and current concepts related to the treatment of the cost of service compositions, we observed the existence of several approaches to manage costs in Service-Oriented Computing (SOC). However, most of these approaches cannot cope with complex cost behaviours. Moreover, we did not find any solution that treats cost throughout the service composition life-cycle. Thus, the main objective of this thesis has been to develop a framework to manage cost throughout the service composition life-cycle in an integrated way, taking into account all classes of cost behaviour and all types of cost drivers. To achieve this objective, we first provided a metamodel to express cost behaviours of computational services by taking into account some characteristics of SOA and service cost, such as that services can have a particular cost for each operation and that service costs can have several types of cost drivers. In addition, we developed algorithms to
analyse the cost of service compositions and atomic services. Additionally, we proposed an architecture for developing software environment that are able to manage the cost throughout the service composition life-cycle. Finally, we implemented a prototype based on the proposed architecture and performed simulations that show the effectiveness and efficiency of our approach to manage the cost of service composition.
Wereldwijd hebben veel organisaties de Service-georienteerde Architectuur (SOA) geadopteerd om hun computing infrastructuren te verbinden (Business-to-Business) en interfaces aan hun klanten aan te bieden (Business-to-Customer). SOA kan deze organisaties helpen om de markt sneller te bedienen, om te reageren op veranderingen in business omgevingen, en om business processen en customer services te verbeteren en zelfs kosten te verminderen. Bij gebruik van SOA is service compositie een belangrijke strategie om samenwerking tussen applicaties van verschillende organisaties (Business-to-Business) te bevorderen. Met het groeiende aantal Web services met vergelijkbare functionaliteit maar verschillende prijschamers is de keuze van de meeste geschikte verzameling services met de laagste kosten in service composites een uitdaging. Onze literatuurverkenning van uitdagingen en huidige concepten gerelateerd aan de afhandeling van de kosten van service compositie laat zien dat er vele benaderingen bestaan om de kosten in Service-georienteerde Computing (SOC) te beheren. Echter, de meeste van deze benaderingen kunnen niet omgaan met complexe kosten. Bovendien hebben we geen oplossing kunnen vinden voor het beheren van kosten gedurende de hele service compositie levenscyclus. Deze bevindingen resulteerden in het belangrijkste doel van dit proefschrift: het ontwikkelen van een raamwerk waarmee we de kosten van service compositie gedurende de hele service compositie levenscyclus op een gентegreerde manier kunnen beheren, door alle klassen van kostengedrag en alle kostenfactoren te beschouwen. Om dit doel te verwezenlijken hebben we eerst een metamodel gemaakt om kosten van computing services
uit te drukken. Dit metamodel houdt rekening met de eigenschappen van SOA en service kosten, zoals dat services specifieke kosten kunnen hebben voor elke operatie en dat service kosten verschillende kostenfactoren kunnen hebben. Verder hebben we algoritmen ontwikkeld om de kosten van atomaire services en van service composities te analyseren. Bovendien hebben we een architectuur voorgesteld om systemen te ontwikkelen waarmee kosten gedurende de hele service compositie levenscyclus beheerd kunnen worden. Tenslotte hebben we een prototype gemoduleerd gebaseerd op de voorgestelde architectuur, en simulaties uitgevoerd die de effectiviteit en efficiency aantonen van onze benadering om service compositie kosten te beheren.
This work would not have been achievable without the support of many. I would like to thank and dedicate this thesis to the following people:

- To God, who provides me with enough strength and wisdom to achieve my objectives;
- To Nelson Rosa and Luís Pires, for their great assistance. Both helped me whenever I had questions with prompt feedback. Their guidance and great support helped me to build a professional research career. Their leadership and enthusiasm were fundamental to motivate me throughout this research;
- To my promoter Prof. Dr. Roel Wieringa, for all his support in defining the research methodology of my thesis;
- To University of Twente, for all supports on the development of this research in the Netherlands, especially Suse Engbers, Bertine Scholten-Koop and Geert Jan Laanstra;
- To my friends and colleagues from GFADS. Thank you for the thought-provoking discussions;
- To Rural Federal University of Pernambuco (UFRPE), for all supports on the development of my research;
- To my friends, including (but not limited to) Fernando Aires, Erica Sousa, Marcelo Marinho, Gilberto Cysneiros, Glaucia Campos, José Gonçalves, João Moreira and Carlos Azevedo. Thank you for all happy moments and for being there when I needed most;
To my beloved parents, Vera and Fernando, and my brother Antônio Jr., for always supporting me through my whole life. I would not be here without you. Saying thanks to you is not enough;

Last but not least, from the bottom of my heart, my deepest thank goes to my beloved wife Adriana and my son Lucas, without their love I would not have finished this thesis. Words would never say how thankful I am to you.
1 Introduction ......................................................... 1
  1.1 Motivation .................................................. 1
    1.1.1 Cost Management ........................................ 3
    1.1.2 Cost Behaviours ........................................... 4
    1.1.3 Example .................................................. 4
  1.2 Research Design .............................................. 7
  1.3 Problem ...................................................... 8
  1.4 Objective .................................................... 9
  1.5 Research Questions ......................................... 9
  1.6 Scope and Non-Objectives ................................. 11
  1.7 Structure of Thesis ................................... 11

Part I Problem Investigation

2 Background .......................................................... 17
  2.1 Service-Oriented Architecture ............................... 17
    2.1.1 Web Services .............................................. 18
    2.1.2 Service Composition ...................................... 21
  2.2 Concepts of Business Process Management ............... 24
    2.2.1 Business Process Model and Notation ................... 24
  2.3 Cost Management ........................................... 28
  2.4 Reliability .................................................. 36
    2.4.1 Reliability of Sequential Workflow ..................... 36
    2.4.2 Reliability of Parallel Workflow ....................... 36
    2.4.3 Reliability of Conditional Workflow .................... 37
## Contents

2.4.4 Reliability of Loop Workflow .......................... 38
2.5 Concluding Remarks ................................. 39

3 Cost Management in Service-Oriented Computing ............ 41
3.1 Planning ............................................ 41
3.2 Definition .......................................... 42
3.3 Scheduling and Construction ............................ 43
3.4 Execution .......................................... 46
3.5 Discussion .......................................... 46
3.6 Key Challenges ...................................... 48
3.7 Concluding Remarks ................................. 49

### Part II Treatment Design

4 Cost Behaviour Modelling ................................. 53
4.1 Motivation and Requirements ............................ 53
4.2 Overview of the Cost Behaviour Metamodel .............. 55
4.3 The Cost Behaviour Definition .......................... 55
4.4 Metamodel Serialisation ................................ 64
4.5 Metamodel Instance Examples ............................ 65
4.6 Concluding Remarks .................................... 72

5 Techniques for Cost Analysis ................................. 73
5.1 Motivation and Requirements ............................ 73
5.2 Overview .......................................... 74
5.3 Cost Analysis Using Annotations in Planning Phase .... 76
5.4 Cost Analysis Using Event Logs in Planning Phase .... 79
5.5 Cost Analysis Using Event Logs in Execution Phase ..... 82
5.6 Cost Analysis for One Service Composition Instance in Execution Phase .................................. 84
5.7 Concluding Remarks .................................... 88

6 Architecture of Cost Management Systems ................. 91
6.1 Motivation and Requirements ............................ 91
6.2 Overview .......................................... 92
6.3 Modeller .......................................... 95
6.4 Execution Engine .................................... 97
6.5 Service Selector ...................................... 97
Contents

6.6 Monitor ........................................ 101
6.7 Analyser ....................................... 102
6.8 Adapter ......................................... 107
6.9 Concluding Remarks ......................... 107

Part III Treatment Validation

7 Prototype ........................................ 111
  7.1 Implementation ................................. 111
  7.2 Example Applications ....................... 118
    7.2.1 Audio Files Converter ................. 119
    7.2.2 Pizza Delivery ......................... 120
  7.3 Concluding Remarks ......................... 120

8 Experimental Evaluation ....................... 123
  8.1 Cost Prediction by Using BPMN ............ 123
    8.1.1 Experimental Setup ................... 124
    8.1.2 Statistical Analysis ................... 125
    8.1.3 Predicting Cost ......................... 126
    8.1.4 Experimental Results .................. 127
  8.2 Cost Predictions by Using BPMN and Event Log .... 127
    8.2.1 Experimental Setup ................... 128
    8.2.2 Predicting Cost ......................... 130
    8.2.3 Experimental Results .................. 130
  8.3 Adapting Service Composition in Execution Phase .......... 132
    8.3.1 Experimental Setup ................... 134
    8.3.2 Experimental Results .................. 136
  8.4 Concluding Remarks ......................... 140

9 Conclusions .................................... 141
  9.1 Answers to the Research Questions .......... 141
  9.2 Limitations and Future Work ................ 144
  References ..................................... 147

List of Publications ................................ 157

XML Schema Definition of the Cost Behaviour Metamodel .... 159
Acronyms

SOC  Service-Oriented Computing ......................... 1
SOA  Service-Oriented Architecture ...................... 1
QoS  Quality of Service .................................... 91
WSDL Web Service Definition Language .................. 1
SOAP Simple Object Access Protocol ...................... 1
BPM Business Process Management ....................... 12
BP  Business Process ....................................... 24
BPD Business Process Diagram ............................. 24
BPMN Business Process Model and Notation .............. 22
WS-BPEL Web Services Business Process Execution Language .... 22
IT Information Technology .................................. 17
Java EE Java Platform, Enterprise Edition ................ 19
HTTP Hypertext Transfer Protocol ......................... 2
XML eXtensible Markup Language .......................... 18
UDDI Universal Description, Discovery, and Integration ... 20
TCP Transmission Control Protocol ....................... 19
SMTP Simple Mail Transfer Protocol ...................... 19
URI Uniform Resource Identifier ........................... 20
IDL Interface Definition Language ........................ 18
PTCSP Priced Timed Communicating Sequential Process .... 43
PTCCS Priced Temporal Calculus of Communicating Systems ... 43
PPPA Priced Probabilistic Process Algebra ................ 43
TCSP Timed Communicating Sequential Process ............ 43
TCCS Temporal Calculus of Communicating Systems ........ 43
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPA</td>
<td>Probabilistic Process Algebra</td>
<td>43</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
<td>46</td>
</tr>
<tr>
<td>ABC</td>
<td>Activity-Based Costing</td>
<td>32</td>
</tr>
<tr>
<td>TDABC</td>
<td>Time-Driven Activity-Based Costing</td>
<td>32</td>
</tr>
<tr>
<td>RCA</td>
<td>Resource Consumption Accounting</td>
<td>32</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
<td>93</td>
</tr>
<tr>
<td>REST</td>
<td>REpresentation State Transfer</td>
<td>19</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
<td>19</td>
</tr>
<tr>
<td>XES</td>
<td>eXtensible Event Stream</td>
<td>118</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
<td>118</td>
</tr>
<tr>
<td>AOP</td>
<td>Aspect-oriented Programming</td>
<td>118</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma-Separated Values</td>
<td>19</td>
</tr>
<tr>
<td>MOF</td>
<td>MetaObject Facility</td>
<td>55</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition</td>
<td>64</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

Nowadays, several organisations provide services on the Internet, where these services offer similar functionality but with different non-functional properties, such as cost, reliability and performance. Since these services are used in service compositions, the providers of composed services have to face the challenge of managing the compositions by taking into account the trade-off relationship between their qualities.

This chapter initially presents the motivation of this thesis. It also discusses the research design followed in this thesis, and outlines our primary research objectives. After that, this chapter presents the research questions that we have answered to achieve the objectives of this thesis, and highlights our scope and non-objectives. Finally, it presents the structure of this thesis.

1.1 Motivation

Service-Oriented Architecture (SOA) [1] is a widely used approach to develop network applications based on the notion of Service-Oriented Computing (SOC) [2]. SOA is currently the favoured architectural style to achieve applications adaptability and has facilitated the interoperability among computer systems of possibly different businesses [3].

In SOA, service is the fundamental concept which can be characterised as an abstract resource that represents a capability [4]. This capability is described, published, and discovered by using languages and protocols (e.g., Web Service Definition Language (WSDL) [5], Simple Object Access Pro-
tocol (SOAP) [6] and Hypertext Transfer Protocol (HTTP)) that provide a flexible infrastructure to connect heterogeneous components and systems.

Many organisations across the world have adopted SOA to interconnect their computing infrastructures (Business-to-Business) and offer interfaces to their customers (Business-to-Customer) [7, 8]. SOA allows these companies to access the marketplace, quickly respond to the change in a business environment, and even reduce their development costs [3, 8]. A factor that contributes to reducing development cost in SOA projects is the reusability characteristic of the services, facilitated by the implementation-independence and standardized description of the services.

SOA also allows to create complex services by composing existing services, namely service composition. Service composition has emerged as an important strategy to enable the collaboration between companies through applications (Business-to-Business) [9]. Once services can be composed to create more complex services, the entire (or part of the) capability of a service can be outsourced to third-party instead of implemented by the service composition provider.

Once service compositions are implemented by combining other services, their quality depends on the quality of each service involved in the composition. For instance, a single insecure or unreliable service can affect the security and reliability of the whole service composition, respectively.

The significant business value provided by SOA has made the number of companies investing in SOA to increase even more. This market growth can affect both positively and negatively the service composition providers. On the one hand, service composition providers can find services in the marketplace with similar functionalities but with different qualities, which allows them to use the best services to be composed that meet the requirements of their business. On the other hand, service composition providers need to properly manage their finance to guarantee long-term profitability, especially in these highly competitive environments.

Service providers can maximise service profit in many ways, for example, by raising service price or reducing service costs. However, an increase in the service price can have an adverse effect on service profit, since the provider may become less competitive in the marketplace and lose its customers to other less expensive providers. Alternatively, by reducing the service costs, the provider can not only make its service more profitable but
also create opportunities to reduce the service price to increase its market penetration.

To manage and reduce the cost of service compositions, one needs to take into account the services available in the marketplace and their qualities, such as reliability and price. These qualities directly affect the cost of a service composition. Concerning reliability, a service failure may demand more resources for the execution of some corrective behaviour and may lead to a cost increase. Price is the amount of money paid by a customer to use the service [10, 11, 12]. From the point of view of the service composition providers, the price charged by the third-party providers for a component service is part of the cost to develop the service composition.

### 1.1.1 Cost Management

Cost management is a process that helps business experts to plan, control, and take decisions about the operational and financial situation of organisations [12, 13].

Cost management is crucial for service composition and must be considered throughout the service composition life-cycle to control its costs and keep its profit as planned. The service composition life-cycle consists of four phases, namely **Planning, Definition, Scheduling and Construction**, and **Execution** [14]. Moreover, two phases can be performed in parallel to the **Execution phase**, namely **Monitoring** and **Adaptation**.

An essential tool for performing cost management is a cost management system, which can be used to collect, categorise, summarise, and analyse costs and produce helpful information to [11]:

- Determine the effect of decisions concerning the service composition, for instance, the choice of the services participants and the business process;
- Ensure that the activities of the service composition are performed as efficiently as possible;
- Plan the service composition budget for a certain period; and
- Identify problems that can make the financial aspects of the service composition deviate from the planned situation.
1.1.2 Cost Behaviours

Cost behaviour denotes cost changes as a function of the activity levels. In other areas than SOA, the cost behaviour of each resource (e.g., persons and machines) is defined internally by the organisation. Therefore, the cost behaviours of similar resources tend to be similar. In contrast, in SOA most of the resources (services) are often provided by third-parties and, in the worst case, each service can have a different cost behaviour.

Service compositions have some cost management concerns throughout its life-cycle when complex cost behaviours have to be taken into account. The cost behaviour of a service is defined as the way the service cost changes depending on its utilisation, and can be classified as fixed, variable, mixed and step cost [10, 11, 12].

Factors that cause variations in the cost behaviour are known as cost drivers. A cost driver is a variable related to the service utilisation that changes the cost when the service is executed, such as a number of invocations and message size. Even if services belong to the same class of cost behaviour, their costs may be computed differently depending on the cost drivers. For instance, two services can have variable cost behaviours, but one can be charged based on the number of invocations while the other can be charged according to the size of the messages sent to the service. Moreover, to stimulate the service consumption and remain competitive, providers can offer discounts that vary according to the volume of use of their services.

1.1.3 Example

Fig. 1.1 illustrates some cost management concerns by showing a service composition example that converts audio files from WAV to MP3 format. This process starts when the customer invokes the service composition by informing her e-mail address and the WAV file to be converted. After that, the task Validate Email Address validates the e-mail address and, if it is valid, the task Convert WAV to MP3 Audio Format converts the WAV file into the MP3 file format. In contrast, if the e-mail address is invalid, the task Notify Administrator by Email sends an e-mail message to notify the administrator that an error occurred. If the file is converted correctly, the task Send Email...
1.1 Motivation

to Customer with MP3 File sends an e-mail to the customer having the MP3 file. Each task can be performed by a set of candidate services, i.e., services with similar functionality but with different cost behaviours, as shown in Table 1.1.

![Fig. 1.1 Service Composition Example.](Image)

<table>
<thead>
<tr>
<th>Task</th>
<th>Candidate Services</th>
<th>Cost Model (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validate Email Address</td>
<td>ABValidateEmail</td>
<td>0.05 per email address.</td>
</tr>
<tr>
<td></td>
<td>CDValidateEmail</td>
<td>0.06 per email address.</td>
</tr>
<tr>
<td>Notify Administrator</td>
<td>SimpleEmail</td>
<td>40.00 per month.</td>
</tr>
<tr>
<td>Convert WAV to MP3</td>
<td>VMail</td>
<td>0.10 per 100 message sent</td>
</tr>
<tr>
<td>Send Email to Customer with</td>
<td>ConvertFile</td>
<td>0.03 per 1 MB of file.</td>
</tr>
<tr>
<td>the MP3 File</td>
<td>AudioFileConverter</td>
<td>0.10 per file.</td>
</tr>
<tr>
<td>Send Email to Customer with</td>
<td>CompleteEmail</td>
<td>0.10 per 100 messages / month; and, Attachments:</td>
</tr>
<tr>
<td>the MP3 File</td>
<td></td>
<td>- First 10 GB/month: 0.12/MB,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Above 10 GB/month: 0.09/MB</td>
</tr>
<tr>
<td></td>
<td>CMail</td>
<td>0.09 per 100 messages / month; and, Attachments:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- First 10 GB/month: 0.12/MB,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Next 10 GB/month: 0.10/MB,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Above 20 GB/month: 0.08/MB</td>
</tr>
</tbody>
</table>

When a service composition is being planned, business experts can predict its cost to establish goals (e.g., the budget) and specify how to achieve
them. However, this cost can be affected, for instance, by the cost behaviours of the candidate services, the qualities of the candidate services, the demand for the service composition and the values of some attributes of the service composition, like the size of messages. For example, task Notify Administrator by Email has two candidate services: service SimpleEmail with a fixed cost (€40.00 per month) and service VMail with a variable cost (€0.10 per 100 messages sent), as shown in Table 1.1. In this case, the service that has the lowest cost is determined by how often the task Notify Administrator by Email is performed in a month, which depends on the demand for the service composition in the period and the probability that this task is performed in an execution of the service composition. If this task is executed less than 40,000 times within a month, the service with variable cost has the lowest cost for the composition. Otherwise, the service with fixed cost is the one with the lowest cost.

In the execution phase, the service composition and its cost should be monitored so that adjustments can be made to adhere to the required costs. In this phase, when a task has more than one candidate service, one concern is to select the candidate services to perform each instance of the service composition with the lowest cost. For example, the task Convert WAV to MP3 Audio Format in Fig. 1.1 can be performed by the services ConvertFile and AudioFileConverter. However, the service with the lowest cost depends on the size of the WAV audio file, as shown in Table 1.1, in which can be different in each execution of the service composition. Another example is the task Send Email to Customer with the MP3 File. In this task, the cheapest service depends on the demand of the service composition and the size of the MP3 files sent to the customers in a month of utilisation, since the cost behaviours of the candidate services can change depending on the number of attachments sent in a month. Therefore, proper tooling is necessary to estimate the utilisation of the service composition before selecting the most appropriate service.
1.2 Research Design

Following the recommendations of [15], the research design adopted in this thesis has three phases, namely problem investigation, treatment design and treatment validation, as depicted in Fig. 1.2.

![Fig. 1.2 Research Design Cycle.](image)

We started with the problem investigation, which includes the study of related work. After that and based on the results of the literature study, we defined our research questions, and the requirements for the development of the treatment, which are described as follows:

- **R1**: By the fact that there are many services adopting different cost behaviours, the treatment must support all identified classes of cost behaviours;
- **R2**: In order not to have a major impact on the service composition performance, the treatment must compute and predict service costs with little human intervention;
- **R3**: Due to the characteristics of the pricing models adopted in the services market, the treatment must support runtime adaptation of service compositions based on cost requirements and the cost behaviours of the services; and
- **R4**: Since there are already solutions to perform service composition, the treatment must support the integration of the proposed solution with existing service composition engines.
With these requirements for the development of the treatment, a framework was developed with the following artefacts:

- A metamodel to express cost behaviours;
- Algorithms to analyse the cost of services and service compositions during the planning and execution phases;
- An architecture for the development of cost management systems able to execute, monitor, and adapt service composition based on cost aspects; and
- A reference implementation of the proposed architecture.

After designing the treatment, we worked on the treatment validation. This procedure consists of the implementation of a prototype of the treatment and the evaluation of its performance, accuracy, and effectiveness in the cost management of service compositions.

1.3 Problem

When we started this research, we figured out that some challenges related to cost management of service composition are still open. For instance, in the Planning phase, an open challenge was to predict and analyse business cost taking into account all class of cost behaviour (i.e., fixed, variable, mixed and step cost). In the Definition phase, an open challenge was to express complex cost models like the ones defined by cloud providers. In the Scheduling and Construction phase, the challenge was to find services to perform the service composition being aware of its cost. When the service composition is in the Execution phase, one challenge was to minimise execution cost. For this to happen, activities in the Monitoring and Adaptation phases can be performed in order to monitor the service composition so that adjustments can be made to adhere to the expected cost taking into account all classes of cost behaviour. Finally, another current challenge was to treat cost information in all phases of the service composition life-cycle in an integrated way.

Therefore, the problem addressed in this thesis is the cost management throughout the service composition life-cycle taking into account complex cost behaviours and different type of cost drivers.
Furthermore, the sub-problems addressed in this thesis are related to the phases of the service composition life-cycle. In the Planning phase, the problem is to predict the cost of service composition taking into account complex cost behaviours. In the Definition phase, the problem is to express cost behaviour so that costs can be managed in a cost management system. In the Scheduling and Construction phase, the problem is to choose the best set of services to form a composition considering that these services have different cost behaviours. Finally, in the Execution phase the problem is to keep the cost of the service composition within the planned values, which can be enforced in the Monitoring and Adaptation phases.

1.4 Objective

The objective of this research is to develop a framework to develop service composition mechanisms that can manage cost throughout the service composition life-cycle taking into account cost properties of services and service compositions. This framework must provide means to express cost behaviour, to define techniques to predict the cost of service and service composition, to develop methods to choose services based on their cost behaviours and to propose techniques to monitor and adapt service composition based on cost.

