A Proactive Adaptation Framework for Composite Web Services

Rafael Roque Aschoff

Department of Computer Science

City University London

A thesis submitted for the degree of

Doctor of Philosophy

September, 2014

Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to thank my supervisor, Professor Andrea Zisman, for her dedication, patience, assistance and motivation during my research at City University. Our conversations and her attention to detail was surely an opportunity for my personal and professional growth.

I would also like to thank my colleagues and friends for their support in times of self-doubt and frustration.

I would like to express my gratitude toward Professor Judith Kelner and the Network and Telecommunication Research Group for their help during the final stages of my work.

Finally, I would like to thank my family, my parents and sister, for their unquestionable support and encouragement to overcome all my challenges.

Declaration on consultation and copying

Abstract

Service orientation is a design paradigm consisting of a set of principles governed by a service-oriented architecture (SOA) to support the creation of software systems as a composition of interoperable services. The ability to effectively compose services is not a trivial task due to the dynamic nature of the execution environment of service compositions. In this context, dynamic service selection and composition is a critical requirement and one of the major research challenges for service-based systems.

This research investigates the identification, detection and prediction of the need for adaptation as well as ways to autonomously reconfigure the service composition during its execution time in order to improve service reliability and conformance with systems requirements and policies. We propose a framework for proactive adaptation of service compositions that extends current approaches for dynamic service composition by proactively and individually identifying the need for adaptation for each parallel running instance of service composition while avoiding