This framework must also consider computational solutions related to the development, execution, monitoring, and adaptation of service compositions. Moreover, it must take into account all classes of cost behaviour and different types of cost drivers.

The stakeholders of interest of this thesis are developers of service compositions and cost-aware service composition engines, which will be able to develop new or adapt existing service composition engines to overcome cost management limitations typically found in these systems.

1.5 Research Questions

When defining our research questions, we broke down the problems found in cost management of service composition into smaller sub-problems so
that by solving these sub-problems we could reach our research objective. In this thesis, we addressed three research questions, which are categorized as empirical research questions (EQ), which require empirical research in order to be answered, technical research questions (TQ), which call for a design problem to be solved [16], and validation research questions (VQ), which require experiments that needs to evaluate how the proposal solution responds to different scenarios.

- **RQ1 (EQ)**: What are the current challenges found in the context of cost management of service compositions?
  - **RQ1.1**: What is the state-of-the-art in cost management of service compositions?
  - **RQ1.2**: What are the open challenges identified in the literature concerning the cost management of service compositions?

  In **RQ1**, we elaborated the research questions to identify more specific challenges in the context of cost management of service compositions. This research question aims at determining the problematic phenomena and the current state-of-the-art concerning cost-aware service compositions.

- **RQ2 (TQ)**: How to manage the cost of service compositions throughout their life-cycle?
  - **RQ2.1**: What are the phases of the service composition life-cycle?
  - **RQ2.2**: How to express cost behaviours in cost management systems?
  - **RQ2.3**: How to predict service composition costs with complex cost behaviours?
  - **RQ2.4**: How to manage the execution of the service composition to control costs?

  In **RQ2**, we elaborated research questions to identify techniques to manage the cost of service compositions throughout its life-cycle taking into account the cost behaviours of services.

- **RQ3 (VQ)**: What are the impacts and accuracy of our treatment to manage cost of service compositions?
  - **RQ3.1**: What is the financial gain of cost management in service compositions?
1.7 Structure of Thesis

- **RQ3.2** What is the performance impact of cost management in the execution of service compositions?
- **RQ3.3** What is the accuracy of our approach in the cost analysis of service compositions?

In **RQ3**, we elaborated research questions to validate the proposed techniques to manage the cost of service compositions.

1.6 Scope and Non-Objectives

The scope of this thesis is the cost management of service compositions. The focus of this thesis is on the modelling of cost behaviours taking into account the characteristics of services and service compositions. Moreover, this thesis addresses cost analysis, monitoring, and adaptation of service compositions.

In this thesis, we consider that a service composition is a business process in which all tasks are performed by computational services. Therefore, this thesis does not address tasks that are not performed by computational services, such as manual tasks.

Although a service is considered profitable when its total cost is lower than the income it generates, which is determined by its price, this thesis does not address the pricing of service compositions that can be affected by other factors, like, e.g., the competitor’s price [17, 18]. Moreover, we do not address financial accounting, which produces information for external stakeholders, such as accountants and auditors.

1.7 Structure of Thesis

The structure of this thesis reflects our research design discussed in Section 1.2. Fig. 1.3 depicts this structure, indicating which chapters answer our research questions and how the chapters relate to our research design. Therefore, we organised the chapters in parts according to our research design, as follows:

**Part I: Problem Investigation**
Chapter 2 (Background) introduces the concepts and terminology used throughout the thesis. This chapter introduces the principles of SOA, Business Process Management (BPM) and cost management. Since reliability is taken into consideration to compute the cost of service compositions, this concept is also discussed in this chapter.
1.7 Structure of Thesis

• Chapter 3 (Cost Management in Service-Oriented Computing) presents a survey of the currently available techniques used to manage cost in SOC.

Part II: Treatment Design

• Chapter 4 (Cost Behaviour Modelling) presents the metamodel designed to model service cost behaviours in SOC.
• Chapter 5 (Techniques for Cost Analysis) proposes techniques to analyse the cost of service compositions in the planning and execution phases of the service composition life-cycle.
• Chapter 6 (Architecture of Cost Management Systems) introduces our architecture for cost management of service compositions by discussing its components and their relationships.

Part III: Treatment Validation

• Chapter 7 (Prototype) discusses the prototype we built to validate our framework.
• Chapter 8 (Experimental Evaluation) discusses the experimental evaluation of our framework by using the developed prototype.
• Chapter 9 (Final Remarks) presents the conclusions of this thesis and identifies topics for further investigation.
Part I
Problem Investigation
Chapter 2
Background

This chapter introduces the general concepts and basic terminology used throughout this thesis. It presents the principles of Service-Oriented Architecture (SOA), Business Process Management (BPM) and cost management. Since reliability is taken into consideration in our research to compute the cost of service composition, this concept is also discussed here.

This chapter initially introduces the notion of SOA development. Next, it describes the basic concepts of BPM and cost management. Finally, it introduces the idea of reliability used in this thesis to analyse the cost of service composition.

2.1 Service-Oriented Architecture

Nowadays, in a connected and competitive world, organisations need to respond quickly to market opportunities, and the area of Information Technology (IT) has a fundamental role to make this possible. In general, organisations use different applications, new and legacy, scattered across various departments (or areas) or even outside their administrative domain to perform their activities. These applications need to interoperate in a consistent manner to make the organisations more productive in this growing marketplace competitiveness [19].

SOA [1] is a software architecture style based on the principles of distributed computing in which the functionality implemented by the applications is made available in the form of services. Services are platform-
independent entities that can have their capabilities described, published, discovered, orchestrated and programmed by using open standards languages and protocols [1] such as eXtensible Markup Language (XML) and HTTP, respectively. With this style, developers can overcome many challenges of distributed computing, including application integration and business agility [20, 21].

The interoperability of SOA-based systems takes place through sets of loosely coupled interfaces that provide a contract between service providers and consumers. To allow both the service provider and the service customer to understand this agreement, they use an Interface Definition Language (IDL) [22] that both parties can understand.

The service provider, service customer, and service broker are the main roles identified in SOA [23], and Fig. 2.1 illustrates their relationship. If a service provider wishes to offer its services to be used by service consumers, it can register (publish) the services in a service broker (repository). Once these services are registered, potential service consumers can search for their desired services and use them as needed.

![Fig. 2.1 Relationship Among Service Broker, Provider and Customer [23].](image)

### 2.1.1 Web Services

Web service [24] is the main technology used to implement SOA-based applications. The use of Web services allows the development of complex applications where independent systems developed on different platforms,
such as Microsoft .NET [25] and Java Platform, Enterprise Edition (Java EE) [26], are able to interoperate via standard Internet protocols. This fact is possible because Web services use standardised technologies, such as HTTP and XML, which can be used on the Internet. The messages exchanged between clients and Web services are defined in XML [27], which is a language broadly adopted in the software industry. The main characteristics of XML are its simplicity and extensibility, which allow human and computer systems to easily write and understand XML-based documents [28].

The World Wide Web Consortium [29] defines Web service as follows:

A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Other software systems can discover its definition. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

Web services can be developed by using two approaches [30]: REpresentation State Transfer (REST) and SOAP. REST can be defined as an architectural style to develop Web services (RESTful services) [31]. It focuses on resources and how to provide access to them. In REST, resources can be represented in XML or JavaScript Object Notation (JSON). The communication protocol to access the resources is implemented by HTTP methods (e.g., GET and POST) and the messages can be represented in many formats, such as XML, JSON and Comma-Separated Values (CSV).

SOAP [6] is an XML-based protocol used to specify the rules to exchange messages in a Web-based environment. Protocols such as HTTP, Transmission Control Protocol (TCP) and Simple Mail Transfer Protocol (SMTP) can be used to transfer SOAP messages. Moreover, SOAP enables synchronous and asynchronous interactions over the Internet, specified in terms of the exchange of XML messages.

A SOAP message consists of a SOAP Envelope that can be divided into two parts: header and body. The header is optional and contains meta-information about the SOAP message required by the application (e.g., authentication and transaction management). The body is required and includes the data input, output or a fault element used to indicate an error.

In SOAP, the interface (contract) of a Web service is defined in WSDL [23], which is an XML-based language used to describe services by specifying how they can be accessed and which operations they support [5]. As illustrated in Fig. 2.2, a Web service interface defined in WSDL can support
multiple operations, which together represent the total service functionality. The service customers access these operations through endpoints. An Endpoint is a unique address identified by a Uniform Resource Identifier (URI). A Web service can have multiple endpoints, for instance, to make the service available using different protocols or at different locations.

![Diagram of Web Service Endpoints](image)

**Fig. 2.2** Web Service Endpoints [23].

A [WSDL](https://www.w3.org/2001/06/wsdl) document is partitioned into two sections, namely abstract service description and a concrete service description. The abstract service description specifies all operations supported by the provided service, the message protocols that can be exchanged with the customers and all abstract data types used to define the messages. The concrete service description section describes where the service is located and how the customers can reach it. This section gives the network addresses and protocols used to reach the service specified in the abstract service description.

Before a service can be used, its interface may be made available to the consumer through a service broker such as a Universal Description, Discovery, and Integration (UDDI) registry [32]. A UDDI registry provides a standardised way to publish and discover information about services. By using the metaphor of phone books, a UDDI business registry consists of white, yellow, and green pages. The white pages provide information about service providers, such as business companies name, address, phone numbers and
2.1 Service-Oriented Architecture

a known identifier. The yellow pages contain more details about business companies by providing a classification of services or businesses, based on standard taxonomies. The green pages contain technical information about services in a way that allows service costumers to bind and use them.

The information of a UDDI registry is defined in XML in terms of four data structures: `businessEntity`, `businessService`, `bindingTemplate` and `tModel` [23], as shown in Fig. 2.3. The information about a company that provides services, such as name, description, and the unique identification of the service provider, is described in a `businessEntity` data structure. The `businessService` data structure contains information about services provided by a business entity. Once a service provider can provide multiple services, each `businessEntity` data structure can have various `businessService` data structures, one for each service provided by that entity. Moreover, each `businessService` data structure can have multiple `bindingTemplate` data structures, one for each access point of the service. The `bindingTemplate` data structure contains technical information about the way that the business service is accessed. It contains the access point for a given service or an indirection mechanism that will lead one to the access point.

A `tModel` is a generic data structure that contains descriptions and pointers to a reusable concept, external technical specifications or taxonomies, for instance, Web service type, a protocol used by Web services and information about the quality of service. Moreover, `tModel` data structure provides a reference system to assist in the discovery of business services.

2.1.2 Service Composition

Services can be provided and reused by different organisations. Furthermore, they can be composed to create more complex services, which are called service compositions [33]. In this thesis, we use the term ’service’ to refer to services in general, and we refer to an ’atomic service’ to refer to a service that is not further composed of other services. Similarly, we use the term ’service composition’ to denote a composition of related services.

Service composition has emerged as an important strategy to enable the collaboration between companies through applications (Business-to-Business) [9]. This collaboration can be achieved by recursively combining
atomic services or service compositions [2]. However, the service concept allows the customers see a service composition as an atomic service without knowing that the service is a combination of existing ones.

Service compositions can be defined as processes that can be automated by Execution engines, also known as Orchestration engines. Execution engines are responsible for performing service compositions, invoking Web services and controlling data input and output of the composition, according to the service composition definitions.

Several languages and graphical notations have been proposed for defining service compositions, such as Web Services Business Process Execution Language (WS-BPEL) [34] and Business Process Model and Notation (BPMN) [35]. WS-BPEL is an XML-based language and it has some
constructs of imperative languages, such as sequential execution, variables, assignment statements, and repetition. Additionally, WS-BPEL describes the interactions between the process and its partners and describes how services are composed. BPMN is a high-level language used by business people to model graphical workflows.

Once service compositions are implemented by combining other services, their quality depends on the quality of each service involved in the composition. For instance, a single insecure or unreliable service can affect the security and reliability of the whole service composition, respectively. Moreover, the levels of security and reliability can also affect the cost of the service composition. In the case of security, for instance, more secure services tend to consume more computational resources (e.g., CPU, memory, and bandwidth) needed to perform activities that enforce security (e.g., encryption and decryption) [36]. In the case of reliability, the failure of a component may demand more resources for the execution of corrective behaviours in the service composition, leading to an increase in the cost of the composition.

To perform and manage a service composition correctly, we must understand the phases of its life-cycle. We based our work on the service life-cycle proposed by [14]. This life-cycle consists of four main phases, namely Planning, Definition, Scheduling and Construction, and Execution (see Fig. 2.4), which can be identified in the most of service composition’s life-cycle proposed in the literature [37, 38, 39].

In the Planning phase, all business interactions, activity sequences, and non-functional requirements (e.g., reliability and cost) are planned, analysed and described by business experts. In the Definition phase, a service composition and its constraints are defined by using techniques such as WSDL [5] and WS-BPEL [34] according to the planned service description. In the Scheduling and Construction phase, one determines how and when the service is supposed to run. In this phase, alternative schedules can be generated and proposed according to the planning, so that they can be chosen before the service execution. After that, the service composition is made ready for execution. In the Execution phase, the service is executed and can be monitored and controlled according to the needs of the service provider to manage its qualities, e.g., to maintain the cost of the service composition below some threshold.
2.2 Business Process Management

Business Process Management (BPM) aims to support the design, management, configuration, enactment, and analysis of Business Processes (BPs) [40]. BPs are usually modelled in workflows like BPMN in order to facilitate the communication among the different stakeholders.

Once modelled, the BP can be analysed, configured, and enacted. The BP functional and non-functional attributes can be analysed, for instance, by using simulation or verification techniques to check whether these attributes are satisfied [40]. After that, the BP is configured in order to be executed by an engine like, e.g., Activiti [41]. While executing, the activities of the BP are controlled by the engine, which is responsible for triggering the process activities according to what is specified in the process description.

2.2.1 Business Process Model and Notation

BPMN is a notation that is readily understandable by business experts [42]. It allows a business expert to specify BPs in a Business Process Diagram
(BPD) that depicts events, activities, gateways, connections, and artifacts, such as data objects.

Table 2.1 Flow Objects [35].

<table>
<thead>
<tr>
<th>Element</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td>〇</td>
</tr>
<tr>
<td>Intermediate</td>
<td>〇</td>
</tr>
<tr>
<td>End</td>
<td>〇</td>
</tr>
<tr>
<td>Activity</td>
<td></td>
</tr>
<tr>
<td>Exclusive</td>
<td>氪</td>
</tr>
<tr>
<td>Parallel</td>
<td>〇</td>
</tr>
<tr>
<td>Inclusive</td>
<td>〇</td>
</tr>
<tr>
<td>Complex</td>
<td>〇</td>
</tr>
<tr>
<td>Gateway</td>
<td></td>
</tr>
</tbody>
</table>

The main element of BPMN is the Flow Object (see Table 2.1), which represents the business process behaviour. There are three types of Flow Objects: Event, Activity, and Gateway. BPMN processes must start and end with Events. Moreover, BPMN processes can have Intermediate Events, which are located between the Start and End Events. The Activity element, as its name suggests, represents an activity of the process and can be a Task or a Sub-process. The entire process flow is controlled by Gateways. With a Gateway, the process flow can be converged or diverged depending on the gateway types, i.e., Exclusive, Parallel, Inclusive and Complex.

A BPMN Task is an atomic activity within a process flow that can have one of the following types: Service, Send, Receive, User, Manual, Business
Rule, and Script (see Fig. 2.5). A Service task (see Fig. 2.5(a)) is performed by a Web service or a software system. When a BP has a task that sends messages to an external participant, this task is represented as a Send task (see Fig. 2.5(b)). When a task waits for a message that can be sent by an external partner, this task is represented as a Receive task (see Fig. 2.5(c)). A task that involves user interaction using a software system is represented by a User task (see Fig. 2.5(d)). When a task is performed without the support of a software system, it is represented as a Manual task (see Fig. 2.5(e)). Tasks can also be performed by scripts written in any scripting language and, in this case, they are represented by Script tasks (see Fig. 2.5(g)). Tasks that provide input to a Business Rule Engine (i.e., a software system that interprets logical rules) are represented by Business Rule tasks (see Fig. 2.5(f)).

Swimlanes are elements that organise the responsibilities in business processes regarding participants and subdivisions, as shown in Table 2.2. A participant is represented by a Pool. The Pool contains the responsibilities of each member, and it may have Lanes, which represent subdivisions to help organise the activities of a process.

Flow Objects can be connected by Connecting Objects, which also provide information about the business process flow (see Table 2.3). A Sequence Flow connects the activities to indicate their sequence in the process. A Message Flow can be used to model the flow of the messages that the participants of the BP exchange. An Association is used to associate data, text and other artifacts with Flow Objects. This element can be used to indicate whether
Table 2.2  Swimlanes Notation [35].

<table>
<thead>
<tr>
<th>Element</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pool</td>
<td>Name</td>
</tr>
<tr>
<td>Lane</td>
<td>Name</td>
</tr>
</tbody>
</table>

a Data Object is being created or read by a Flow Object, depending on the direction of the association, which is represented by an arrow.

Table 2.3  Connecting Object Notation [35].

<table>
<thead>
<tr>
<th>Element</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Flow</td>
<td></td>
</tr>
<tr>
<td>Message Flow</td>
<td></td>
</tr>
<tr>
<td>Association</td>
<td></td>
</tr>
</tbody>
</table>

In BPMN, additional information about the process is provided by Artifacts (Message, Group and Text Annotation), as shown in Table 2.4. A Message represents the contents of the communication between two participants (i.e., Pools). Group is used to group graphical elements in categories, for instance, in order to organise the BP documentation. Another element that can be used for documentation is the Text Annotations, which are used to enrich the process with additional text information.

Processes can produce and consume data during their life-cycle. To represent these data, BPMN provides the graphical elements shown in Table 2.5.

Data objects provide information about input and output of activities. They can model relevant parameters, e.g., user address and credit card number. When Activities store data beyond the scope of the process, a Data Store is used to represent the place where the data are stored.
Table 2.4 Artifact Notation [35].

<table>
<thead>
<tr>
<th>Element</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>![Message Notation]</td>
</tr>
<tr>
<td>Group</td>
<td>![Group Notation]</td>
</tr>
<tr>
<td>Text Annotation</td>
<td>![Descriptive Text]</td>
</tr>
</tbody>
</table>

Table 2.5 Data Object Notation.

<table>
<thead>
<tr>
<th>Element</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>![Data Notation]</td>
</tr>
<tr>
<td>Data Store</td>
<td>![Data Store Notation]</td>
</tr>
</tbody>
</table>

2.3 Cost Management

Nowadays, competition has become quite fierce, and consumers now demand services and products with high quality and low price. As a consequence, the marketplace determines selling prices, and the organisations need to manage their costs and maintain the quality of their products and service if they want to be profitable.

Profit can be defined as the difference between price and cost. A service or product is profitable if its price is higher than its cost. Price is the amount of money paid by the customers to obtain the product and service, and the
cost is the entire expenditure required to create and sell products and services [10].

To maximise profit, one can raise price or reduce costs. However, an increase in price can set back the profit, since the provider may become less competitive in the marketplace and lose its customers to other less expensive providers. However, by reducing costs, services and products can be more profitable, but this also creates opportunities by potentially increasing market penetration.

Cost reduction can be obtained through cost management of services and products. Cost management is a process concerned with planning, estimating, controlling, and decision making about the operational and financial situation of projects of an organisation [12, 13]. Cost management should span the whole project life-cycle by recording, measuring, estimating, analysing, managing, and control costs so that the project can be completed within the approved budget [43].

In the case of service composition, when it is planned, the business process and the required resources (e.g., atomic services) need to be defined. After that, the cost associated with each resource can be defined according to some characteristics of the resources, such as the pricing models of the outsourced services.

The cost estimating allows the prediction of the cost and definition of the budget to perform a service composition. Moreover, providers can use cost estimation to decide on the deployment of the service composition. When the service composition is executed, its cost can be controlled by a software tool that monitors and analyses the actual cost to make corrective actions to minimise this cost.

Cost can be classified as direct and indirect [10, 11, 12]. Direct costs refer to activities or resources directly associated with the production of a product or service, while indirect costs refer to activities or resources that cannot be accurately attributed to each specific product or service.

Cost can also be classified according to its behaviour, which is usually referred to as fixed, variable, mixed and step cost [10, 11, 12]. Fixed cost is a behaviour that remains constant independently from the use of the service within a period, such as a day, month or year, as shown in Fig. 2.6(a). In contrast, variable cost is only computed if the service is used, and its value is directly proportional to the level of use of the service, as shown in Fig. 2.6(b). Factors that cause changes in the cost behaviour are known as
cost drivers. A cost driver is a variable of the activity that changes the cost of the service when the activity is executed, such as the number of invocations and message size.

Fig. 2.6 Classes of Cost Behaviour.

Mixed cost behaviour combines variable and fixed costs. Mixed cost starts with a fixed value that increases proportionally to the use of the service, as shown in Fig. 2.6(c). Finally, Step cost is a class of cost behaviour that can have three alternative patterns:
2.3 Cost Management

- **Pattern 1**: The cost of a unit of a cost driver starts with a particular value and changes when the volume of the cost driver reaches some threshold in a period, as shown in Fig. 2.6(d);
- **Pattern 2**: In a period, the cost is fixed within ranges of use of the service, and after reaching a threshold, the fixed cost is modified to a new value, as depicted in Fig. 2.6(e); and
- **Pattern 3**: A particular cost is defined as a volume unit of a cost driver between the thresholds for a period, as shown in Fig. 2.6(f).

As mentioned, *fixed costs* are constant regardless the number of instances of the service or unit of product. However, *fixed costs* per unit change as the number of instance of the service changes, as shown in Fig. 2.7.

![Cost Per Unit (€)](image)

**Fig. 2.7** Fixed Cost per Instance of Service Composition.

Differently, *variable costs* change the aggregate cost of the service each time the service is invoked. With a *variable cost*, the more the service is invoked, the higher the total cost of the service. However, the cost of an instance of the service stays the same independent of the number invocations in a period, as shown in Fig. 2.8.

An important tool for performing cost management is a costing system. A costing system or cost management system is used to collect, categorise, summarise, and analyse costs to produce helpful information to managers [11]. This system must provide relevant information used to:

- Estimate the cost of services;
- Measure the effect of decisions concerning the service, e.g., analysing the cost after the adaptation of the service;
• Ensure that activities that transform computational resources into services are performed as efficiently as possible;
• Plan the service budget for a period; and
• Identify problems that can make the financial aspects of the service deviate from the planned situation.

These costing systems are based on costing methodologies. Some costing methodologies are available in the industry, such as Activity-Based Costing (ABC) [44], Time-Driven Activity-Based Costing (TDABC) [45] and Resource Consumption Accounting (RCA) [46]. The choice of a costing methodology depends on many factors, such as the characteristics of the company, type of product or service being produced and even the background of the managers [47, 48].

ABC is a costing methodology developed by [44] to overcome the weaknesses of traditional costing methodologies. Traditional costing methodologies that prevailed before the rise of ABC use single indirect cost rate to assign the cost to services and products. This approach makes that methodology simple and not expensive to operate but, as a consequence, the cost information provided by them is not accurate. The general idea of ABC is that the cost to produce services and products come from the activities performed to produce them.

Fig. 2.9 shows the relationships between the resources, activities and cost objects in ABC. A cost object is anything for which costs are accumulated or measured, such as, for instance, product, service and process. First, all activities associated with producing services and products are determined.
by the managers in advance. Then, the resource consumption of these activities is determined for each resource cost driver to define the actual cost of each activity. After the costs of the resources are assigned to the activities, the activities consumption of the cost objects for each activity cost driver is determined to define the cost of the cost objects.

Despite the advantages of ABC over traditional costing methodologies, some issues have hindered the implementation of ABC [49]. Most of these problems are related to the interview with employees to estimate the cost driver rate to perform each activity. For instance, when the employees are asked to inform the time spent on each activity, they tend to ignore their idle or unused time. Moreover, this inaccurate information and changes in the activities and services make it necessary to assess these costs again, e.g., through interviews or surveys, which demand time, resources and money [45]. Additionally, ABC uses a single cost driver rate for each activity and this makes it difficult to model multi-driver activities [49].

To overcome the limitations of ABC, [45] proposed the TDABC, which is a costing methodology that uses time as cost driver without ignoring the concepts of ABC. In this approach, similar resources are assigned to a group (i.e., resource pool), as shown in Fig. 2.10. For each group of resources, TD-ABC estimates the cost per time unit of supplying resource capacity taking into account the practical capacity of the resources. For instance, the practi-
cal capacity of a person can be 80% of the full theoretical capacity, and the practical capacity of a machine can be 85% of its full theoretical capacity. In this case, the cost per time unit of a pool of employees can be determined by dividing the salary of the employees by the practical capacity of the resource pool. Having calculated the cost per time unit of supplying resources, TD-ABC determines the time spent to perform one unit of each activity. Finally, TDABC calculates the cost of one activity by multiplying the cost per time unit by the time spent to carry out the activity, both calculated previously.

Another costing methodology proposed to overcome the difficulties presented in ABC is the Resource Consumption Accounting (RCA). RCA emerged to assist in accounting management. As shown in Fig. 2.11, RCA incorporates characteristics of ABC and ‘Grenzplankostenrechnung’ (GPK), which is a German management accounting practice that focuses on the cost of resources [46].

RCA has three main principles:

1. View of resources;
2. Quantities of consumed resources and their capacity; and

In RCA, resources cause costs, and the cost management must understand the resources and their consumption. In an organisation, resources must be arranged into homogeneous resource pools that can support either another resource pool or the production of a product or service.

In the case of resource consumptions, in RCA they must be expressed through a quantity instead of a percentage. Like TDABC, RCA differs from ABC in the parametrisation of capacity. While in ABC the capacity is specified in the activities, in both RCA and TDABC the capacity is specified in the resource. Moreover, ABC does not recognise idle resources while RCA and TDABC do [50].

The nature of cost means that cost behaviour of resource pools can change as the resources are consumed. Different from ABC and TDABC in which the resource cost behaviour is variable, in RCA the resource cost behaviour can be either fixed or variable.
2.4 Reliability

Reliability is often defined as the probability that a system performs its intended function free of failures within a specified period [51]. The reliability of a service composition can be computed by recursively applying rules on the workflow patterns, such as the sequential, parallel, loop and conditional branching patterns [52, 53]. The use of these patterns facilitates the calculation of the reliability of the entire service composition.

2.4.1 Reliability of Sequential Workflow

Fig. 2.4.1 shows services \( s_i \) ordered sequentially in a process. This pattern can be represented by a service \( s_r \) whose reliability is computed by multiplying the reliability of all services, since failure probabilities are cumulative, as shown in Equation 2.1.

\[
R(s_r) = \prod_{i=1}^{n} R(s_i)
\]  

Fig. 2.12 Sequential Workflow Pattern.

2.4.2 Reliability of Parallel Workflow

When two or more services are executed in parallel, as shown in Fig. 2.13, the sequence of execution cannot be guaranteed at design time. However, like the sequential pattern, all services must be executed correctly in each parallel branch to enable the service composition to continue running. Therefore, the reliability of this workflow pattern, which is represented by a service \( s_r \), can also be computed by multiplying the individual reliabilities of the \( n \) parallel services, as shown in Equation 2.1.
2.4 Reliability

2.4.3 Reliability of Conditional Workflow

In the conditional branching pattern, only one path is completed in a workflow execution. To analyse this pattern, we consider the probability of each path being executed. For instance, in Fig. 2.14, each service has a probability of $w_i$ of being executed in the workflow, and the sum of the probabilities must be 100% since we assume that one service is chosen to be performed, although it may fail.

\[
R(s_r) = \sum_{i=1}^{n} w_i R(s_i) \quad (2.2)
\]
2.4.4 Reliability of Loop Workflow

When a loop pattern has one or more services, these services can be invoked multiple times during a specified process execution, depending on the condition defined in the service composition. Loop systems can be characterised by simple (Fig. 2.15) and dual loop systems (Fig. 2.16).

![Simple Loop Workflow Pattern](image1)

**Fig. 2.15** Simple Loop Workflow Pattern.

![Dual Loop Workflow Pattern](image2)

**Fig. 2.16** Dual Loop Workflow Pattern.

To analyse the loop workflow patterns, we consider the probability that the process moves either forward or backwards to invoke the services in the loop again. Equation 2.3 shows the reliability of simple loop workflows, represented by a service $s_r$. In this equation, $w_i$ is the probability that the workflow reaches the loop and $(1 - w_i)$ is the probability that the workflow leaves the loop.

$$R(s_r) = \frac{(1 - w_i)R(s_i)}{1 - w_iR(s_i)} \quad (2.3)$$
Equation 2.4 shows the reliability of dual loop workflows $R(s_r)$. In this equation, the probability that the process moves forward and backwards to invoke the services in the loop is $(1 - w_j)$ and $w_j$, respectively.

$$R(s_r) = \frac{(1 - w_j)R(s_i)}{1 - w_jR(s_i)R(s_j)}$$

(2.4)

2.5 Concluding Remarks

This chapter presented essential concepts to facilitate the understanding of this work and its associated contributions. Initially, relevant SOA concepts were introduced, and the most important concepts in this area were presented. After that, the BPM initiative and BPMN were briefly introduced. Next, cost and cost management concepts were also detailed. Finally, reliability concepts were introduced.
This chapter presents a survey of existing approaches used to manage the cost of service compositions. The structure of this chapter follows the phases of the service composition’s life-cycle. Hence, it initially describes approaches related to Planning phase, followed by ones associated with Definition and Scheduling and Construction phase, and concludes by describing approaches on Execution phase. Finally, it also presents challenges on cost management of service compositions.

3.1 Planning

As shown in Section 2.1, the Planning phase consists of planning, analysing and describing all business interactions, activity sequences and non-functional requirements such as reliability and cost. In this phase, we should identify and predict situations that can affect the service composition profitability, such as inadequate business process and execution environment. Considering the description of the business process, BPMN has been extended with cost information (e.g., cost intervals and average costs) and the probability of alternative paths to predict the cost of processes [54]. These extensions enable us to analyse alternative activity costs, paths and even external partners before the service is executed.

Further non-functional requirements such as reliability and security can also affect the cost of services. In a business process, unreliable activities
can harm the entire cost of the process [55, 56] because when an activity fails, all tasks performed previously have had their cost accounted. Therefore, for example, unreliable activities should be performed early in the process so that their unreliability can have a smaller effect on the process cost. About security, its cost may be related to the computational resources (e.g., CPU, memory, and bandwidth) needed to perform security activities such as encryption and decryption [36]. For instance, an activity can have its cost modified according to the algorithm used to encrypt its data. Once the costs of others non-functional requirements like security are predicted, the business expert can evaluate their level of relevance for the process taking into consideration the cost required to support these requirements.

In the context of service execution infrastructure, the promise to reduce the cost of investments has motivated service providers to run their services in a cloud environment to decrease their cost. However, sometimes this environment may not be the best option to run a service. At other times, choosing the cloud provider with the lowest cost for a specific application can be a challenging task [57]. In this case, this situation can be predicted by activity cost models, e.g., models to compute the cost of data storage according to data size and storage cost [58]. These cost models allow service providers to simulate some scenarios, compute their cost and decide whether or not to port a service to the cloud [59]. Additionally, this cost can be used together with the cost of the business process to estimate the total cost of the service (business and execution cost).

3.2 Definition

As shown in Section 2.1, services and their cost constraints are precisely stated in the Definition phase according to the planned service composition description. Service interactions can be defined in an orchestration or a choreography along with cost information. For example, Chor [60] is a language to describe service choreography, and it was extended to support the representation of non-functional requirements, such as cost and performance.

Additionally, process algebras have been used to express service orchestrations [61]. Furthermore, some of these techniques have been extended to
allow the representation of cost information in service compositions, such as in e.g., Priced Timed Communicating Sequential Process (PTCSP) [62], Priced Temporal Calculus of Communicating Systems (PTCCS) [63] and Priced Probabilistic Process Algebra (PPPA) [64]. PTCSP extends Timed Communicating Sequential Process (TCSP) by incorporating cost and performance information. PTCCS extends the Temporal Calculus of Communicating Systems (TCCS) with cost information in order to model the price of resource consumption. Finally, PPPA extends the Probabilistic Process Algebra (PPA) to incorporate cost in the service definition, but in a stochastic approach.

3.3 Scheduling and Construction

In the **Scheduling phase**, one determines how and when the service is supposed to run, and which services are used to perform the service composition, as shown in Section 2.1. The selection of the services used in a service composition causes an impact on the composition cost. Hence, several solutions (see Table 3.1) have been proposed to perform this selection by considering functional and non-functional service requirements.

Service compositions with minimal execution cost can be obtained by considering all possible compositions, computing their execution costs and selecting the composition with the lowest execution cost [71]. However, if another non-functional attribute, like reliability [70], also have to be optimised, the complexity of this selection increases.

Services with many tasks performed sequentially in different parallel paths may have paths that spend more time to finish than others, which are called critical paths. These paths need to be scheduled to complete before the user’s deadline and with the minimum execution cost [68].

In a grid environment in which services have different prices and different priorities (the higher the priority, the higher the cost), it is necessary to select an adequate priority to minimise the total costs. The optimisation of the service selection for grid computing tasks can be modelled as a prioritised PERT graph [69], so that all paths and tasks of the graph are analysed to identify the critical path. Once this path is identified, its tasks are prioritised to optimise the execution time. Then, the priorities of the tasks of the
Table 3.1 Automated Service Selection and Scheduling with Cost Requirements.

<table>
<thead>
<tr>
<th>Work</th>
<th>Description</th>
<th>Quality attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[65]</td>
<td>Scheduling algorithm for cost optimization at real-time.</td>
<td>- Cost</td>
</tr>
<tr>
<td>[66]</td>
<td>Heuristic method to minimise the workflow execution cost with the deadline</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>constraints during allocating services to tasks in a grid.</td>
<td>- Performance</td>
</tr>
<tr>
<td>[67]</td>
<td>A cost-gradient metric to provide criteria for service selection and a cost</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>minimisation scheduling strategy with specified deadline.</td>
<td>- Performance</td>
</tr>
<tr>
<td>[68]</td>
<td>QoS-based workflow algorithm that schedules the overall critical path of the</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>workflow such that it completes before the user’s deadline and with the</td>
<td>- Performance</td>
</tr>
<tr>
<td></td>
<td>minimum execution cost.</td>
<td></td>
</tr>
<tr>
<td>[69]</td>
<td>Methodology to optimise service priority selections for a multi-subtask</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>request in a grid network with prioritised services at different prices.</td>
<td>- Performance</td>
</tr>
<tr>
<td>[70]</td>
<td>Economic Objective Function (EOF) that dynamically allocates each service of</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>the composition aware of different price policies and cost models.</td>
<td>- QoS in general</td>
</tr>
<tr>
<td>[71]</td>
<td>An approach to automatically compose services with a minimal execution</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>price.</td>
<td></td>
</tr>
<tr>
<td>[72]</td>
<td>An algorithm that identifies and analyses every possible combination of</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>services that could be used to construct the composite service in a given</td>
<td>- Performance</td>
</tr>
<tr>
<td></td>
<td>time interval by analysing the time-sensitive rules in each services’</td>
<td></td>
</tr>
<tr>
<td></td>
<td>charging schemes to identify sub-intervals of time which the cost of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>invocation of that service composition will not change.</td>
<td></td>
</tr>
<tr>
<td>[73]</td>
<td>A service selection model based on mathematical programming that</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>minimises the expected cost by taken into account complex service charging</td>
<td>- Performance</td>
</tr>
<tr>
<td></td>
<td>models.</td>
<td></td>
</tr>
<tr>
<td>[74]</td>
<td>A solution to create a profitable composition by selecting service using</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>mechanisms like genetic algorithms.</td>
<td>- QoS in general</td>
</tr>
<tr>
<td>[75]</td>
<td>Recursive algorithm to select the best set of services with delay</td>
<td>- Cost</td>
</tr>
<tr>
<td></td>
<td>constraints.</td>
<td>- Performance</td>
</tr>
</tbody>
</table>
3.3 Scheduling and Construction

non-critical paths are relaxed to reduce the total cost and still finish their execution before the critical path.

Additionally, a service can have deadline constraints and can be composed of others services. In this scenario, a scheduler can adjust the reservation schedule for some service of the composition at runtime to compensate the delay of the other services [65]. Another solution anticipates and minimises the workflow execution cost with deadline constraints during the allocation of services [66, 67].

Services can have costs with complex models, e.g., services with different prices according to their configuration [76], or discounts depending on the use of the service within a period of the time. A solution to this issue is to check every possible combination of services that could be used to construct the service composition [72]. First of all, an algorithm can be used to identify all services that can be used to create the service composition and create a set of sub-intervals according to the time interval of interest to run the composition and the charging scheme of each service. Then, the cost of each sub-interval is calculated by adding up the costs associated with each service during that sub-interval. After that, the mean expected cost is calculated. Finally, the combination that has the minimum cost is selected [72]. However, this situation can be more complex when, for example, the discount depends on the number of invocations during a month. In this case, before selecting a service, the solution has to estimate the number of requests expected in this period, compute the estimated discount and then choose the best service [73].

Genetic algorithms can also be used to optimise the service cost-aware selection and other non-functional requirements such as reliability [74], and the number of times the service is invoked, which is often limited by the service provider [77].

Service providers can offer not only single but also composite services, and it can invoke services either from the same provider or different ones [75]. In this case, some solutions need to meet all services invoked directly or indirectly by the composition to minimise their cost. Such solutions analyse all possible services that can fulfil a non-functional requirement. In the case of composite services, each service of this composition is examined recursively [75].
3.4 Execution

In the Execution phase, the service composition is executed and can be monitored and controlled according to the needs of the service composition provider to manage the service qualities. The execution of service composition can take place in environments such as grid and cloud environments. In cloud computing, all resources are provided as services and usually billed on a pay-per-use basis. The cost of running the application can be significantly reduced with no significant impact on the performance when the application is running in the Cloud computing [78]. However, there is no guarantee that cloud consumers can compute the cloud costs programmatically [79], which means that the costs computed by the customer can differ from the cost computed by the cloud provider. Nevertheless, cost models can be used to compute costs at the time of processing, storage, and bandwidth [80] when provided by the cloud provider. These models can also be used to monitor and analyse the costs at runtime by capturing information about the execution time, data transfer and storage per event to ensure that the service cost remains as planned [58]. However, monitoring and adaptation in Execution phase tend to act the service composition performance negatively [73].

Services can also be monitored to prevent violations of Service Level Agreements (SLAs) and be adapted to both keeping them profitable and minimising the total costs that can arise from these violations. However, these adaptations can generate new computing costs, which can increase (instead of decrease) the total cost. Therefore, we should evaluate existing deterministic and heuristic algorithms used to minimise the total costs of the violations and the adaptation used to prevent them [81].

3.5 Discussion

Our research shows that most of the works on cost management of service composition do not take into account all classes of cost behaviours, as shown in Table 3.2. In this case, only [73] takes into account all classes of cost behaviours, but they do not consider different types of cost drivers, such as Byte and time. Furthermore, no one work treats cost throughout the service composition life-cycle.
Additionally, we observed that most of the works on cost management of computational services refer to the *Scheduling and Construction phase*. Moreover, all efforts on these phases are concerned with finding services according to cost and other non-functional requirements.

**Table 3.2** Comparison of the Approaches for Cost Management of Service Composition.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Life-cycle</th>
<th>Cost Behaviour</th>
<th>Consider Different Types of Cost Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>[55]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[56]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[82]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[36]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>Yes</td>
</tr>
<tr>
<td>[60]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[62]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[63]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[83]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[65]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[66]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[67]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[71]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[72]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[73]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[74]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[75]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[74]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td>[81]</td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>No</td>
</tr>
<tr>
<td><strong>Our Objective</strong></td>
<td>X X X X X X</td>
<td>X X X X X</td>
<td>Yes</td>
</tr>
</tbody>
</table>

P= Planning  D= Definition  S&C = Scheduling & Construction  E = Execution  F = Fixed Cost  V = Variable Cost  M = Mixed Cost  S = Step Cost

The keen interest in the *Scheduling phase* reflects the natural concern of computer scientists to develop efficient algorithmic solutions, in particular, algorithms that aim at optimising computational resources, such as processing time. Furthermore, time and cost are non-functional requirements that have a strong relationship with each other. For example, more efficient services tend to be better financially valued. Consequently, the majority of
the papers concern these two non-functional requirements, namely cost and time.

Therefore, our objective is to fill this gap by providing an integrated solution for managing cost throughout the service composition life-cycle by taking into account all classes of cost behaviour and different type of cost drivers.

### 3.6 Key Challenges

As a result of our research, we figured out that some challenges related to cost accounting in computational service are still open. In the *Planning phase*, an open challenge is to predict and analyse business cost taking into account all class of cost behaviour (i.e., fixed, variable, mixed and step cost) and other non-functional requirements, such as reliability and performance. In the *Definition phase*, an open challenge is the absence of notations able to express complex cost models like ones defined by cloud providers. In these models, service cost may depend, for example, on the rate of invocation and amount of data transfer per month.

Once these complex cost models are defined, finding services to perform cost-aware service composition is another challenge. This fact happens because the cost of a service can depend on many variables, such as, for example, the number of invocations or amount of data transfer per month, so that a service can be the best option at some point in time and be the worst choice a time later.

When the service composition is running, we have to compute all costs necessary to run it (e.g., costs of outsourced services and processing), keep the cost within the budget constraints and ensure the service quality remains following the planning. With the service running, one challenge is to monitor the service composition so that adjustments can be made to adhere to the expected cost.

We understand that service development and management need to be incremental and interactive to maximise the profit. For that reason, another current challenge is to treat cost information in all phases of the service composition life-cycle. Furthermore, cost information should be treated in
an integrated way to facilitate, for example, cost tracking throughout the whole life-cycle.

3.7 Concluding Remarks

This chapter presented a survey on available techniques used to manage the cost of service compositions. Initially, it discussed the costing techniques employed in the Planning phase of service composition. Next, it described the techniques applied in the Definition phase. Then, it discussed techniques adopted to manage cost of service composition in the Scheduling and Construction phase. After, it presented the techniques utilised in the Execution phase to manage the cost of the execution of service composition. Finally, it provided a discussion about the works for cost management in computational service.
Part II
Treatment Design
Chapter 4
Cost Behaviour Modelling

This chapter presents a metamodel designed to model service cost behaviours. The chapter also presents examples of models that instantiate our proposed metamodel.

This chapter initially discusses the motivation and requirements for the development of the metamodel. Next, it presents an overview of the metamodel. Then, all the metaclasses of the metamodel are detailed. After that, this chapter introduces an XML Schema created to serialise the cost behaviour models. Finally, it presents examples of cost behaviours modelled with the proposed cost behaviour metamodel and defined in XML.

4.1 Motivation and Requirements

Nowadays, providers offer services with different pricing schemas, such as flat monthly subscription fee, pay-per-use, and a mix of them. The pressure to reduce costs combined with a multitude of different charging schemas have turned service cost management into a challenge to service providers.

In our survey to answer the research question RQ1, we identified open challenges in cost management of service composition, which mainly associated with the phases of the service composition life-cycle, namely Planning, Definition, Scheduling and Construction, and Execution. In particular, a key challenge that concerns all phases is the definition of complex cost models that are suitable to support the whole service composition life-
cycle. To address this challenge, we designed a metamodel to model cost behaviours of computational services meeting the characteristics of SOA.

In industry, most of the resources used to provide services and products are within the administrative domain of the provider. The provider, therefore, is responsible for managing these resources and defining their cost behaviours according to the characteristics and functions of the resources in the production chain. In SOA, however, most of the resources (services) of a service composition are outsourced. Hence, their cost behaviours can be determined by complex pricing models defined by the outsourced service providers, such as ones offered in cloud computing [85]. Additionally, resources in the industry have limited capacity. For example, an employee and a machine can be available to work 40 hours per week, and their full practical capacity can be defined as 80% and 85% of this value, respectively [45]. In SOA, many services can be provided in the Cloud [86], and the elasticity is one of the characteristics of cloud computing, which makes the service’s capabilities appear to be unlimited to customers.

Besides these SOA characteristics, the metamodel was designed taking into account some essential requirements (RM) identified in our research, as follows:

- **RM.1**: Once service providers (e.g., Amazon, Twilio, Shopify and CDYNE) have adopted variable, fixed, mixed and step costs, the cost behaviour metamodel must support heterogeneous cost behaviours (see Section 2.3);
- **RM.2**: The metamodel can be used by different companies around the world. Therefore, it must support different monetary units;
- **RM.3**: Once the service providers can define a specific cost for each service operation, the metamodel must be able to represent different cost model for each service operation;
- **RM.4**: Service providers have adopted different attributes (e.g., size of message and quantity of files) to charge for their services. Therefore, the metamodel must support several types of cost drivers; and
- **RM.5**: An organisation can have more than one service composition using the same service. Therefore, the metamodel must allow to rate the service cost between service compositions.

In the next sections, we describe the cost behaviour metamodel designed by following the requirements as mentioned earlier.
4.2 Overview of the Cost Behaviour Metamodel

Fig. 4.1 shows the proposed cost behaviour metamodel as a MetaObject Facility (MOF) metamodel [87]. A cost behaviour model defines how to compute the cost of a resource (service). The main concept expressed by the metamodel is the cost behaviour (CostBehaviour). Cost behaviour is defined in terms of none or many business attributes (BusinessAttribute), none or many cost drivers (CostDriver) and one or more rules (Rules).

A business attribute (BusinessAttribute) represents an attribute of the service composition (e.g., message or file sent to the service). Since the service can be used for more than one service composition, the business attributes of the cost behaviour model can be mapped to the attributes of many service compositions (Mapping) (RM.5).

As introduced in Section 2.3, cost driver (CostDriver) is a variable that directly affects the cost over a given time. In the metamodel, a cost driver is directly associated with a business attribute (BusinessAttribute) and a set of estimated consumptions (Estimated-Consumption).

The last concept related to the cost behaviour metamodel is the rules (Rules). Rules define how to compute the cost of a service or a specific operation. They are directly associated with none or many cost (Cost), none or many choose (Choose) and none or many upTp (UpTo). However, it must contain at least one of this kind of elements.

Choose is associated with one or many when condition (When) and an otherwise condition (Otherwise). Both when and otherwise conditions are defined with at least one cost (Cost) and bonus (Bonus), which the last one defines a discount that should be applied when calculating the actual cost. UpTo is defined with two or more slices (Slice) and each slice is also associated with one or more cost.

4.3 The Cost Behaviour Definition

The main concept of the metamodel is the root metaclass CostBehaviour. This metaclass is defined in terms of an id, a description, a currency, an effectiveFrom, an effectiveTo, and a resource. Moreover, it has a set of cost
Fig. 4.1 Cost Behaviour’s Diagram.
4.3 The Cost Behaviour Definition

Drivers \((CD)\), business attributes \((BA)\) and rules \((R)\), where \(|CD| > 0\), \(|BA| > 0\) and \(|R| > 0\).

The id represents the identification of the cost behaviour model, which must be unique within an organisational domain to be easily identified in a repository. The effectiveFrom and effectiveTo represent the initial and final validity period of the cost behaviour model, respectively. They are used to prevent that old cost models could be used in the management of the service composition. Another information that should be contained in the cost model of a service is the service identification, that is represented by the attribute resource of the CostBehaviour. The monetary unit adopted in the cost behaviour model is expressed by the currency attribute of the CostBehaviour. The set of values supported by the currency is the currency codes defined in ISO 4217 [88] (RM.2).

A business attribute \((BusinessAttribute)\) represents an attribute of the service composition that directly affects the cost of the service (e.g., message or file sent to the service). The business attribute can be defined in terms of a name, a type and a set of mappings \((MP)\). The name and type represent the name and the type (e.g., Int, Double, and String) of the business attribute in the cost behaviour model, respectively. The mapping maps the business attribute in the cost behaviour model to the name of business attribute \((name)\) used in each service composition \((processId)\) adopted in an organisational domain.

**Definition 1 (Business Attribute):** A business attribute is a tuple \(ba=<name, type, MP>\), where

- **name** is the name of the business attribute,
- **type** is the type of the business attribute, where \(\text{type} \in \{"String","Int","Double","Byte","Milliseconds"\}\), and
- **MP** is a set of mappings. A mapping is a tuple \(mp=<name, processId>\), where
  - **name** is the name of the business attribute adopted in a service composition, and
  - **processId** is the unique identifier of the service composition.

The way that the business attribute affects the cost of the service is defined in the cost driver. Cost driver \((CostDriver)\) is defined in terms of a
business attribute, a unit and a set of estimated consumption (EstimatedConsumption). A unit of a cost driver represents how a business attribute affects the cost of the service (RM.4). For instance, the business attribute file can be used to represent that the cost of a service variates according to the file size as well as the number of files sent to the service. Therefore, in the cost behaviour metamodel these situations are defined by the value of the unit, as follows:

- **Quantity**: cost is driven by the number (quantity) of the business attributes associated with the cost driver. For instance, a cost that varies according to the number of files sent to the service;
- **Int, Double, Byte, Millisecond, Day, Week, Month, and Year**: cost is driven by the value of some measure of the business attribute. For instance, a cost that varies according to the size in bytes of all files sent to the service.

The estimated consumption attribute of the cost driver assists in the cost prediction (RM.1). In the cost management, cost prediction is a task that can be used to select services to perform service compositions. However, depending on the service’s cost behaviour model, estimating the cost of a service depends on the estimation of the value of the cost driver over a period. For instance, in the case of fixed cost, the estimated value of a cost driver allows predicting the cost of an instance of the service. Assuming a service with a fixed cost per month, the estimated cost of an invocation depends on the number of invocations that this service receives from the service composition in a month. In the case of step costs, the estimated values of the cost drivers are used to select the appropriated set of cost functions used to compute the cost of a service.

The estimated consumption of cost drivers is defined taking into account that different engines can manage service compositions in the same administrative domain (RM.5). In this case, a problem related to the estimated consumption of cost drivers is that each engine can assess the use of the services separately without taking into account other ones. Furthermore, services can have several operations, and each operation can have an exact cost (RM.3). For instance, Amazon S3 [89] is a file storage Web service in which the operations GET, PUT and DELETE are priced differently. Therefore, to deal with these cases, the estimated consumption of cost drivers
can be associated with a service composition and an operation of the service.

**Definition 2 (Cost Driver):** A cost driver is a tuple $cd = <ba, EC, u>$, where

- $ba$ is a business attribute used to compute the cost driver,
- $u$ informs how to compute the variation of cost driver, where $u \in \{\text{Quantity}, \text{Int}, \text{Double}, \text{Byte}, \text{Millisecond}, \text{Day}, \text{Week}, \text{Month}, \text{Year}\}$, and
- $EC$ is the set of estimated consumption of the cost driver in a period of time. An estimated consumption is a tuple $ec = <op, p, v, processId>$, where
  - $op$ is the name of a service operation to which the consumption is associated,
  - $p$ is the period of time in which the estimated consumption is computed, where $p \in \{\text{Day} = 1, \text{Week} = 7, \text{Month} = 30, \text{Year} = 365\}$,
  - $v$ is the estimated consumption value, and
  - $processId$ is the unique identifier of the service composition.

The last concept is the rules (Rules). Rules are functions that define how to compute the cost of the service or a service operation. They are defined in terms of an operation, a set of costs (Cost), a set of chooses (Choose) and a set of upTo (UpTo). The attribute operation of Rules must be informed when the cost function is related to a service operation and omitted when the cost function is related to all service operations (RM.3).

**Cost** defines the cost of an amount of a cost driver or the use of the service in a period of time. It is defined in terms of a type (type), a cost driver (costDriver), a period, an amount of the cost driver and a monetary value for the amount of cost driver (valuePerAmount).

**Definition 3 (Cost):** A cost is a function:

$$c(cd, days) = \begin{cases} 
(va/a) \times cdv\text{.value} & \text{if } t \text{ is variable} \\
(va/a) \times (days/p) & \text{if } t \text{ is fixed} 
\end{cases}$$

where
• \( cdv \) is a tuple \( cdv = \langle \text{name}, \text{value} \rangle \), where \text{name} and \text{value} are the name and value of a cost driver \( cd \),
• \( days \) is the number of days to compute fixed cost,
• \( p \) is a cost period, where
\[
\{ \text{Day} = 1, \text{Week} = 7, \text{Month} = 30, \text{Year} = 365 \}\]
• \( a \) is the amount of cost driver,
• \( va \) is the valuer for this amount \( (a) \) of cost driver, and
• \( t \) is the type of cost, where \( t \in \{ \text{variable, fixed} \} \).

When the cost depends on any condition such as in the step cost behaviours, the costs are specified in choose (Choose) or upTp (UpTo) definitions. Choose is defined in terms of a period (period), a set of when conditions (When) and an otherwise condition (Otherwise). Both when and otherwise conditions are defined with a set of costs and bonus (Bonus).

In the case of fixed costs, the rules are defined with a cost specified with the monetary value (valuePerAmount) charged for a certain period (period) of utilisation of the service. In the case of variable costs, the rule is also defined with a cost. But the behaviour of variable costs is defined in terms of a value (valuePerAmount) charged for an amount (amount) of a cost driver (costDriver). The cost driver specified in variable cost is related to the set of cost drivers \( \text{CostDrivers} \) declared in the cost behaviour model. Mixed costs, however, combine fixed and variable costs. In these cases, rules consist of a conjunction of multiple costs.

Section 2.3 introduced three patterns of step costs. In the case of Patterns 1 and 2, the cost of the resource is computed by a set of cost functions that must be chosen according to the volume of a cost driver consumed in a period. To define these patterns, one has to specify a condition (logical expression) that determines the cost behaviour when it holds. A logical expression consists of logical, numeric or relational operators defined in When definition contained in the Choose definition. If the logical condition holds, the costs defined in the when condition have to be considered. Otherwise, a set of default cost functions defined in Otherwise condition is used. Besides cost functions, When and Otherwise can also have a bonus (Bonus), which defines a discount that should be applied when calculating the actual cost.

Definition 4 (When): A when is a function:
4.3 The Cost Behaviour Definition

\[
wh(CDV, \text{days}) = \begin{cases} 
\sum_{i=1}^{n} c_i(cdvi, \text{days}) - b & \text{if condition is 1} \\
\text{null} & \text{if condition is 0}
\end{cases}
\]

where

- \( CDV \) is a set of cost driver values \( cdv \),
- \( \text{days} \) is the number of days to compute fixed cost,
- \( c_i \) is the costs defined in the \( \text{when} \),
- \( b \) is a value that represents a bonus, and
- \( \text{condition} : \ CDV \rightarrow B \), where \( B = \{0, 1\} \).

**Definition 5 (Otherwise):** An otherwise is a function:

\[
ow(CDV, \text{days}) = \sum_{i=1}^{n} c_i(cdvi, \text{days}) - b
\]

where

- \( CDV \) is a set of cost driver values \( cdv \),
- \( \text{days} \) is the number of days to compute fixed cost,
- \( c_i \) is the costs defined in the \( \text{when} \), and
- \( b \) is a value that represents a bonus.

**Definition 6 (Choose):** A choose is a function:

\[
ch(CDV, \text{days}) = \begin{cases} 
wh_1(CDV, \text{days}) & \text{if } wh_1(CDV, \text{days}) \text{ is NULL} \\
wh_2(CDV, \text{days}) & \text{if } wh_1(CDV, \text{days}) \text{ is NULL} \\
\cdots & \text{if } wh_{n-1}(CDV, \text{days}) \text{ is NULL} \\
wh_n(CDV, \text{days}) & \text{if } wh_{n-1}(CDV, \text{days}) \text{ is NULL} \\
ow(CDV, \text{days}) & \text{if } wh_n(CDV, \text{days}) \text{ is NULL}
\end{cases}
\]

where

- \( CDV \) is a set of cost driver values \( cdv \),
- \( \text{days} \) is the number of days to compute fixed cost,
- \( wh_i \) is a function according Definition 4, and
- \( ow \) is a function according Definition 5.
In case of Pattern 3, the total volume of a cost driver consumed in a period of time may be sliced, and each slice (Slice) can have a specific cost. Contrarily to Patterns 1 and 2 that use only a set of costs defined in either a When or the Otherwise, in Pattern 3 more than a set of cost functions defined in many slices of the total volume of a cost driver can be selected to compute the cost of the service.

Fig. 4.2 Illustrative Step Cost Pattern 3.

Fig. 4.2 illustrates Pattern 3 of step cost using the concepts of the cost behaviour metamodel. In this case, the cost behaviour model defines $n$ slices ($sl_1$, $sl_2$, ..., $sl_n$) of a cost driver, and for each slice a set of cost functions ($C_1$, $C_2$, ..., $C_n$) is defined. In this example, $x$ represents the real volume of the cost driver consumed in a period of time. Therefore, the sets of cost functions $C_1$ and $C_2$ are selected and their costs are computed based on the cost driver values $|v_1 - 0|$ and $|x - v_1|$, respectively.

**Definition 7 (Slice):** A slice is a function:

$$sl(CDV, days) = \sum_{i=1}^{n} c_i(min(cdvi, a), days)$$

where

- $CDV$ is a set of cost driver values $cdv$,
- $a$ is a amount of cost driver defined in the slice,
- $days$ is the number of days to compute fixed cost,
4.3 The Cost Behaviour Definition

- \( c_i \) is the costs \( c \) defined in the slice, and
- \( \min(cdv, a) \) is a function that return the minimum value between \( cdv \) and \( a \).

**Definition 8 (UpTo):** An upTo is a function:

\[
up(CDV, \text{days}) = \sum_{i=1}^{n} sl_i(CDV'_i, \text{days})
\]

where

- \( CDV \) is a set of cost driver values \( cdv \),
- \( \text{days} \) is the number of days to compute fixed cost,
- \( sl_i \) is the slices defined in the upTo, and
- \( CDV'_i \) is a set of cost driver values sliced according the amount defined in slice \( sl_i \).

**Definition 9 (Rules):** rules is a function:

\[
r(CDV, \text{days}) = \sum_{i=1}^{m} c_i(CDV, \text{days}) + \sum_{j=1}^{n} up_j(CDV, \text{days}) + \sum_{k=1}^{p} ch_k(CDV, \text{days})
\]

where

- \( CDV \) is a set of cost driver values \( cdv \),
- \( \text{days} \) is the number of days to compute fixed cost,
- \( c_i \) is a cost functions defined in the rules,
- \( ch_j \) is a choose defined in the rules, and
- \( up_k \) is an upTo defined in the rules.

**Definition 10 (Cost Behaviour):** A cost behaviour is a tuple \( cb = \langle id, desc, cur, ef, et, BA, CD, R > \rangle \), where

- \( id \) is the unique identifier of the cost behaviour;
- \( desc \) is a text summary about the cost behaviour;
- \( cur \) is the currency where the cost behaviour is defined;
- \( ef \) is the initial validity period of the cost behaviour;
- \( et \) is the final validity period of the cost behaviour;
- \( BA \) is the set of business attributes used to compute the service cost;
- \( CD \) is a set of cost drivers, and
- \( R \) is a set of rules.
4.4 Metamodel Serialisation

Fig. 4.3 shows an XML Schema Definition (XSD) of the proposed cost behaviour metamodel created to serialise the cost behaviour models.

Fig. 4.3 XML Schema of the Cost Behaviour Metamodel.

In the XSD, all metaclasses of the cost behaviour metamodel are represented as complex types. The main element of the cost behaviour XSD is the costBehaviour. This element is a CostBehaviour complex type that has
the same attributes of the CostBehaviour metaclass. Moreover, it has one or more rules, a businessAttributes and a costDrivers elements of Rules, BusinessAttributes, CostDrivers complex types, respectively.

The Rules complex type represents the Rules metaclass of the metamodel. This complex type has sequences of none o more cost, none or more choose and none or more upTp elements of Cost, Choose and UpTo complex types, respectively.

The complex type Cost represents the Cost metaclass and has a type and a period as attributes. Moreover, it has a sequence of none or one description, none or one costDriver, an amount and a valuePerAmount elements that represent the respective attribute of the Cost metaclass. The Choose and UpTo represent the metaclasses of the metamodel with the same name and attributes.

The BusinessAttributes and the CostDriver complex types were created to declare the set of businessAttribute and costDriver of the metaclass, respectively. Therefore, the BusinessAttributes complex type has a sequence of one or more BusinessAttribute elements and the CostDrivers complex type has a sequence of one or more CostDriver elements.

The entire cost behaviour metamodel with all elements and attributes can be seen in Appendix B.

4.5 Metamodel Instance Examples

In order to demonstrate the use of the proposed metamodel, we define the cost behaviours of the following services: PhoneMessageService, ABCEmailService, YAZEmailService, ValidateEmailService and ConvertWAVtoMP3Service. The cost behaviours of these services can be seen in Table 4.1. They are expressed in XML format in which instances of the metamodel are serialised.

The PhoneMessageService sends short and multimedia messages. For each kind of message, this service has a specific operation (i.e., sendSMS and sendMMS) and each operation has a specific variable cost that depends on the number of messages sent. The ABCEmailService sends e-mail messages with attachments and has a variable cost that depends on the number of e-mail messages sent by the service user and a step cost behaviour (pattern
### Table 4.1 Examples of Cost Model.

<table>
<thead>
<tr>
<th>Service</th>
<th>Cost Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneMessageService</td>
<td>SMS: €0.0075 per message [1] MMS: €0.02 per message [2]</td>
</tr>
<tr>
<td>ABCEmailService</td>
<td>€0.10 per 100 emails sent / month; and [3] Data Transfer: [4]</td>
</tr>
<tr>
<td></td>
<td>First 10 GB/month: €0.12/MB,</td>
</tr>
<tr>
<td></td>
<td>Next 10 GB/month: €0.10/MB,</td>
</tr>
<tr>
<td></td>
<td>Over 20 GB/month: €0.8/MB.</td>
</tr>
<tr>
<td>YAZEmailService</td>
<td>€10.00 per month.</td>
</tr>
<tr>
<td>ValidateEmailService</td>
<td>€2.00 per month; and [5] Email address: [6]</td>
</tr>
<tr>
<td></td>
<td>If less than or equal to 1000 email: €0.050/email address,</td>
</tr>
<tr>
<td></td>
<td>If more than 1000 email: €0.045/email address.</td>
</tr>
<tr>
<td>ConvertWAVtoMP3Service</td>
<td>€20.00 per month; and [7] €0.01 per 1 MB of file.</td>
</tr>
</tbody>
</table>

3) that depends on the size of the message attachments. The YAZEmailService is a simple e-mail service that sends e-mail without attachment and has a fixed cost. The ValidateEmailService is used to validate e-mail addresses and has a fixed cost behaviour and a step cost behaviour (Pattern 1) that depends on the number of e-mails evaluated in a month. The ConvertWAVtoMP3Service converts WAV to MP3 audio format, and it has a mixed cost, i.e., a fixed cost with a variable cost.

The cost behaviour of PhoneMessageService is depicted in Listing 4.1. The cost functions of operations sendSMS and sendMMS are based on the cost driver numberOfMessage, defined in Lines 7-10. Moreover, it is defined a specific estimated consumption for operations sendSMS and sendMMS in Lines 8-9, respectively, in order to allow cost prediction for this service. The cost function of each operation is declared separately. The cost of operation sendSMS is defined in Lines 12-18 and the cost of operation sendMMS is defined in Lines 19-25.
4.5 Metamodel Instance Examples

Listing 4.1 Cost Behaviour of PhoneMessageService.

```xml
<costBehaviour id="PhoneMessageCM" currency="EUR"
    resource="PhoneMessageService">
    <businessAttributes>
        <businessAttribute name="MESSAGE" type="byte"/>
    </businessAttributes>
    <costDrivers>
        <costDriver name="numberOfMessage" unit="quantity"
            businessAttribute="MESSAGE">
            <estimatedConsumption operation="sendSMS"
                value="2000" period="month"/>
            <estimatedConsumption operation="sendMMS"
                value="400" period="month"/>
        </costDriver>
    </costDrivers>
    <rules operation="sendSMS">
        <cost type="variable">
            <costDriver>numberOfMessage</costDriver>
            <amount>1</amount>
            <valuePerAmount>0.0075</valuePerAmount>
        </cost>
    </rules>
    <rules operation="sendMMS">
        <cost type="variable">
            <costDriver>numberOfMessage</costDriver>
            <amount>1</amount>
            <valuePerAmount>0.02</valuePerAmount>
        </cost>
    </rules>
</costBehaviour>
```

The cost behaviour of YAZEmailService, which is only defined as €10.00 per month, is expressed in Listing 4.2. It has a cost driver `numberOfEmailSent` with estimated consumption (Lines 7-9) that can be used to predict the cost of this service. Its cost function is set in Lines 12-16. The charging period of the fixed cost is defined in Line 12. Lines 14-15 determine the value charged for the amount of this period, which is €10.00 for each month the service is used.
Listing 4.2 Cost Behaviour of YAZEmailService.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<costBehaviour id="YAZEmailCM" currency="EUR"
  resource="YAZEmailService">
  <businessAttributes>
    <businessAttribute name="ID" type="string"/>
  </businessAttributes>
  <costDrivers>
    <costDriver name="numberOfEmailSent" unit="quantity"
      businessAttribute="ID">
      <estimatedConsumption value="500.0"
        period="month"/>
    </costDriver>
  </costDrivers>
  <rules>
    <cost type="fixed" period="month">
      <description>EUR10.00 per 1 month</description>
      <amount>1.0</amount>
      <valuePerAmount>10.00</valuePerAmount>
    </cost>
  </rules>
</costBehaviour>
```

Listing 4.3 shows the cost behaviour of ABCEmailService. Lines 7-12 define the cost drivers `numberOfEmail` and `sizeOfAttachment` of this cost model that are related to business attributes `DESTINATION` and `ATTACHMENT` (Lines 4-5), respectively. The cost behaviour is defined in Lines 13-42. Lines 14-18 have a variable cost definition, in which the cost varies according to the `numberOfEmail` cost driver (Line 15) that can be obtained from an execution engine. Lines 19-41 define the step cost behaviour (pattern 3) of this service, in which the cost varies according to the volume of cost driver `sizeOfAttachment`. The cost function of the first 10 Gigabytes of the volume of the cost driver is determined in Lines 20-26, and the cost function of the next 10 Gigabytes is defined in Lines 27-33. Over 20 Gigabytes of attachment in a month, the cost is computed according to Lines 34-40. The sizes of the first and second slices of the volume are defined in Lines 20 and 27, respectively, and the size of the third slice is not informed because it is the last slice and its size is infinite.
Listing 4.3 Cost Behaviour of ABCEmailService.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<costBehaviour id="ABCEmailCM" currency="EUR"
    resource="ABCEmailService">
  <businessAttributes>
    <businessAttribute name="DESTINATION" type="string"/>
    <businessAttribute name="ATTACHMENT" type="float"/>
  </businessAttributes>
  <costDrivers>
    <costDriver name="numberOfEmail" unit="quantity"
        businessAttribute="DESTINATION"/>
    <costDriver name="sizeOfAttachment" unit="float"
        businessAttribute="ATTACHMENT"/>
  </costDrivers>
  <rules>
    <cost type="variable">
      <costDriver>numberOfEmail</costDriver>
      <amount>100</amount>
      <valuePerAmount>0.10</valuePerAmount>
    </cost>
    <upTo costDriver="sizeOfAttachment" period="month">
      <slice amount="10737418240">
        <cost type="variable">
          <costDriver>sizeOfAttachment</costDriver>
          <amount>1048576</amount>
          <valuePerAmount>0.12</valuePerAmount>
        </cost>
        </slice>
    </upTo>
  </rules>
</costBehaviour>
```
As shown in Listing 4.4, ConvertWAVtoMP3Service has mixed cost, i.e., a combination of fixed and variable cost. The fixed cost is defined in Lines 10-13, and the variable cost, which varies according to cost driver fileSize, is defined in Lines 14-18. This cost driver is defined in Line 7, which is associated with a business attribute defined in Line 4.

**Listing 4.4 Cost Behaviour of ConvertWAVToMP3Service.**

```xml
<slice>
  <cost type="variable">
    <costDriver>sizeOfAttachment</costDriver>
    <amount>1048576</amount>
    <valuePerAmount>0.08</valuePerAmount>
  </cost>
</slice>
<upTo>
</rules>
</costBehaviour>
```
The cost behaviour of `ValidateEmailService` is shown in Listing 4.5. Lines 6-9 define the cost driver of this model. The roles of this cost behaviour model are defined in Lines 10-31. Lines 11-14 have a fixed cost definition. Lines 15-30 describe a step cost behaviour, in which the cost varies according to the given condition. Cost variation is defined in a `when` (Lines 16-22) clause with a possible `otherwise` (Lines 23-29) clause. In case the guard of `when` clause is true, the cost associated with this clause is computed. Otherwise, the cost associated with the `otherwise` clause is computed.

**Listing 4.5** Cost Behaviour of ValidateEmailService.

```xml
<costBehaviour id="ValidateEmailCM" currency="EUR"
    resource="ValidateEmailService">
    <businessAttributes>
        <businessAttribute name="EMAIL_ADDRESS" type="string"/>
    </businessAttributes>
    <costDrivers>
        <costDriver name="numberOfEmailSent" unit="quantity"
            businessAttribute="EMAIL_ADDRESS"/>
    </costDrivers>
    <rules>
        <cost type="fixedCost" period="month">
            <amount>1</amount>
            <valuePerAmount>2.00</valuePerAmount>
        </cost>
        <choose period="month">
            <when condition="numberOfEmailSent &lt;= 1000">
                <cost type="variable">
                    <costDriver>numberOfEmailSent</costDriver>
                    <amount>1</amount>
                    <valuePerAmount>0.05</valuePerAmount>
                </cost>
            </when>
            <otherwise>
                <cost type="variable">
                    <costDriver>numberOfEmailSent</costDriver>
                    <amount>1</amount>
                    <valuePerAmount>
```


```xml
    <valuePerAmount>0.045</valuePerAmount>
</cost>
</otherwise>
</choose>
</rules>
</costBehaviour>
```

### 4.6 Concluding Remarks

This chapter presented a metamodel to model the cost behaviour of computational services. With this metamodel, we intend to allow the integration of existing approaches that consider cost in service composition, to support different cost behaviours throughout the phases of service composition life-cycle. This integration helps business and service experts plan, control and take a decision about the financial situation of service compositions.

Initially, this chapter presented the motivation and requirements of the proposed cost behaviour metamodel. Next, it showed an overview of the metamodel and the definition of the elements of the metamodel. After that, it presented an XSD to serialise the cost behaviour models. Finally, it showed examples of cost behaviour models expressed in XML by using the proposed cost behaviour metamodel.
This chapter presents the techniques we developed to analyse the cost of service compositions in planning and execution phases of its life-cycle.

This chapter initially discusses the motivation and requirements for the development of the techniques. Next, it presents an overview of the techniques. Then, this chapter describes two techniques to analyse the cost of service compositions in the planning phase, and two techniques to compute the cost of services when the composition is already in execution to compare the cost of candidate services.

5.1 Motivation and Requirements

Cost analysis is a task of cost management responsible for computing the cost of atomic services and the entire service composition. With the increasing number of Web services having similar functionality but different pricing schemes, choosing the best set of services with the lowest cost has been a challenge in service composition.

In our survey on cost management in service-oriented computing presented in Chapter 3, we identified techniques to compute the cost of service compositions [73, 56]. However, most of them consider cost as a fixed amount per invocation without taking into account all classes of cost behaviour and different unit of cost driver (e.g., Byte and time).

To overcome these challenges, we provide techniques to analysis cost of atomic services and service composition taking into account the different
classes of cost behaviour described in Section 2.3. The primary requirement of the development of these techniques is to compute service cost considering the cost behaviour metamodel defined in Chapter 4.

5.2 Overview

In the Planning phase of the service composition’s life-cycle, business experts need to work out the compositions and establish goals (e.g., service reliability, and budget) and specify how to achieve them [90]. Furthermore, the business experts have to predict and analyse costs to take decisions that can increase the service composition’s profit [23]. In this phase, the atomic services can not be contracted yet and, therefore, the cost analysis can be used to choose the more appropriated services for the service composition taking into account the characteristics of the service composition and the cost behaviours of the services.

In Execution phase, however, we assume that candidate services have their contracts and SLAs established before the execution of the service composition. Once the services are already contracted, all fixed costs are independent of the use of the composition. This fact means that if a service with a fixed cost is contracted, its cost is accounted whether it is used or not. For example, given two candidate services ($s_a$ and $s_b$) able to perform a task, where $s_a$ has variable cost and $s_b$ has fixed cost behaviour, the choice of $s_a$ (variable cost) increases the total cost of the composition every time it is invoked. Differently, the choice of $s_b$ (fixed cost) does not change the cost of the entire composition when it is invoked because the fixed cost of $s_b$ was already associated with the cost of the service composition.

Taking into account these characteristics, we do not consider costs that do not depend on the number of instances when comparing service cost in the execution phase. Fig. 5.1 depicts the difference between the original cost behaviours and the ones in the execution phase. In execution phase, fixed cost is completely despised. In the case of mixed cost, only the variable cost must be computed. For Pattern 2 of step cost, in which the fixed cost is modified according to the use of the service, the initial fixed cost is subtracted from all fixed costs, making the initial fixed cost equal to zero. The other classes of cost behaviour remain unchanged.
5.2 Overview

The first and second techniques described in next sections are provided to analyse the service composition cost in the planning phase. The first one is an analytical approach to predict the cost based on annotations made on the service composition. The second technique computes the cost of the service compositions by using execution logs obtained from their simulation or real execution.

The third and fourth techniques presented in next sections are provided to analyse and compare costs of services in the execution phase. The third technique uses event logs to compute the service costs. It allows execution engines to monitor and adapt service composition in parallel to its execution.
by using historical data. The fourth technique uses business attributes of one specific instance of the service composition. It aims to allow execution engines to select the most appropriate service to perform each instance of the service composition by taking into account the business attributes (e.g., the size of message and number of files) of these instances.

5.3 Cost Analysis Using Annotations in Planning Phase

Business cost is defined as the cost incurred to perform all activities of a process correctly to achieve its goals [56]. In BPM, the business cost can be computed by dividing the cost incurred to perform all tasks by the reliability of the entire business process [56]. Since a service composition is a process composed of multiple services provided by different service providers [91], its business goals are achieved when the invoked services are performed correctly. Therefore, as consequence of this aggregation, the business cost of a service composition is computed by dividing the cost incurred to invoke all services by the reliability of the entire composition, which is determined by the reliability of the aggregated services.

In this technique, the cost of the service composition is computed by using Algorithm 1. This algorithm receives as input the period of analysis (periodOfAnalysis), the average number of executions (n) of the service composition over the period of analysis, and the service composition (ASC) annotated with the following data:

- The reliability of each task;
- The identification of each service with its operation that will perform each task;
- The average values of cost drivers spent by each task in executions over a period; and
- The probability of each alternative path.

Algorithm 1 assumes that the cost of a service is accounted only if the service is performed correctly [82]. Hence, the cost of a task \( t_j \) is only computed if the task executes and terminates successfully. Therefore, it calculates the business cost of the service composition based on the average values of the cost drivers associated with the cost behaviour of the services.
In its turn, the average values are computed taking into account the probability that the service terminate correctly \( PCE \) during \( n \) executions of the service composition and the values of the cost drivers \( (acd_{ji}) \) annotated in the service composition. The probability of correct execution \( (PCE) \) of a task \( (t_j) \) is computed according to Equation 5.1, where \( t_j \) is a task with expected reliability \( rb_j \), \( t_i \) is the task executed before \( t_j \), and \( w_{ij} \) is the probability that \( t_j \) is executed after \( t_i \).

**Algorithm 1** Compute the Cost of the Service Composition Using the Annotated Service Composition \( ASC \), a period of analysis \( periodOfAnalysis \) and the number of executions \( n \) in this period of analysis.

1: procedure **COMPUTECOSTDRIVERVALUE**\((ASC, periodOfAnalysis, n)\)
2: \( CB.CDV := empty \) \( \triangleright \) A set of cost driver values per cost behaviour
3: \textbf{for all} tasks \( t_j \) in \( ASC \) \textbf{do}
4: \( cb_j := \text{getCostBehaviour}(t_j, getService()) \)
5: \( op_j := t_j, getOperation() \)
6: \( ACD_j := t_j, getCostDrivers() \)
7: \textbf{for all} \( acd_{ji} \) in \( ACD_j \) \textbf{do}
8: \( \text{value} := PCE(t_j) \times acd_{ji}.value \times n \) \( \triangleright PCE(t_j) \) Equation 5.1
9: \( CB.CDV.add(cb_j, op_j, acd_{ji}.name, value) \)
10: \textbf{end for}
11: \textbf{end for}
12: \textbf{for all} \( cb_j \) annotated in \( ASC \) \textbf{do}
13: \( cost := 0.0 \)
14: \textbf{for all} rules \( r_{ji} \) in \( cb_j \) \textbf{do}
15: \( CDV := CB.CDV, \text{getCDV}(cb_j, t_{ji}, op) \)
16: \( cost := cost + r_{ji}(CDV, periodOfAnalysis) \) \( \triangleright \) Definition 9.
17: \textbf{end for}
18: \( ServiceCost.add(cb_j, resource, cost) \)
19: \textbf{end for}
20: \textbf{return} ServiceCost
21: end procedure

\[
PCE(t_j) = \begin{cases} 
PCE(t_i) \times w_{ij} \times rb_j & \text{if } t_j \text{ is NOT the first task} \\
rb_j & \text{if } t_j \text{ is the first task}
\end{cases}
\]  

(5.1)
As presented in Section 2.4, when a task \( t_j \) is preceded by a set of tasks in sequence, in parallel, having a condition or in a loop, Equations 2.1-2.4 can be applied to replace the reliability of the set of tasks with a single reliability value in order to compute \( PCE(t_j) \).

Since the same service can be used to perform one or more tasks and the operations of a service may have different cost models, the average value of the cost driver is stored in a set of cost driver values per cost behaviour and service operation \( (CB, CDV) \), where each element is represented by tuple \( (cb_{cdv} = \langle key, value >) \). In this tuple, \( key \) is defined in terms of the cost behaviour model \( (cb_j) \), operation \( (op_j) \) and cost driver identification \( (acd_{ji}, name) \), and \( value \) is computed by multiplying the value of the cost driver annotated in \( t_j \) by the \( PCE(t_j) \) of the respective task (Line 8). When a new \( cd_{cdv} \) needs to be stored in \( CB, CDV \) and its \( key \) already exists, its \( value \) is summed up to the existing \( cb_{cdv} \). Otherwise, a new tuple is added to \( CB, CDV \).

After computing the average values of the cost drivers, Algorithm 1 computes the cost of the service by calling all the rules (Definition 9) defined in the cost behaviour (Line 16). The rules receive as input the period of analysis \( (periodOfAnalysis) \) and a set of cost driver values \( (CDV) \). Recall that \( CDV \) is a set of a tuple \( cdv = \langle name, value >, where name and value are the name and value of a cost driver cd. CDV is obtained by getting all cost driver values that match the operation defined in the rule \( (r_i) \) (Line 15), as follows:

- If the rule \( (r_i) \) is associated with an operation, only the cost driver values associated with the cost behaviour \( cb_j \) and the operation of \( r_i \) are extracted from \( CB, CDV \); or
- If the rule \( (r_i) \) is NOT associated with an operation, all cost driver values associated with the cost behaviour \( cb_j \) are extracted from \( CB, CDV \).

After computing the cost of each service, Algorithm 1 puts it in a set of service cost values \( (ServiceCost) \). In \( ServiceCost \), each element is represented as a tuple \( < s_j, cost_j >, where s_j is a service and cost_j is the cost of s_j. Finally, with these expected costs of the services computed, the expected business cost of \( n \) instances of the service composition in a period of analysis is calculated according to Equation 5.2 [56]. This equation shows
that the more unreliable the service composition is, the higher is its business cost.

\[ \text{totalCost} = \sum_{j=1}^{n} \frac{\text{serviceCost}_{j}}{rb_{jc}}, \text{ where cost}_{j} \geq 0 \text{ and } 0 < rb_{jc} \leq 1 \quad (5.2) \]

5.4 Cost Analysis Using Event Logs in Planning Phase

In this section, we present a technique to compute the cost of service composition by using event logs (LOG) obtained from either the real execution or simulation of service compositions.

An example of an event log with cost attributes for cost analysis is shown in Table 5.1. This example is based on a fragment of event logs presented in [92]. In Table 5.1, each line presents an event of an instance of the service composition. Each event is associated with a process (Process id), an instance of the process (Case id), an activity of the business process (Activity), a resource (Resource) and a set of cost attributes (Cost Attributes). The resource represents the service that performed the task, and it is composed by the identification of the service and its operation used to perform the activity (service id:operation). With this information, it is possible to identify the correct cost function in the cost behaviour of the service that performed each activity.

In the event logs, Cost Attribute represents the attributes present in the cost behaviour model of the service and are used to compute the cost driver values (CDV) and, consequently, the service costs.

Using the event logs (LOG), the cost of the service composition is computed by Algorithm 2, which the input is the execution log (LOG) and the period in which the execution log was extracted (periodOfAnalysis). In LOG, each register (reg_k) must have at least the service identification (s_k), the service operation (op_k) and the list of attributes (acd_kw) associated with the cost drivers (CD_k) of the service’s cost behaviour. For each register (reg_k) in the event logs (LOG), the algorithms get the service operation (op_k) used to perform the task (Line 5), and the cost behaviour model (cb_k) of the service (Line 6).
The cost behaviour model has a set of cost drivers ($CD_k$), which the values are computed by using the business attribute values ($ba_{kw}$) registered in $LOG$. The values of the cost drivers are calculated according to the value of the unit ($u$) of the respective cost driver according to the following rules:

- If the unit of the cost driver ($cd_{kw}$) is $Quantity$, the added value is 1,
- If the unit of the cost driver ($cd_{kw}$) is NOT $Quantity$, the added value is $acd_{ji}$.

The values of the cost drivers are added to a set of cost driver values per cost behaviour and operation ($CB.CDV$). Recall that $CB.CDV$ is a set of cost driver values in which each element is represented as a tuple <key,value>, where key must be unique and is defined in terms of the cost behaviour ($cb_k$), operation ($op_k$) and cost driver identifications ($acd_{ji}.name$). Since the cost driver values must be accumulated for the period of analysis, when a new tuple needs to be stored in $CDV$, and its key already exists in $CB.CDV$ its value is added to the existing one. Otherwise, a new tuple is added to $CB.CDV$.

After computing all cost driver values ($CB.CDV$) based on the event logs ($LOG$), Algorithm 2 extracts the cost driver values ($CDV$) from the $CB.CDV$ by getting all cost driver values that match the operation of the rule ($r_{ji}$) (Line 20). If the rule ($r_{ji}$) is not associated with an operation, all
Algorithm 2 Compute Cost of Service Composition Using LOG and a period of analysis periodOfAnalysis.

1: procedure COMPUTECOSTPERLOG(LOG, periodOfAnalysis)
2:     \(CB_CDV := \text{empty} \quad \triangleright \text{A set of cost driver values per cost behaviour}
3:     \text{for all register reg}_k \text{ in } LOG \text{ do}
4:         s_k := reg_k.getResource()
5:         op_k := reg_k.getOperation()
6:         cb_k := s_k.getCostBehaviour()
7:         CD_k := cb_k.getCostDrivers()
8:         \text{for all } cd_{kw} \text{ in } CD_k \text{ do}
9:             ba_{kw} := reg_k.getAttribute(cd_{kw}.getBusinessAttribute(reg_k.getProcessId()))
10:            if unit u of \(cd_{kw}\) is Quantity then
11:                \(CB_CDV.add(cb_k,op_k,ba_{kw}.name,1)\)
12:            else
13:                \(CB_CDV.add(cb_k,op_k,ba_{kw}.name,ba_{kw}.value)\)
14:            end if
15:     end for
16:     \text{for all } cb_j \text{ in } CB_CDV \text{ do}
17:         cost := 0.0
18:         \text{for all rules } r_{ji} \text{ in } cb_j \text{ do}
19:             CDV := CB_CDV.getCDV(cb_j,r_{ji},op)
20:             \text{cost} := cost + r_{ji}(CDV, periodOfAnalysis) \quad \triangleright \text{Definition 9.}
21:         end for
22:     end for
23:     ServiceCost.add(cb_j,resource,cost)
24: end procedure

cost driver values associated with the cost behaviour \(cb_j\) are extracted from \(CB_CDV\). Otherwise, only the cost driver values associated with the cost behaviour \(cb_j\) and the operation of \(r_{ji}\) are extracted from \(CB_CDV\). Then, it computes the cost of the service by calling all \(r_{ji}\) (Definition 9) defined in the cost behaviour (Line 21). Next, it puts the cost computed according to each cost behaviour model in a set of service cost values (ServiceCost), where each element is represented as a tuple \(<s_j,cost_j>\). Finally, with the cost of all services, the cost of the service composition is calculated according to Equation 5.3.
5 Techniques for Cost Analysis

\[ \text{totalCost} = \sum_{j=1}^{n} \text{cost}_j, \text{ where } \text{cost}_j \geq 0 \] (5.3)

5.5 Cost Analysis Using Event Logs in Execution Phase

This technique uses event logs \((LOG)\) of the service composition to compute the cost of candidate services for a specific task \((t)\) of a service composition \((sc)\). These event logs, which are obtained from the execution history of the composition, must have the same properties of the fragment of the event logs shown in Fig. 5.1. However, since the purpose of this event log is to compare the cost of a set of services, it needs all the cost attributes of all candidate services that will have their cost computed. For example, task \textit{Convert WAV to MP3 Audio Format} of the service composition described in Section 1.1 has two candidate services, \textit{ConvertFile} and \textit{AudioFileConverter}. As defined in Table 1.1, the cost of service \textit{ConverFile} is computed according to the size of a WAV file, while the cost of \textit{AudioFileConverter} is computed according to the number of files that must be converted into MP3. Therefore, to estimate the cost of the services and decide which one must be adopted to convert the files, both the size and number of MP3 files must be logged in the event logs.

The costs of the candidate services are computed by Algorithm 3. In addition to the event logs \((LOG)\), this algorithm receives the service composition identification \((sc)\), the task \((t)\), all candidate services \((S)\) of \(t\) and the period \((\text{periodOfAnalysis})\) at which the \(LOG\) was extracted. Before computing the estimated cost, the algorithm computes the cost driver values \((CB.CDV)\) according to the cost behaviours of the candidate services (Lines 2-16). For that, it processes all entries of the log \((reg_k \in LOG)\) for the service composition \((sc)\) and task \((t)\), and gets all business attributes associated with the cost behaviours of the candidate services \((s_j \in S)\). The value of the cost driver of each cost behaviour \((cb.cdv_{jw})\) is then added to the set of cost driver values \((CB.CDV)\) (Lines 7-14) according to the following rules:

- If the unit of \(cd_{kw}\) is \textit{Quantity}, the value is increased by 1,
- Otherwise, the value of attribute \(acd_{kw}\) is assigned to the cost driver.
Algorithm 3: Compute Cost of Candidate Services at Runtime by Using a Task, a Service Composition, a Set of Services, Event Logs, and Period of Analysis.

1: procedure COMPUTE_SERVICE_COSTS_BY_EVENT_LOG(t, sc, S, LOG, pA)
2:   for all regₖ in LOG of the process sc and task t do
3:     for all sₖ in S do
4:       opₖ := sₖ.getOperation()
5:       cbₖ := sₖ.getCostBehaviour()
6:       CDₖ := cbₖ.getCostDrivers()
7:       for all cdⱼw in CDₖ do
8:         acdⱼw := regₖ.getAttribute(cdⱼw.variable)
9:         if unit u of cdⱼw is Quantity then
10:            CB_CDV.add(cbⱼ, opⱼ, acdⱼw.name, 1)
11:            else
12:               CB_CDV.add(cbⱼ, opⱼ, acdⱼw.name, acdⱼw.value)
13:         end if
14:       end for
15:     end for
16:   end for
17:   CB := CB_CDV.getCostBehaviours()
18:   for all cbⱼ in CB do
19:     for all rules rⱼi in cbⱼ do
20:       CDV := CB_CDV.getCDV(cbⱼ, rⱼi, op)
21:       for all variable cost cⱼi in rⱼi do
22:         cdvⱼi := CDV.getValue(cb, rⱼi, op, cⱼi.getCostDriver())
23:         costValue := costValue + cⱼi(cdvⱼi, pA) ▷ Definition 3
24:       end for
25:       for all choose chⱼi in rⱼi do
26:         costValue := costValue + chⱼi(CDV, periodOfAnalysis) ▷ Definition 6
27:       end for
28:       for all fixed cost cⱼi1y in whⱼi1 in chⱼi do
29:         costValue := costValue - cⱼi1y(null, pA) ▷ Definition 3.
30:     end for
31:   end for
32:   for all upTo upⱼi in rⱼi do
33:     costValue := costValue + upⱼi(CDV, pA) ▷ Definition 8
34:   end for
35:   ServiceCost.add(cbⱼ, service, cost)
36: end for
37: return ServiceCost
38: end procedure
Once the set of cost driver values for all cost behaviours \((CB_{CDV})\) is computed, the algorithm calculates the costs of the candidate services. Therefore, it gets the set of cost behaviours \((CB)\) contained in \(CB_{CDV}\) (Line 17). After that, it analyses all rules \((r_{ji} \in cb_j)\) and gets the cost driver values from \(CB_{CDV}\) associated with the cost behaviour \((cb_j)\) and the operation defined in \(r_{ji}\). Recall that if \(r_{ji}\) is not associated with an operation, all cost driver values associated with \(cb_j\) are extracted from \(CB_{CDV}\). Otherwise, only the cost driver values associated with \(cb_j\) and the operation of \(r_{ji}\) are obtained from \(CB_{CDV}\).

Differently from the Algorithms 1 and 2, Algorithm 3 does not use Definition 9 to compute the rules of the cost behaviours. This is because in execution phase we do not consider fixed costs that do not depend on the service invocations. Therefore, only the variable costs \((c_{jik})\) declared directly in \(r_{ji}\) is computed according to Definition 3 (Lines 21-24). In the case of \textit{choose} \((ch_{jik})\), the cost is computed according to Definition 6 (Line 26) and than the fixed costs declared in the first \textit{when} \((wh_{jik1})\) defined in \(ch_{jik}\) are subtracted from the total cost (Lines 27-29). In the case of the \textit{upTo} \((up_{jik})\) defined in \(r_{ji}\), they are computed according to Definition 8 (Lines 31-33).

After that, Algorithm 3 puts the cost of each service in a set of service cost values \textit{ServiceCost} (Line 35), where each element is represented as a tuple \(<s_j, cost_j>\). With the cost of all candidate services located in the set of service cost values \textit{ServiceCost}, it is possible to identify the services with the highest and lowest costs.

### 5.6 Cost Analysis for One Service Composition Instance in Execution Phase

This technique has the purpose of computing costs of candidate services to be compared in execution phase for a particular instance of the service composition.

In this technique, Algorithm 4 computes the cost of the candidate services \((S)\) by using the business attributes \((A)\) of an instance of the service composition. The main difference between this algorithm and Algorithm 3 presented in Section 5.5 is the way the cost driver values are computed and the use of the estimated values of cost drivers defined in the cost behaviour.
models. Due to the size of this algorithm, we divided it into three parts, as follows:

- **Algorithm 4** is the main algorithm and responsible for computing the cost driver values for the cost behaviours \((CB_{CDV})\) and calling Algorithm 5 to compute the service costs;
- **Algorithm 5** computes the service costs based on the cost driver values computed previously by Algorithm 4; and
- **Algorithm 6** is called by Algorithm 5 to compute fixes costs by using estimated consumption of cost driver.

### Algorithm 4 Compute Cost of Candidate Services for Service Comparison at Runtime by Using the Service Composition, Set of Candidate Services and Business Attributes.

```plaintext
1: procedure COMPUTE_SERVICE_COSTS_FOR_COMPARISON_BY_ATTRIBUTE(sc,S,A)
2:     for all service \(s_i\) in \(S\) do
3:         \(cb_i := s_i.getCostBehaviour()\)
4:         \(op_i := s_i.getOperation()\)
5:         \(CD_i := cb_i.getCostDrivers()\)
6:             for all \(cd_{ij}\) in \(CD_i\) do
7:                 \(a_{ij} := A.getAttribute(cd_{ij}.getBusinessAttribute(sc))\)
8:                     if unit u of \(cd_{ij}\) is Quantity then
9:                         \(CB_{CDV}.add(cb_i,op_i,cd_{ij}.name,1)\)
10:                    else
11:                         \(CB_{CDV}.add(cb_i,op_i,cd_{ij}.name,a_{ij}.value)\)
12:                 end if
13:             end for
14:         end for
15:     \(CB := CB_{CDV}.getCostBehaviours()\)
16:     for all \(cb_j\) in \(CB\) do
17:         \(cost := computeServiceCost(cb_j,CB_{CDV})\) \(\triangleright\) Call Algorithm 5
18:         ServiceCost.add(cb_j,service,cost)
19:     end for
20:     return ServiceCost
21: end procedure
```

The input of Algorithm 4 is the service composition \((sc)\), the set of candidate services \((S)\) and the set of business attributes \((A)\) of the task that is going to be performed. In order to identify the appropriate cost function,
the algorithm gets the cost behaviour model \((cb_j)\) (Line 3) and operation \((op_i)\) (Line 4) of each service \((s_i \in S)\) that can perform the task. After that, it gets the set of the cost drivers \((CD_i)\) of the cost behaviour \((cb_i)\) (Line 5) and computes the values of these cost drivers according to the values of the business attributes \((a_i \in A)\) (Lines 6-13).

After computing the set of cost driver values \((CB_{CDV})\), Algorithm 4 calls Algorithm 5 to compute the cost of each candidate service \((s_i \in S)\) (Line 17). After that, it puts the cost of each service in a set service costs (Line 18) that can be used to identify the service with the lowest cost.

Algorithm 5, in its turn, computes the cost of a candidate service by using the cost behaviour \((cb_j)\) and the set of cost drivers \((CB_{CDV})\) computed previously by Algorithm 4. As mentioned in Section 5.2, fixed cost is despised when comparing service costs in the execution phase. Therefore, when there are cost functions \((c_{ij})\) defined directly in rules \((ri)\) Algorithm 5 computes only the ones with variable type (Lines 4-7). In the case of \((c_{ij})\) defined in choose functions \((ch_{ij})\), Algorithm 5 uses the estimated consumptions of the cost drivers \((EC_{CDV})\) to identify which when’s or otherwise must be computed. Once identified the appropriated when or otherwise function, the cost the choose is computed by calling Algorithm 6 (Lines 13 and 19). After that, the cost functions in the choose that do not depend on the use of the service is removed from the total cost of the service (Lines 22-24).

In upTo functions \((up_{ij})\) defined in \(ri\), Algorithm 5 also uses the estimated consumptions of the cost drivers \((EC_{CDV})\) instead of using the cost driver values \((CDV)\) computed by using the business attributes of the task. This occurs because the choice of the slices that must be used to compute the service cost depends on the value of cost driver consumed in a period. Since \(up_{ij}\) is computed by considering the estimated consumption in a period and we need the cost for one instance of the service composition, the cost computed by the function \(up_{ij}\) is divided by the estimated consumption of the cost driver defined in \(up_{ij}\) in order to calculate the estimated cost of a unit of the cost driver. With the estimated cost of a unit of the cost driver, it then multiplies this value by the respective cost driver value defined in \(CDV\) (Line 30).

In Algorithm 6, all cost functions \((c)\) with variable behaviour is computed according to Definition 3. However, once the cost driver values \((CDV)\) refers to only one instance of the service composition and not to a period of analysis, this algorithm gets the estimated consumption of the default cost
5.6 Cost Analysis for One Service Composition Instance in Execution Phase

Algorithm 5 Compute Cost of Service at Runtime by Using a Cost Behaviour and the Cost Driver Values.

```
procedure COMPUTE_SERVICE_COST(cb, CB_CDV)
  for all rules ri in cb do
    CDV := CB_CDV.getCDV(cb, ri, op)
    for all variable cost cij in ri do
      cdvij := CDV.getValue(cb, ri, op, cij, getCostDriver())
      costValue := costValue + cij(cdvij, NULL)
    end for
    for all choose chij in ri do
      EC_CDV := cb.getEstimatedConsumptions(ri, op, chij, period)
      for all when whijk in chij do
        if (condition(EC_CDV) of whijk is 1) then
          for all cost cijkl in whijk do
            costValue := costValue + computeCost(cb, cijkl, ri, op, CDV)
          end for
          if (there is an otherwise owij in chij) AND (no one whijk was chosen) then
            for all cost cijkl in owij do
              costValue := costValue + computeCost(cb, cijkl, ri, op, CDV)
            end for
            end if
            end for
        end if
      end for
    end for
    for all upTo upij in ri do
      EC_CDV := cb.getEstimatedConsumptions(ri, op, upij, period)
      cdvij := CDV.getValue(cb, ri, op, upij, getCostDriver())
      cdvECij := EC_CDV.getValue(cb, ri, op, upij, getCostDriver())
      costValue := costValue + upij(EC_CDV, upij, period)/cdvECij × cdvij
      end for
    end for
  end for
  return costValue
end procedure
```

Definition 8:  

```
```

Algorithm 6 Compute Cost In Execution Phase by Using Cost Behaviour, a Cost Function, a Service Operation and a Cost Driver Value.

1: \textbf{procedure} \textsc{computeCost}(cb, c, op, CDV) \\
2: \hspace{1em} \textbf{if} \ c \ \text{is variable} \ \textbf{then} \\
3: \hspace{2em} cdv := CDV.getValue(cb, op, c.getCostDriver()) \\
4: \hspace{2em} costValue := costValue + c(cdv, NULL) \quad \triangleright \text{Definition 8} \\
5: \hspace{1em} \textbf{else} \\
6: \hspace{2em} dcd := cb.getDefaultCostDriver() \\
7: \hspace{2em} ec := dcd.getEstimatedConsumption(op) \\
8: \hspace{2em} cdv := CDV.getValue(cb, op, dcd.name) \\
9: \hspace{2em} costValue := costValue + c(NULL, ec.period)/ec.value \times cdv \quad \triangleright \text{Definition 8} \\
10: \hspace{1em} \textbf{end if} \\
11: \hspace{1em} \textbf{return} \ costValue \\
12: \textbf{end procedure}

The driver defined in the cost behaviour (Lines 6-7) to compute the fixed cost according to the period of the estimated consumption of the default cost driver. With the estimated fixed cost, it computes the estimated cost for a unit of the default cost driver and then multiplies this value by the respective cost driver value defined in \textit{CDV} (Line 9).

5.7 Concluding Remarks

This chapter presented techniques to compute the cost of service compositions in planning and execution phases of the service composition life-cycle. All the techniques were developed by taking into account the complex cost behaviours and the cost behaviour metamodel defined in Chapter 4. Moreover, in our techniques to compute cost to compare services in execution phase we defined that fixed costs that do not depend on the use of the service must be despised. This is because we assume that in this phase all candidate services are already contracted by the service composition provider.

Initially, it discussed the motivation and requirements for the development of the techniques. Next, it presented an overview of the techniques. Than, it described the techniques to analyse the cost of service compositions in the planning phase. Finally, it described the techniques to compute the
cost of services in order to compare the cost of candidate services when the composition is already in execution.
Chapter 6
Architecture of Cost Management Systems

This chapter describes the architecture we proposed for developing service-based cost management systems. The proposed architecture is an end-to-end solution to handle the cost of service compositions throughout the service composition life-cycle introduced in Section 2.1. The architecture uses the cost behaviour metamodel and the set of algorithms presented in Chapter 4 and Chapter 5, respectively. This chapter initially gives the motivation and requirements for the development of the architecture. Next, it provides an overview of the architecture. Finally, it presents details of the architecture’s components.

6.1 Motivation and Requirements

Nowadays, software systems that support organisational business processes need to continuously adapt themselves in an automated manner to guarantee the Quality of Service (QoS) [93, 94]. This ability depends on the presence of monitoring mechanisms that enable us to identify, detect, and even predict critical situations that require changes, for instance, in the system’s configuration and behaviour [95].

As an architecture to develop service-based applications, it must provide the following tasks to allow monitoring and adaptation of service composition [96]:

- Define adaptation requirements;
- Define the artifacts that must be monitored;
Specify which properties of the monitoring artifacts should be monitored;
• Provide monitoring mechanisms;
• Log and collect service execution details;
• Obtain business metrics by analysing service execution data;
• Detect business situations that require management attention;
• Determine appropriate control actions to take in response to business conditions; and
• Perform adaptation.

Moreover, the architecture must meet the following requirements that were based on some key challenges identified in Chapter 3:

• Allows the management of service composition throughout the service composition life-cycle;
• In the Planning phase, the architecture must allow to analyse service costs taking into account all class of cost behaviour;
• In the Definition phase, it must allow configuring the service composition to ensure the cost requirements defined in the Planning phase;
• In the Scheduling and Constriction phase, the architecture must allow maximising the profit of service composition by reducing its cost when selecting candidate services;
• In the Execution phase, it must allow to execute, monitor, and adapt service composition to ensure the cost requirements specified in the Planning phase.

6.2 Overview

Fig. 6.2 shows the architecture proposed to develop service composition systems with cost management. This architecture has components to plan, define, schedule, construct, execute, analyse, monitor and adapt service composition. It was developed based on the components of IBM’s reference model for autonomic control loops [97], namely Monitor, Analyse, Plan, Execute, and Knowledge.

Although the focus of this thesis is the cost management, the proposed architecture has also been designed to meet other quality requirements, such as reliability and performance.
6.2 Overview

The main components of the architecture are Knowledge Repositories, Modeller, Execution Engine, Service Selector, Monitor, Analyser and Adapter. They were developed taking into account the aforementioned requirements of the architecture.

The Knowledge Repositories store data produced and used in the architecture, such as cost behaviours models. With this component, the architecture can manage the cost of a service composition throughout its life-cycle. UDDI [32], for instance, can be used to store services. For the other information, such as execution log and cost behaviours, relational Database Management System (DBMS) or even XML files can be utilised.

The Modeller allows business, service and cost experts to model service compositions and cost behaviours in the Planning phase of the service composition (see Fig. 2.4). Developers use Service Composition Editor to specify service compositions. This component also allows experts to annotate the service composition definition with additional information, such as the set of candidate services, the quality requirement and the attributes logged in the Knowledge Repositories. The cost behaviours are modelled as instances of the cost behaviour metamodel presented in Chapter 4 with the Cost Behaviour Editor. Once modelled, all service composition definitions and cost behaviour models are stored in the Knowledge Repositories that allow all components of the architecture to access the models.

After modelling the service composition and cost behaviours, one can analyse the cost of the service composition. The Analyser outlines the service composition cost in two alternative strategies: using the modelled com-
Fig. 6.2 Generic Architecture of Cost Management Systems of Service Composition.
position annotated with expected cost behaviours and reliability information as described in Section 5.3; or using the event logs of the execution of the service composition as outlined in Section 5.4. The first strategy can be employed when there is no event log of the service composition available to be analysed. The second strategy can be adopted when there is event log from the real execution or a simulation of the service composition.

Once the service composition has been modelled and analysed, it must be configured with all information that allows the Execution Engine to execute the service composition correctly. After that, the service composition is deployed and executed in Execution Engine, which can be any available service composition engine, such as Apache ODE [98] or Activiti [41]. In the deployment, the Execution Engine receives the service composition from the Modeller and asks the Service Selector to create the lists of candidate services and quality requirements of all tasks of the service composition.

With the lists of candidate services and quality requirements, the Service Selector is responsible for selecting the services to perform the tasks of the composition. After creating the list of candidate services, when a new service is stored in the list or when the Execution Engine asks the Service Selector for a service to perform a task, Listeners catch the list of candidate services and send it to the Monitor. In this way, cost and other quality attributes of the services in the list of candidate services can be computed by Analyser. When the Analyser computes all required quality attributes of all services, it informs the Adapter that the quality attributes of the candidate services have changed. Then, the Adapter acts to adapt the service composition, for instance, by asking the Service Selector to re-order the list of candidate services. Following sections present each component of the architecture in more detail.

### 6.3 Modeller

The Modeller is the user interface for service and cost experts to plan and define service compositions, and model service cost behaviours. Additionally, it allows cost and service experts to interact with the Analyser, the Execution Engine and the Knowledge Repositories to analyse costs, deploy service
compositions and store artefacts (e.g., service definitions and cost models), respectively.

Experts specify the service composition by using Service Composition Editor. This component also allows experts to annotate the service composition with service and cost information (e.g., service identification and cost drivers) required by the Analyser and the Execution Engine to analyse cost and execute the composition, respectively.

The Modeller interacts with the Analyser to examine the cost of the composition in two different ways: using an event log or the annotated service composition model. To analyse the cost using the event log, the Modeller must make the event log available to the Analyser. The event log can be obtained by simulation or from Knowledge Repositories when the composition has a history of executions, as shown in Table 5.1.

Once modelled and analysed, the composition is configured to be executed, monitored and adapted by the management system. This configuration occurs in Definition phase of the service composition life-cycle (see Fig. 2.4). In this phase, the composition is annotated with the candidate services that can perform each task, i.e., the information required to invoke these services by the Execution Engine, such as the WSDL and service operation. In addition to the information about the candidate services to be monitored, it is also necessary to annotate information that allows monitoring the cost of the composition, such as the business attributes that must be logged in the Knowledge Repositories, the quality requirement of each task and the level of monitoring.

The proposed architecture is designed to meet other quality requirements as well, such as performance and reliability. Therefore, we assume that each task of the composition can have a specific set of non-functional requirements that must be monitored when the tasks are performed. Therefore, in the Definition phase, it is also necessary to inform the quality requirements for the tasks, which can be valued (e.g., cost=$10.00 and reliability=0.96) or their relative priority (e.g., cost=0.6 and reliability=0.4). The priority of each attribute must be greater than zero and smaller than 1.0, and the sum of the priorities must be equal to 1.0. Additionally, the level of monitoring should be defined in annotations, to inform the Monitor how the task must be monitored, like in every execution of a task or in a certain time interval.
6.4 Execution Engine

The Execution Engine is responsible for managing the execution of service compositions. This component can be any available service composition engine, such as Activiti, that invokes services and coordinates these invocations according to the service composition specification produced in the Planning and Definition phases.

When the Execution Engine receives the service composition to be deployed, it creates a list of business attributes for each task of the composition that must be logged into Knowledge Repositories. Moreover, the Execution Engine sends to the Service Selector the list of candidate services with the non-functional requirements to be used by Service Selector for choosing the service whose quality best fits the quality needs of each task.

In the Execution phase, the Execution Engine requests the services from the Service Selector to perform each task of the composition without knowing how the services were chosen by the Service Selector. When the Execution Engine invokes the services, Listeners capture the invocation and send the data to the Monitor, which is responsible for logging the data into the Knowledge Repositories.

6.5 Service Selector

The Service Selector is responsible for managing the list of candidate services and provides the services that meet the required quality attributes of each task. Among other operations, this component has the following functionalities:

- CRUD operations on the list of candidate services for each task; and
- Rank the list of candidate services taking into account the quality attributes of the services.

When a list of candidate services is created, or new services are added to a list, the Service Selector ranks the services of the list based on quality attributes of the services and the quality requirement of the task. However, when a new service is placed on this list, the quality attributes of the service can not be known yet. Therefore, the Monitor gets a new service and asks
the Analyser to compute the quality attributes of the service and then the list of candidate services is ranked taking into account the quality requirements of the task and the quality attributes of all candidate services.

To determine the ranking, the Service Selector checks if the task has one or more quality requirements. If the task has only one quality requirement, candidate services are ranked taking into account this requirement and the inherent direction of the quality attribute, which indicates if bigger or smaller values of the quality attributes are better for the quality of the service. When the task has two or more quality requirements, the Service Selector performs three operations, namely normalisation, aggregation and ordering [99], as shown in Fig. 6.3. In the first operation, the quality attributes of the candidate services and requirements of the task are normalised by the Normaliser, to made them range between 0 and 1. Next, the quality attributes are aggregated by the Ranker to compute a single value that represents the level of the quality attributes offered by a service. Finally, the list is ranked by the Service Selector by using the aggregated value computed by the Ranker.

The normalisation takes into account the inherent direction of each quality attribute [99]. If bigger values are better than smaller ones, the Normaliser applies Equation 6.1 to the quality attributes of all candidate services and requirements of the task. Otherwise, the Normaliser uses Equation 6.2 instead.

\[
q'_i = \begin{cases} 
\frac{q_{max} - q_i}{q_{max} - q_{min}} & \text{if } q_{max} - q_{min} \neq 0 \\
1 & \text{if } q_{max} - q_{min} = 0
\end{cases}
\] (6.1)

\[
q'_i = \begin{cases} 
\frac{q_i - q_{min}}{q_{max} - q_{min}} & \text{if } q_{max} - q_{min} \neq 0 \\
1 & \text{if } q_{max} - q_{min} = 0
\end{cases}
\] (6.2)

Where,

- \( S \) is the set of all candidate services;
- \( s_i \) is a service in \( S \);
- \( q_i \) is a quality attribute of the \( i-th \) service \( s_i \);
- \( q_{min} \) is the minimum value of the quality attributes \( q \) of all services in \( S \);
- \( q_{max} \) is the maximum value of the quality attributes \( q \) of all services in \( S \);
- \( q'_i \) is the normalised value of \( q_i \).
The Ranker aggregates the quality attributes of the services [99] according to the strategy adopted by business experts when defining the quality requirements of the task. If the experts set the quality requirements with values, like, e.g., cost = $10.00, the Ranker calculates Euclidean Distance to aggregate the quality attributes. If experts use the priority (weight) of the quality attributes, like, e.g., cost=0.6 and reliability=0.4, the Ranker applies the Simple Additive Weighting equation to perform the aggregation.

Euclidean Distance evaluates the similarity between quality attributes of candidate services and user’s quality requirements as shown in Equation 6.3.

\[ D(s_i) = \sqrt{\sum_{i=1}^{n} (q_{ji} - r_j)^2} \]  

(6.3)

Where,
- \( S \) is the set of all candidate services;
- \( s_i \) is the \( i \)-th service in \( S \);
- \( n \) is the number of quality attributes associated with the service \( s_i \);
- \( q_{ji} \) is the normalised value of the \( j \)-th quality attribute of the \( i \)-th service;
- \( r_j \) is the value of the \( j \)-th quality requirement.

Simple Additive Weighting uses multiplication and addition operations to compute a weighted average over the collection of quality attributes of the candidate service as shown in Equation 6.4.

\[ SWA(s_i) = \sum_{i=1}^{n} q_{ji} \times w_i \]  

(6.4)

Where,
- \( S \) is the set of all candidate services;
- \( s_i \) is the \( i \)-th service in \( S \);
- \( n \) is the number of quality attributes associated with the service \( s_i \);
- \( q_{ji} \) is the normalised value of the \( j \)-th quality attribute of the \( i \)-th service;
- \( w_j \) is the weight of the \( j \)-th quality requirement.

Other strategies, such as Utility Function and Entropy Function [99], can also be adopted by the Ranker to aggregate quality attributes.

After computing the aggregate values of the quality attributes of the candidate services, the Service Selector ranks the candidate services and selects...
Fig. 6.3 Sequence Diagram of the Ranking.
6.6 Monitor

While the service composition is being executed by the Execution Engine, the Monitor observes the execution, the cost and others quality attributes of the composition, and its individual services. When monitoring, this component logs all information necessary to compute the cost as specified by the cost and service experts in the Planning phase (see Fig. 2.4). Additionally, this component is responsible for asking the Analyser to compute the cost of the candidate services.

As previously mentioned, the proposed architecture has been designed to manage cost and other quality attributes of service compositions. Hence, the Monitor consists of a Monitoring Manager and a set of Quality Monitors, like the Cost Monitor, which is plugged into the Monitor to watch a quality attribute of the composition. Besides, the Monitor has a module called Event Log Extractor which is responsible for extracting event logs from different data sources. When the Monitor receives an event to monitor, it calculates the cost of all candidate services of a task of the composition. These events that start the cost monitoring can come from the Service Selector or the Monitor. In the case of the Monitor, it can be configured to observe the cost of services at time intervals (e.g., 30 seconds, 2 minutes or one day) so that monitoring is done in parallel with the execution of the composition.

In the Execution phase (see Fig. 2.4), the cost of the candidate services can be calculated in two ways: by using the event log of the business instances already executed (Section 5.5); or by using the task attributes of the instance of the composition being executed (Section 5.6).

Fig. 6.4 and Fig. 6.5 show the sequence diagrams of monitoring the cost of candidate services according to the attributes to perform one instance of the service composition and the execution history of the service composition, respectively. When the service costs are computed based on the task attributes, the Monitor receives the set of attributes and the list of services
from the Service Selector and sends them to the Analyser. Otherwise, when
the service costs are computed based on a log, the Monitor requests the
Monitoring Manager to generate the event log to be sent to Analyser along
with the list of services and the list of quality attributes that are being mon-
itored. In this case, the Monitoring Manager asks the Quality Monitors to
inform which attributes need to be extracted from the repository to generate
the event log so that their respective Analysers can calculate the value of the
quality appropriately. After that, the Monitoring Manager requests the Event
Log Extractor to extract the event log from the Knowledge Repositories and
after that, the Monitoring Manager sends the log to the Analyser.

6.7 Analyser

The Analyser aims to assess the quality attributes of services. This com-
ponent includes the Analysis Manager, which handles the Quality Analysers
components plugged into Analyser to compute a particular quality attribute.
The Cost Analyser is an example of a Quality Analyser component.

The Cost Analyser is responsible for computing the cost of both indi-
vidual services and the entire composition by using the algorithms presented
in Chapter 5. In the Planning phase, business and cost experts can use the
Modeller to examine the expected cost of the service. In this phase, the ex-
erts can analyse the cost by using annotated compositions (Section 5.3) or
execution logs (Section 5.4).

In the Execution phase, the Analyser receives requests from the Monitor
to examine the cost taking into account the history of execution (event logs)
or a set of attributes of an instance of the composition. In the case of event
logs, the request to analyse the cost is asynchronous, and the activities follow
the sequence as shown in Fig. 6.7. Otherwise, the request is synchronous
shown in Fig. 6.6.

After it receives a request to calculate the cost, the Analyser asks the
Analysis Manager to compute the quality attributes of the services. When
the Analysis Manager receives this request, it asks the Quality Analysers,
such as the Cost Analyser, to calculate the quality attributes.

The Cost Analyser provides two techniques to compute the cost of ser-
vices in the Execution phase, taking into account the event log or an instance
Fig. 6.4 Sequence Diagram of Service Monitoring using Event Log.
Fig. 6.5 Sequence Diagram of Service Monitoring per Task Execution.
Fig. 6.6: Sequence Diagram of Quality Analysis per Task Execution.
of the composition. In which case, the Cost Analyser uses the technique described in Section 5.5, or in Section 5.6, respectively. After analysing the quality attributes, the Analyser informs the Adapter that the quality attributes of the candidate service have changed and the Adapter acts to adapt the service composition.

### 6.8 Adapter

The Adapter can interact with both the Execution Engine and the Service Selector. Concerning the Execution Engine, the Adapter can adapt the complete execution of the service composition to manage its quality. For instance, if the cost of the composition gets higher than a certain threshold, it can ask the Execution Engine to stop the entire composition to avoid that the cost increases even more. The Adapter can also ask the Service Selector to re-order the list of candidate services according to the cost of each service. Additionally, this component can also send information to the Modeller to inform the cost/service experts about some unexpected problem with the cost of the service composition.

The Adaptation Manager is responsible for creating an adaptation plan according to the rules [100] defined in the Knowledge Repositories. An adaptation plan is a sequence of actions with the purpose of keeping the service composition as planned in the Planning phase. Finally, the Adapter performs the adaptation of the service composition according to the adaptation plan, like re-order the list of candidate services of a task and notify the business experts.

### 6.9 Concluding Remarks

This chapter describes the generic architecture for building service composition systems that support cost management throughout the phases of the service composition life-cycle.

The proposed architecture allows service and cost experts to plan and define service compositions aware of the cost. Moreover, it provides components to construct the service composition with the best set of services that
meets the cost requirements defined by the experts. In the Execution phase, the architecture provides components to execute, monitor and adapt the service composition to ensure the cost requirements defined in the Planning phase.

For cost analysis, the architecture uses the techniques detailed in Chapter 5. In the Planning phase, the experts can analyse the cost of the service composition by using the modelled composition annotated with expected cost behaviours and reliability information or using the event logs of the execution of the service composition. In execution phase, the cost of candidate service can be analysed in parallel with the execution of the composition or before the execution of each task. The choice of these techniques depends, for instance, on the performance required for the service composition, once the cost analysis before each task execution tends to affect the performance of the composition more negatively than the cost analysis in parallel with the execution of the composition.

The chapter presented the motivation and requirements for the development of the architecture initially. Next, an overview of the proposed architecture was presented, followed by a detailed description of its core elements.
Part III
Treatment Validation
This chapter presents a prototype developed over our approach for cost management of service composition. It initially introduces the architecture of the prototype. Next, it describes two running examples of service compositions implemented by the prototype.

7.1 Implementation

The prototype of the proposed approach was implemented by extending the Activiti [41], an existing BPM platform able to develop and execute service compositions. Activiti is open source and provides an execution engine and tools to manage business processes to be executed in Cloud infrastructures.

Fig. 7.1 shows the architecture of Activiti [41], which is composed of a process engine (Activiti Engine), a Web-based modelling environment (Activiti Modeller), an Eclipse plugin (Activiti Designer), a Web application (Activiti Explorer) and a REST interface (Activiti REST).

Activiti Engine is the core component of Activiti, and can be embedded in a Java (Web) application or run standalone to execute BPMN processes and creates workflow tasks. Activiti Modeller and Activiti Designer are modelling environments for creating and configuring business process diagrams. Finally, Activiti Explorer and Activiti REST are Web applications that provide Web pages and REST interface, respectively, to interact with Activiti Engine, for example, for starting and stopping business process instances.
Fig. 7.1 Architecture of Activiti [41].

Fig. 7.2 shows the architecture proposed in Chapter 6 used to implement the prototype. In the implementation, we both reuse and implement new components, which are identified in Fig. 7.2.

To implement Modeller, we improved Activiti Modeller and Activiti Explorer to support cost analysis and service compositions monitoring. To support the cost analysis described in Section 5.3, we added new attributes to service tasks and flow elements in Activiti Modeller. In the case of service tasks, we added attributes Service:operation, Cost drivers and Reliability, as shown in Fig. 7.3. Service:operation represents the name of the service and its operation that performs the task. Cost drivers represent the list of service’s cost drivers and their respective values, for instance, number_of_files:2. For the flow element, we added attribute Probability that represents the probability of the respective flow to be executed, as shown Fig. 7.4.

The BPMN specification provides an extension mechanism that allows to add custom information of a domain-specific concepts to the BPMN XML serialisations. By using this mechanism, we could add new service and cost attributes while still ensuring the validity of the BPMN core elements.

Activiti Explorer was extended to provide an interface for business, service and cost experts to upload and visualise cost behaviour models (see Fig. 7.5), and analyse the cost of service compositions (see Fig. 7.6). All
7.1 Implementation

Fig. 7.2 Architecture of the Implemented Prototype
Besides cost behaviours, the Knowledge Repository also stores service descriptions, execution log, and service composition definitions. For the service descriptions, we used Apache jUDDI [102]. To represent the cost behaviour in the service description, we created a cost \textit{tModel}, as shown in Listing 7.1.

\textbf{Listing 7.1} Cost Behaviour \textit{tModel}.

```
<\textit{tModel}>
   <\textit{tModelKey}="uddi.ufpe.br:cost"
   xmlns="urn:uddi-org:api_v3"
   xmlns:ns2="http://www.w3.org/2000/09/xmldsig#">
    <\textit{name xml:lang}="en">Cost Behaviour</\textit{name}>
    <\textit{overviewDoc}>
     <\textit{overviewURL} useType="xsd">
     </\textit{overviewURL}>
    </\textit{overviewDoc}>
</\textit{tModel}>
```

Different from other works that extend the UDDI, such as [103, 104, 105, 106], we do not inform the cost per request or volume of data in the service description. Instead, we report the cost behaviour identification of the service, which allows the execution engine, for instance, to compute the
Fig. 7.5 Cost Behaviour Screenshot.
Fig. 7.6 Cost Prediction Screenshot.
service cost taking into account complex cost behaviour, execution history and business attributes of each instance of the service composition.

According to Activiti specification, service tasks are executed by an implementation of interface `JavaDelegate`. In order to use a generic `JavaDelegate` implementation that could be used to perform any service task, we developed a class (`WSDelegate`) that implements this interface. To perform a task, this class needs the list of candidate services and their operation used to perform the task (`serviceId::operation`), the values of the parameters of the operations, and the name of the variable that will store the service response, as shown in Fig. 7.7.

![Fig. 7.7 Class Fields.](image.png)

When the `WSDelegate` receives the list of the candidate services defined by the expert, it sends the list to the Service Selector soon after. After that, the `WSDelegate` asks the Service Selector for the best service to perform the task before each task execution.

To invoke the selected service, `WSDelegate` uses Apache CXF [107] and Java Reflexion [108]. Apache CXF is an open source services framework that allows to build, develop and invoke services by a variety of protocols and transport, such as SOAP, RESTful, and HTTP. Reflection allows to examine or modify information about the attributes, methods and constructors of objects running in the Java virtual machine. It is used in the `WSDelegate` to create the required parameters dynamically to invoke the service and the variable that store the service response.
For the service composition monitoring, the Activiti Modeller was improved to add Requirements and Monitoring attributes in the BPMN’s service tasks, as shown in Fig. 7.1. Requirement attributes include the list of monitored non-functional properties and either the value or weight of the respective requirements. Attributes to log is a list of logged business attributes. Business attributes are related to variables used in Activiti business processes, named in Activiti as process variables.

![Attributes for monitoring](image)

**Fig. 7.8** Service Task’s Attributes for Monitoring

Besides Activiti, we also reused XESame [109], which is an application that extracts event logs from non-event log data sources into eXtensible Event Stream (XES) [110] format. XES is a standard for event logs based on XML used by ProM, a process mining framework [92]. Since XESame is not an Application Program Interface (API), we have modified it to use as an API in our prototype.

Listeners were implemented in AspectJ [111]. AspectJ is an Aspect-oriented Programming (AOP) extension that enables us to add additional behaviours to Activiti without modifying its code. We implemented all other components from scratch in Java.

To use our prototype, the users need to have knowledge in BPM, cost management, and SOA. Furthermore, it is desirable for the user to have already worked with Activiti.

### 7.2 Example Applications

This section describes two applications used in the experiments to evaluate our framework. The first experiment consists of a service composition to
convert audio files from WAV to MP3. The second one consists of a pizza
delivery application implemented by service composition.

7.2.1 Audio Files Converter

This section describes a service-oriented application developed to convert
audio files from WAV to MP3 (see Fig. 7.9).

Fig. 7.9 Business Process to Convert WAV to MP3.

The business process of this application starts when the customer in-
vokes the composition by informing her e-mail address and the WAV file
to be converted. After that, a service validates the e-mail address of the
customer (Validate Email Address). If the e-mail address is valid, a service
(Convert WAV to MP3 Audio Format) converts the WAV file into the MP3
file format. Otherwise, if the e-mail address is invalid, the service composi-
tion sends an e-mail to notify the administrator that an error occurred (Notify
Administrator by Email) and the file was not converted correctly. When the
file is converted correctly, the customer receives an e-mail from a service
(Send Email to Customer with the MP3 File) having the converted file.
7.2.2 Pizza Delivery

This section describes a service-oriented pizza delivery application based on a service composition proposed by [112]. Fig. 7.10 shows the business process of the application.

The process starts when a client makes a phone call to the Pizza Company to order a pizza. After that, the application uses a service (Authenticate Client) to authenticate the customer. In case the client is authenticated, the application invokes a service to obtain the profile of the client (Get Client Profile). Then, the application sends the pizza catalogue to the client so that she can choose the desired pizza (Ask Client to Choose the Desired Pizza).

When the customer chooses a pizza, she informs the delivery address and agrees to the price, Credit Card service is responsible for debiting the pizza price from customer’s credit card (Credit Card Transaction). If the transaction is approved, the pizza baker is notified to prepare the selected pizza (Communicate Desired Pizza to Baker). After that, the application uses the client’s delivery address to get the client’s coordinates (Get Address GPS Coordinates). These coordinates are then sent to a Map service responsible for generating the map and the best route from the pizza company to the client’s address (Get Map and Route to GPS Waypoint). Next, the application asks a message service to send the map and route to the personal digital assistant (PDA) of the pizza delivery boy (Send Map to Delivery Boy PDA). Finally, the application sends a text message to the client to inform that the pizza will be delivered within $n$ minutes (Send SMS to Client).

7.3 Concluding Remarks

This chapter initially presented a prototype of the proposed architecture implemented over an existing BPMN engine, called Activiti. After that, it described two running examples of service compositions that were implemented by using the developed solution. With this prototype, we intend to perform experiments in order to evaluate our work.
Fig. 7.10 Business Process of the Pizza Delivery Application.
Chapter 8
Experimental Evaluation

This chapter presents the experiments conducted to evaluate both the accuracy and performance of our approach to analyse and manage the cost of service composition. It shows the results of three experiments carried out to evaluate the proposed solution.

8.1 Cost Prediction by Using BPMN

In this section, we conducted an experiment in order answer the following question:

- What is the accuracy of our approach in the cost analysis of service composition by using the service composition definition annotated with reliability and cost information, as described in Section 5.3?

In order to answer this question, this experiment uses the application Audio File Converter (Section 7.2.1). Table 8.1 shows the Web services used to realise the service composition and their cost behaviours and reliability levels.

To evaluate our approach, the cost of the service composition was calculated in two different ways: by adding up the cost computed directly by each atomic service, and by applying our approach to predict the cost by using the annotated BPMN. Then, we compare both costs to evaluate the results obtained with our approach.
### 8.1 Experimental Setup

All services were implemented in Java and deployed in a Tomcat server running on an instance of `t2.micro` virtual machine of Amazon EC2. Moreover, a MySQL database [101] stores service’s data and executes on the same virtual machine.

The environment presented in Chapter 7 was used to model and execute the service composition. Service `SimpleEmail` executes the task `Notify Administrator by Email`. Web service `ValidateEmail` performs Task `Validate Email Address` whose cost is variable and depends on the number of invocations. To simulate different scenarios, 95% of e-mails sent to this service was valid and 5% invalid. Service `WAVtoMP3` accomplished task `Convert WAV to MP3 Audio Format` whose cost is mixed and composed of a fixed cost of $20.00 per month plus a variable cost driver depending on the file size. This service converts 95% of files correctly, i.e., its reliability is 95%. Besides, the size of WAV files sent to this service followed a uniform distribution between 1 MB and 5 MB. MP3 files sent back to customers by the task `Send Email to Customer with the MP3 File` have a compression ratio following a normal distribution with mean equal to 0.206 and standard deviation equal to 0.048 on the corresponding WAV file. The practical observation of WAV to MP3 conversion rates produced these parameters. Finally, we assumed that 5% of the attempts to send e-mails failed.

### Table 8.1 Cost Behaviours

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Service</th>
<th>Cost Model (€)</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validate Email Address</td>
<td>ValidateEmail</td>
<td>0.05 per invocation.</td>
<td>0.999</td>
</tr>
<tr>
<td>Notify Administrator by Email</td>
<td>SimpleEmail</td>
<td>2.00 per month.</td>
<td>0.999</td>
</tr>
<tr>
<td>Convert WAV to MP3 Audio Format</td>
<td>WAVtoMP3</td>
<td>10.00 per month; and, 0.03 per 1 MB of file.</td>
<td>0.950</td>
</tr>
<tr>
<td>Send Email to Customer with the MP3 File</td>
<td>CompleteEmail</td>
<td>0.10 per 100 emails / month; and Attachments: First 10 GB/month: 0.12/MB, Greater than 10 GB/month: 0.09/MB</td>
<td>0.950</td>
</tr>
</tbody>
</table>
To simulate the number of daily service composition executions, we generated a random number between 20 and 50 with a continuous uniform distribution function. As a result, we obtained the value 33, which means that the service composition in our experiment has an average of 33 daily executions. Therefore, for each simulated day, the number of executions follows a normal distribution with mean 33 and standard deviation 1, which we assumed to compute the distribution.

### 8.1.2 Statistical Analysis

In our analysis, we used a sample with 20 costs relating to a month (30 days) of execution of the service composition. Each cost of the sample was computed by summing up the monthly costs of all individual service of the service composition.

![Probability Plot of Cost of the Sample](image)

**Fig. 8.1** Probability Plot of Cost of the Sample

To identify which statistical test to apply in our experiment, we tested if the costs in the sample fitted a normal distribution. According to the result of our test depicted in Fig. 8.1 with $p\text{-value}>0.2$, we find strong evidence
that the sample follows a normal distribution with 95% of confidence level, mean 205.4 and standard deviation equal to 2.503.

After that, we computed a 95% confidence interval for the costs in the sample by using the $t$ distribution. As result, the mean cost for the sample of 20 costs was 205.44, with a 95% confidence interval of (204.27, 206.61). Therefore, it is appropriate to state that based on the given data we are 95% confident that the cost mean of the population is between 204.27 and 206.61.

### 8.1.3 Predicting Cost

By using the approach presented in Section 5.3, we first modelled the service composition in BPMN (*Model Service Composition*) and the cost behaviour of all services using our cost metamodel (*Model Cost Behaviour*). Next, we annotated the business process with the service and operation identifications (*Annotate Service Identification with Operation*), cost driver values (*Annotate Cost Driver Values*), task reliability (*Annotate Reliability*) and probabilities to all alternative paths of the business process (*Annotate Conditional Statement Probabilities*). After annotating the business process, we executed Algorithm 1 to compute the cost of each service according to its respective cost behaviour (*Compute Cost of Service Composition*), shown in Table 8.2.

<table>
<thead>
<tr>
<th>Service</th>
<th>Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidateEmail</td>
<td>49.50</td>
</tr>
<tr>
<td>SimpleEmail</td>
<td>2.00</td>
</tr>
<tr>
<td>WavtoMP3</td>
<td>90.40</td>
</tr>
<tr>
<td>CompleteEmail</td>
<td>63.88</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>205.78</strong></td>
</tr>
</tbody>
</table>
8.1.4 Experimental Results

In our experiment, we simulated the cost of 30 days to test if the mean cost calculated by service providers invoked by the composition is significantly different from the one predicted by our approach. For this purpose, we generated 20 examples of the cost computed by services providers and compared them with the cost calculated by our approach.

According to our experiments, the cost computed by our approach is in between the confidence interval of the cost mean of the sample used in the analysis. Therefore, we asserted that the difference between the cost computed by our approach and the cost computed directly by the atomic service providers is not statistically significant.

8.2 Cost Predictions by Using BPMN and Event Log

In this section, we conducted an experiment in order answer the following question:

- What is the accuracy of our approaches in the cost analysis of service composition by using the service composition definition annotated with reliability and cost information, and event logs, as presented in Sections 5.3 and 5.4?

In order to answer this question, we compare the costs of service compositions predicted by using both approaches presented in Sections 5.3 and 5.4 with the costs obtained manually and using the average cost [56].

For that, we use the service-oriented pizza delivery system described in Section 7.2.2. Web services execute the tasks of this service composition. Table 8.3 shows the cost behaviours and reliabilities associated with the service composition. The cost behaviours were created or adapted from real services, such as CDYNE [113], Twilio [114], and Shopify [115]. In contrast, the reliability of these services was generated randomly between 0.95 and 0.999.
Table 8.3 Cost Behaviours and Reliabilities

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Service</th>
<th>Cost Model (€)</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authenticate Client</td>
<td>AuthService</td>
<td>59.00 / Month</td>
<td>0.988</td>
</tr>
<tr>
<td>Get Client Profile</td>
<td>DBService</td>
<td>9.90 / Month + 0.05 per 100 requests</td>
<td>0.954</td>
</tr>
<tr>
<td>Ask Client to Choose the Desired Pizza</td>
<td>SQueue</td>
<td>0.02 / Kb</td>
<td>0.991</td>
</tr>
<tr>
<td>Communicate Authentication Failure to Client</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communicate Credit Card Problem to Client</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communicate Desired Pizza to Baker</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credit Card Transaction</td>
<td>ShopCredit</td>
<td>29.00 / Month + 2.9% + 0.30 per transaction</td>
<td>0.956</td>
</tr>
<tr>
<td>Get Address GPS Coordinates</td>
<td>GeoCoding</td>
<td>- 0-1,000 Queries / Month = 0.60 /100 Queries</td>
<td></td>
</tr>
<tr>
<td>Get Map and Route to GPS Waypoint</td>
<td></td>
<td>- 1,001-3,500 Queries / Month = 0.40 /100 Queries</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 3,500+ Queries / Month = 0.30 /100 Queries</td>
<td>0.989</td>
</tr>
<tr>
<td>Send Map to Delivery Boy PDA</td>
<td>PhoneMessage</td>
<td>0.0075 / Transaction</td>
<td>0.956</td>
</tr>
<tr>
<td>Send SMS to Client</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.2.1 Experimental Setup

We implemented all services used in this experiment in Java, deployed them to a Tomcat server, and registered them in a jUDDI server [102]. Moreover, a MySQL database installed on the same machine persists the services’ data.

We used the prototype presented in Chapter 7 to model and execute the service composition. Service AuthService was used to execute the task Authenticate Client with a fixed cost per month. To simulate different scenarios, 95% of the authentication requests are valid, and 5% invalid. Service DBService performed task Get Client Profile with a fixed cost plus a vari-
8.2 Cost Predictions by Using BPMN and Event Log

A flexible cost that depends on the number of requests. SQueue executes tasks Ask Client to Choose the Desired Pizza, Communicate Authentication Failure to Client, Communicate Credit Card Problem to Client, and Communicate Desired Pizza to Baker whose cost is variable.

Service ShopCredit performed task Credit Card Transaction with a fixed cost plus a variable cost that depends on both the value and number of transactions executed. Service GeoCoding has a step cost and performs both tasks Get Address GPS Coordinates and Get Map and Route to GPS Waypoint. Finally, service PhoneMessage performed both Send Map to Delivery boy PDA and Send SMS to Client services and has a variable cost that depends on the number of executed transactions.

Additionally, we assumed that the price of the pizza chosen by the customer follows a normal distribution with mean equal to 20.00 and standard deviation equal to 2.00, and 95% of the credit card transactions are authorised, and 5% are not approved. Moreover, we assumed that the message size average of tasks Ask Client to Choose the Desired Pizza, Communicate Authentication Failure to Client, Communicate Credit Card Problem to Client, and Communicate Desired Pizza to Baker are 9KB, 5KB, 5KB and 6KB, respectively.

In the simulation, the composition executes daily $n$ times in average and during 30 days. It is worth observing that we assume three different daily averages to simulate different volume of executions per month. To obtain the daily averages, we generated three random numbers: $n_1$, $n_2$, and $n_3$. We agreed that $n_1$, $n_2$, and $n_3$ should be less than 20 neither greater than 80, both numbers fixed by us to impose a minimum and maximum daily invocations. Additionally, we decided that $|n_1 - n_2|$, $|n_1 - n_3|$ and $|n_2 - n_3|$ should not be less than 10 to prevent that these numbers were too close to each other. As result, we obtained values $n_1 = 22$, $n_2 = 47$ and $n_3 = 69$. After that, we compared the cost computed by the services providers with the cost computed with the following approaches: our approach using models, our approach using event logs, and an approach using average cost [56].
8.2.2 Predicting Cost

Following our approach to predict the costs of a service composition using models, we first modelled the service composition \textit{(Model Service Composition)} and the cost behaviour of all services \textit{(Model Cost Behaviours)}. After that, we annotated the business process with the task reliability \textit{(Annotate Reliability)}, probabilities to all alternative paths of the business process \textit{(Annotate Conditional Statement Probabilities)}, service and operation identification \textit{(Annotate Service Identification with Operation)}, and cost driver values \textit{(Annotate Cost Driver Values)}. Next, Algorithm 1 computes the cost of each service according to its respective cost behaviour \textit{(Compute Cost of Service Composition)}.

Table 8.4 Average Cost per Invocation for 46 Invocations per Day

<table>
<thead>
<tr>
<th>Service</th>
<th>Average of Invocations per Day</th>
<th>Average of Invocations per Month</th>
<th>Average Cost per Invocation (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthService</td>
<td>45.448</td>
<td>1,363.44</td>
<td>0.04327</td>
</tr>
<tr>
<td>DBService</td>
<td>41.190</td>
<td>1,235.69</td>
<td>0.00851</td>
</tr>
<tr>
<td>Squeue</td>
<td>81.742</td>
<td>2,452.270</td>
<td>0.14894</td>
</tr>
<tr>
<td>ShopCredit</td>
<td>39.023</td>
<td>1,170.684</td>
<td>0.90477</td>
</tr>
<tr>
<td>GeoCoding</td>
<td>72.268</td>
<td>2,168.043</td>
<td>0.00492</td>
</tr>
<tr>
<td>PhoneMessage</td>
<td>68.706</td>
<td>2,061.186</td>
<td>0.00750</td>
</tr>
</tbody>
</table>

To compute the cost of the service composition by using the approach of the average cost of service, we first compute the average cost of each service and then we computed the cost for \(n_1 = 22\), \(n_2 = 47\), and \(n_3 = 69\). To compute the average cost of the services, we used their cost behaviour and reliability (see Table 8.3) and the average of the values of \(n_1\), \(n_2\), and \(n_3\), that is 46 invocations per day. Table 8.4 shows the average costs per invocation we obtained as a result.

8.2.3 Experimental Results

In this experiment, we evaluated the accuracy of our approaches presented in Sections 5.3 and 5.4 in analysing the cost of service compositions. As
a result, Figures 8.2-8.4 show that the costs computed by our approaches are similar to ones calculated by the service providers. In contrast, the cost estimated by using average costs is the same only in the case that the average number of invocations per day is 47, as shown in Fig. 8.3. This fact occurs because the average costs of the services were computed considering 46 invocations per day, which is near to 47 invocations per day.

![Fig. 8.2 Total Cost of 22 Invocations.](image1)

![Fig. 8.3 Total Cost of 47 Invocations.](image2)

Considering the individual cost of each service as shown in Fig. 8.5, we can see that the costs computed by our approaches are similar to ones cal-
Experimental Evaluation

Fig. 8.4 Total Cost of 69 Invocations.

culated by each provider. However, the costs in the approach using average costs are similar only when the service cost has a variable behaviour, e.g., SQueue and PhoneMessage services. In the other cases, i.e., fixed, mixed, and step cost, the approach using average cost does not yield the expected results, such as in AuthService, DBService, and GeoCoding. In the case of ShopCredit, which has mixed cost, the cost computed by using average cost is similar to the cost calculated by the service providers because the fixed cost represents only an average of 3% of the total cost of the service against 97% of the variable cost.

Therefore, we concluded that in this experiment the costs computed with our approaches are similar to ones computed directly by the service providers. Additionally, we compared the cost calculated by the service providers with the cost computed using average costs. From this comparison, we concluded that the cost using average costs only provides reliable results when the cost has a variable behaviour.

8.3 Adapting Service Composition in Execution Phase

In this section, we conducted an experiment in order answer the following question:

• What are the performance impact (t) and financial gain (€) when the service composition is monitored and adapted by using our approach to
8.3 Adapting Service Composition in Execution Phase

Fig. 8.5 Cost of the Services Computed in Different Approaches.

monitor and adapt service compositions based on the cost of the candidate services?

In order to answer this question, we conducted a set of experiments by using the pizza delivery application described in Section 7.2.2.

Fig. 8.6 shows the candidate services able to perform each task of the service composition. Task Client Authentication can be performed by services AuthService and TokenService. Task Get Client Profile has only one candidate service, namely DBService. Tasks Ask Client to Choose the Desired Pizza, Communicate Authentication Failure to Client, Commu-
cate Credit Card Problem to Client and Communicate Desired Pizza to Baker can be performed by the candidate services AmSQS, SQueue and QueueMessage. Services CheckoutM and CreditBeans are the candidate services of task Credit Card Transaction. To perform task Get Address GPS Coordinates, the service composition provider hired services ABCGeo and GeoCoding. Meanwhile, to perform task Get Map and Route to GPS Waypoint, we used third-party services GDiretion and ODiretion. Finally, tasks Send Map to Delivery Boy PDA and Send SMS to Client can be performed by services Tiwiw and AmSNS.

Table 8.5 shows the cost behaviours of the candidate services adopted in this scenario. They were adapted from actual services, such as CDYNE [113], Twilio [114], Amazon SNS [116], Google Maps Directions [117], Amazon Payment [118] and Shopify [115].

8.3.1 Experimental Setup

We defined five scenarios in which the services composition is executed 1000, 2000, 3000, 4000 and 5000 times per month. The number of executions is estimated based on the cost behaviours of some candidate services that change according to the number of executions per month, such as ABCGeo and GeoCoding. For each scenario, we randomly generate the price of pizza, message sizes, file size, authentication failures, and credit card validation. The pizza prices follow the normal distribution with means (μi) and standard deviations (σdi), where 1 ≤ i ≤ 4 and in the simulation i changes after each group of 25 executions to simulate more notable shifts in the value of a cost driver. The message sizes also followed normal distributions, and the percentage of valid authentication requests and approved credit card transactions were defined randomly from [0.9, 0.99].

All scenarios were simulated by using the following monitoring and adaptation strategy:

1. Service selection in the Planning phase, i.e., without adaptation in the Execution phase. In this case, the best set of services to perform the composition was selected according to the services’ cost behaviours and the mean values of their cost drivers;
8.3 Adapting Service Composition in Execution Phase

Fig. 8.6  Business Process of Pizza Delivery Application Annotated with Candidate Services.
Table 8.5 Cost Models of Candidate Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Cost Model (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthService</td>
<td>9.99/month</td>
</tr>
<tr>
<td>TokenService</td>
<td>0.04 / 100 Tokens</td>
</tr>
<tr>
<td>DBService</td>
<td>8.00 / Month + 0.06 / 100 Queries</td>
</tr>
<tr>
<td>CheckoutM</td>
<td>3.0% + 0.27 / transaction</td>
</tr>
<tr>
<td>CreditBeans</td>
<td>2.8% + 0.32 / transaction</td>
</tr>
<tr>
<td>AmSQS</td>
<td>0.05 / request + 0.005 / KB</td>
</tr>
<tr>
<td>SQQueue</td>
<td>0.015 / KB</td>
</tr>
<tr>
<td>QueueMessage</td>
<td>0.09 / request</td>
</tr>
<tr>
<td>ABCGeo</td>
<td>0 - 1,000 Queries / Month = 0.65 / 100 Queries</td>
</tr>
<tr>
<td></td>
<td>1,001 - 3,500 Queries / Month = 0.35 / 100 Queries</td>
</tr>
<tr>
<td></td>
<td>+3,500 Queries / Month = 0.25 / 100 Queries</td>
</tr>
<tr>
<td>GeoCoding</td>
<td>0 - 1,000 Queries / Month = 0.60 / 100 Queries</td>
</tr>
<tr>
<td></td>
<td>1,001 - 3,500 Queries / Month = 0.40 / 100 Queries</td>
</tr>
<tr>
<td></td>
<td>+3,500 Queries / Month = 0.30 / 100 Queries</td>
</tr>
<tr>
<td>GDirection</td>
<td>0 - 900 requests / Month = 0.006 / request</td>
</tr>
<tr>
<td></td>
<td>+900 requests / Month = 0.0044 / request</td>
</tr>
<tr>
<td>ODirection</td>
<td>0 - 1,000 requests / Month = 0.0055 / request</td>
</tr>
<tr>
<td></td>
<td>+1,000 requests / Month = 0.0050 request</td>
</tr>
<tr>
<td>AmSNS</td>
<td>First 20 Mb/month = 0.008 / KB</td>
</tr>
<tr>
<td></td>
<td>Next to 20 MB/month = 0.007 / KB</td>
</tr>
<tr>
<td></td>
<td>Next 20 MB/month = 0.006 / KB</td>
</tr>
<tr>
<td></td>
<td>Greater than 60 MB/month = 0.005 / KB</td>
</tr>
<tr>
<td>Tiwiw</td>
<td>0.0075 / Text Message</td>
</tr>
<tr>
<td></td>
<td>0.075 / Picture Message</td>
</tr>
</tbody>
</table>

2. Service selection before each task execution according to the attribute values of the business process; and
3. Service selection in parallel with the service composition execution by using event logs.

8.3.2 Experimental Results

Fig. 8.7 shows the financial gain of our two techniques (see Sections 5.5-5.6) compared to the service selection in the Planning phase, i.e., the best service to perform each task is selected in Planning phase by an expert and the composition is executed without monitoring and adaptation. We observed
that our two techniques provide adequate financial gain in all scenarios compared with the service selection in the Planning phase. Moreover, we noted that selecting services taking into account the business attributes of each instance of the composition can be more effective than the service selection in the background, which takes into account the execution history.

![Financial Gain Compared to Service Selection in Planning Phase.](image)

Fig. 8.7 Financial Gain Compared to Service Selection in Planning Phase.

Fig. 8.8 shows the average overhead (s) of executing the service composition by applying our techniques compared to service selection in the Planning phase by business experts. In contrast to the financial gain, it shows that the service selection in the Execution phase affects the performance of the composition negatively. In all scenarios, the overhead caused by the service selection in the Execution phase is higher than one in the Planning phase. Moreover, we observed that the overhead posed by the service selection in parallel with the execution of the service composition is lower than one performed before each task execution.

To analyse the financial gain of our techniques, taking into account the cost behaviours of the candidate services, we grouped the services according to their functionality, as follows:

- **Group1**: CheckoutM and CreditBeans;
- **Group2**: AmSQS, SQueue and QueueMessage;
- **Group3**: ABCGeo and GeoCoding;
- **Group4**: GDirection and ODirection; and
- **Group5**: Tiwiw and AmSNS.
With these groups, we summed up the costs of the services by the group and analysed the financial gain when applying our techniques compared to the selection of a service for each task in the Planning phase. We observed that the sum of the costs of Group3 and Group4 was the same in all scenarios, independently of the adopted techniques. Observing the cost behaviours of the services in these groups, we can see that in all cases the cost varies according to the number of invocations.

**Fig. 8.8** Service Composition Execution Overhead (\(t\)) Compared to Service Selection in Planning Phase.

**Fig. 8.9** Financial Gain of *CheckoutM* and *CreditBeans* Candidate Services Compared to Service Selection in Planning Phase.
In the case of Group 1, we observed financial gains when the service selection is performed by using our two techniques of service selection, as shown in Fig. 8.9. In contrast, in Fig. 8.10, we see that the financial gain of Group 5 occurs only when the service is selected before each task execution, which also takes place in Group 2. In both cases, we observed that the cost behaviours of the candidate services in each group have different cost drivers from each other.

Therefore, about financial gain, we conclude that service selection before each task execution is the best among the evaluated techniques, mainly when the cost of the service is based on any business attribute value that varies in each invocation, and the candidate services have cost behaviours with different cost drivers from each other. However, it affects more negatively the performance of the entire service composition than the other techniques. The method that takes into account the execution history of the service composition is more effective when the cost drivers of the monitored candidate services have significant changes over time. Moreover, if the costs of candidate services vary according to the same cost driver, the service selection during the Execution phase does not improve the financial gain of the service composition compared with the service selection in the Planning phase. Therefore, a proper choice of the method of service selection can promote the financial gain of the service composition with a minor impact on its performance.
8.4 Concluding Remarks

According to the experiments, we conclude that the costs computed by using our approaches are similar to the cost calculated by the service providers, which are the actual costs of the service composition. Additionally, we conclude that the cost analysis of candidate services in the Execution phase is beneficial only when the candidate services have a different class of cost behaviours or the cost drivers of the cost behaviours are different. Moreover, cost analysis in the Execution phase affects the performance of the service composition negatively. Therefore, we also conclude that cost analysis without taking into account different cost drivers, such as in [73], should not be applied to analyse service in the Execution phase of service composition, once it is neither effective nor efficient in this phase.

This chapter showed three experiments along with their setups, and results. The first experiment evaluated the accuracy of our approach to analyse costs of service compositions using annotated BPMN specifications. Next, the second experiment showed the accuracy of our strategies when compared with the use of logs. Finally, the third experiment evaluated the financial gain and the execution overhead of our approaches.
Chapter 9
Conclusions

This chapter first highlights the main contributions by reviewing the research questions presented in Chapter 1 and explains how we addressed them. Next, it describes the limitations of this work and gives recommendations for future research in the field of cost management of service composition.

9.1 Answers to the Research Questions

In Section 1.5, we presented the main research questions we addressed in this thesis. Here, we discuss how we answered these questions in the thesis.

**RQ1:** What are the current challenges found in the context of cost management of service compositions? To answer this question, we answered two sub-questions, as follows:

- **RQ1.1:** What is the state-of-the-art in cost management of service compositions?

  In Chapter 3, we discussed the state-of-the-art in cost management of service-oriented computing. We identified several techniques that could be used to manage the cost of services in particular phases of the service composition life-cycle. These techniques helped us understand the key issues of cost management. For example, the challenges in the Planning phase are about cost prediction to maximise service profit, while the
concerns in the Execution phase are about recording and controlling the cost so that adjustments can be made to adhere to the expected cost.

- **RQ1.2**: What are the open challenges identified in the literature about cost management of service compositions?

In Chapter 3, we discussed the open problems related to cost management in computational service, e.g., analyse business cost by applying more complete cost models, define a language able to express these kinds of cost models and finding services to perform service composition at runtime. Moreover, we observed that none of the available approaches treat cost management in all phases of the service composition life-cycle, which became a fundamental aspect of our work.

**RQ2**: How to manage the cost of service compositions throughout its life-cycle? To answer this question, we answered the following sub-questions:

- **RQ2.1**: What are the phases of the service composition life-cycle?

Cost management is a process that spreads out through all phases of the service life-cycle. Therefore, in Chapter 2 (Section 2.1), we discussed the phases of the service development life-cycle, i.e., Planning, Definition, Scheduling and Construction, and Execution. This life-cycle has been used throughout this thesis, especially to classify the challenges and the techniques found out in the literature, and to propose and develop the components of the framework.

- **RQ2.2**: How to express cost behaviours in cost management systems?

In Chapter 2 (Section 2.3), we identified the cost behaviours from the literature that we should be considered in this thesis. In Chapter 4, we proposed a cost behaviour metamodel to express cost behaviours of services taking into account the characteristics of SOA.

- **RQ2.3**: How to predict service composition costs with complex cost behaviours?

In Chapter 3, we discussed some relevant approaches to predict cost of service compositions. However, we realised that these approaches do not cover all classes of cost behaviours. Instead, most of them consider
only fixed cost per service invocation and ignore more complex cost behaviours and the impact of business attributes on variable costs, such as message size and credit card transaction value.

In Chapter 5, we discussed our algorithms to analyse the cost of service compositions in many phases their life-cycle to obtain information to manage service costs. These algorithms were developed by taking into account the cost behaviours presented in Section 2.3 and the service composition characteristics shown in Section 2.1.

- **RQ2.4:** How to manage the execution of the service composition to reduce the cost?

In Chapter 3, we discussed approaches that concentrate on the selection of services to minimise the workflow execution cost while meeting other non-functional requirements. However, these approaches consider only fixed cost per invocation or, when they find complex cost behaviours, they ignore the different types of cost drivers.

In Chapter 6, we presented an architecture for developing cost-aware service composition management systems that includes strategies to monitor and adapt the execution of compositions. In our architecture, monitoring and adaptation of service compositions occur not only before each task execution but also in parallel to the execution of the composition. Moreover, the architecture was developed by taking into account all complex cost behaviours that can be expressed with our cost metamodel.

**RQ3:** What are the impacts and accuracy of our treatment to manage cost of service composition? To answer this question, we answered the following sub-questions:

- **RQ3.1:** What is the financial gain of cost management in the service composition?
- **RQ3.2:** What is the performance impact of cost management in the execution of the service composition?

In Chapter 7, we developed a prototype of the architecture proposed in Chapter 6. In Chapter 8, we evaluated the financial gain and execution overhead empirically by experimenting with this prototype. Based on
these experiments, we observed that the performance of the compositions is always affected negatively when the execution engine manages the costs in the *Execution phase*, as expected, but this effect can be minimised if the monitoring is performed in parallel to the execution of the service composition. Moreover, the cost monitoring functionality in the *Execution phase* is beneficial only when the candidate services have a different class of cost behaviour or a different type of cost drivers. Therefore, we concluded that the approaches that do not take into account that similar services can have different cost behaviours and different cost drivers are not beneficial in the cost management of service composition. However, service selection before each task execution provides effective financial gain when the cost of the service is based on any business attribute value that varies in each invocation, or the candidate services have cost behaviours with different cost drivers from each other. Moreover, The method that performs in parallel with the execution of the service composition by taking into account execution history is more effective when the cost drivers of the monitored candidate services have significant changes over time.

- **RQ3.3:** What is the accuracy of our approach in the cost analysis of service composition?

In Chapter 8, we also reported on the experiments we performed to evaluate the accuracy of our approaches to analyse cost of service composition presented in Chapter 5. As a result of these experiments, we concluded that the cost computed by our approach is statistically similar to the cost computed directly by the atomic service providers.

### 9.2 Limitations and Future Work

Due to time and resource restrictions, this thesis has some limitations, which are opportunities for future research.

Cost drivers are variables that affect the cost of services directly. In the *Execution phase*, if the consumption of cost drivers is not estimated correctly, the service cost can be computed incorrectly and, consequently, the engine can be not able to select the cheapest service to perform the compo-
sition. Currently, our solution does not provide mechanisms to compute cost driver consumptions in the *Execution phase*. Therefore, we have the intention of working on techniques and algorithms to predict the consumption of cost drivers.

Additionally, although the proposed architecture was developed to support other quality attributes than cost we developed only the components to manage cost. Therefore, we plan to include other quality attributes in our architecture, such as reliability and performance.

We also plan to work on an adaptation of the technique to select services in the *Execution phase* since there are situations in which a technique for service selection is more beneficial than others, and the service selection in the *Execution phase* can affect the performance of the service composition negatively. In this case, the service selection would be chosen and adapted at execution time by the execution engine, taking into account the services’ cost behaviours, the behaviours of the business attributes, and the performance requirement of each task.

Concerning our metamodel, we intend to develop a graphical environment for modelling cost behaviours. In the current version of the developed prototype, we have used an XML editor for modelling cost behaviours.

We developed only a prototype of our framework, which was based on Activiti. Hence, our purpose is to develop other prototypes based on other engines to identify more limitations of our work.

Additionally, we plan to evaluate the effect of using microservices [119, 120] in the implementation of the architecture. Therefore, some of the prototypes as mentioned earlier can be implemented taking into account the microservice architectural style.

Finally, we aim to find out companies that can use our approach to perform new experiments that can allow us to identify new opportunities for research.
References


[32] OASIS. Universal Description, Discovery, and Integration (UDDI), 2015. [Online; accessed on 07-December-2016].


References


[89] Amazon. Amazon Simple Storage Service (S3) - Cloud Storage. 2016. [Online; accessed on 19-December-2016].


Appendix A
List of Publications

Appendix B
XML Schema Definition of the Cost Behaviour Metamodel

Listing B.1 CostBehaviourMetamodel.xsd

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<x:schema xmlns:cost="http://www.ufpe.br/CostBehavior"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.ufpe.br/CostBehavior" >
              schemaLocation="http://www.w3.org/2001/XMLSchema"/>
  <xs:element name="costBehaviour" type="cost:CostBehaviour"/>
  <xs:complexType name="CostBehaviour">
    <xs:sequence>
      <xs:element maxOccurs="1" minOccurs="0"
                   name="description" type="xs:string"/>
      <xs:element name="businessAttributes"
                   type="cost:BusinessAttributes"
                   minOccurs="0" maxOccurs="1"/>
      <xs:element name="costDrivers"
                   type="cost:CostDrivers"
                   minOccurs="0" maxOccurs="1"/>
      <xs:element maxOccurs="unbounded"
                   minOccurs="0" maxOccurs="1"
                   name="rules"
                   type="cost:Rules"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

159
<xs:attribute name="id" type="xs:ID" use="required"/>
<xs:attribute name="effectiveFrom" type="xs:date"
    use="optional"/>
<xs:attribute name="effectiveTo" type="xs:date"
    use="optional"/>
<xs:attribute name="currency" type="xs:string"
    use="required"/>
<xs:attribute name="resource" type="xs:anyURI"
    use="required"/>
</xs:complexType>
<xs:complexType name="Rules">
    <xs:sequence minOccurs="1">
        <xs:element maxOccurs="unbounded"
            minOccurs="0" name="cost"
            type="cost:Cost"/>
        <xs:element maxOccurs="unbounded"
            minOccurs="0" name="choose"
            type="cost:Choose"/>
        <xs:element name="upTo" type="cost:UpTo"
            maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="operation" type="xs:NCName"
        use="optional"/>
</xs:complexType>
<xs:complexType name="Cost">
    <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="0"
            name="description" type="xs:string"/>
        <xs:element maxOccurs="1" minOccurs="0"
            name="costDriver" type="xs:string"/>
        <xs:element maxOccurs="1" minOccurs="1"
            name="amount" type="xs:float"/>
        <xs:element maxOccurs="1" minOccurs="1"
            name="valuePerAmount" type="xs:float"/>
    </xs:sequence>
    <xs:attribute name="type" type="xs:string"
        use="required"/>
    <xs:attribute name="period" type="cost:Period"
        use="optional"/>
<xs:complexType name="UpTo">
  <xs:sequence>
    <xs:element maxOccurs="unbounded">
      <xs:element name="slice" type="cost:Slice" />
    </xs:element>
  </xs:sequence>
  <xs:attribute name="costDriver" type="xs:string" use="required" />
  <xs:attribute name="period" type="cost:Period" />
</xs:complexType>
105     </xs:complexType>
106     <xs:complexType name="Slice">
107         <xs:sequence>
108             <xs:element maxOccurs="unbounded"
109                 minOccurs="1" name="cost"
110                 type="cost:Cost">.
111         </xs:element>
112     </xs:sequence>
113     <xs:attribute name="amount" type="xs:float"
114         use="optional"/>
115     </xs:complexType>
116     <xs:complexType name="When">
117         <xs:sequence>
118             <xs:element maxOccurs="unbounded"
119                 minOccurs="0" name="cost"
120                 type="cost:Cost"/>
121             <xs:element name="bonus" type="cost:Bonus"
122                 minOccurs="0" maxOccurs="unbounded"/>
123     </xs:sequence>
124     <xs:attribute name="condition" type="xs:string"
125         use="required"/>
126     </xs:complexType>
127     <xs:complexType name="Choose">
128         <xs:sequence>
129             <xs:element maxOccurs="unbounded"
130                 minOccurs="1" name="when"
131                 type="cost:When"/>
132             <xs:element maxOccurs="1" minOccurs="0"
133                 name="otherwise"
134                 type="cost:Otherwise"/>
135     </xs:sequence>
136     <xs:attribute name="period" type="cost:Period"/>
137     </xs:complexType>
138     <xs:complexType name="Otherwise">
139         <xs:sequence>
140             <xs:element maxOccurs="unbounded"
141                 minOccurs="0" name="cost"
142                 type="cost:Cost"/>
143             <xs:element name="bonus" type="cost:Bonus"
144                 minOccurs="0" maxOccurs="unbounded"/>
145     </xs:sequence>
146     </xs:complexType>
<xs:complexType name="CostDrivers">
  <xs:sequence>
    <xs:element name="costDriver" type="cost:CostDriver" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="CostDriver">
  <xs:sequence>
    <xs:element name="estimatedConsumption" type="cost:EstimatedConsumption" maxOccurs="unbounded" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="unit" type="cost:Unit" use="required"/>
  <xs:attribute name="businessAttribute" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="Bonus">
  <xs:sequence>
    <xs:element name="value" type="xs:float"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="BusinessAttributes">
  <xs:sequence>
    <xs:element name="businessAttribute" type="cost:BusinessAttribute" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="EstimatedConsumption">
  <xs:attribute name="value" type="xs:double" use="required"/>
  <xs:attribute name="period" type="cost:Period" use="required"/>
  <xs:attribute name="operation" type="xs:string"/>
</xs:complexType>

<xs:complexType name="BusinessAttribute">
<xs:sequence maxOccurs="unbounded" minOccurs="0">
    <xs:element name="mapping" type="cost:Mapping"/>
</xs:sequence>
<xs:attribute name="name" type="xs:ID"/>
<xs:attribute name="type" type="cost:Type"/>
</xs:complexType>
<xs:complexType name="Mapping">
    <xs:attribute name="processId" type="xs:NCName"/>
    <xs:attribute name="attributeName" type="xs:NCName"/>
</xs:complexType>
</xs:schema>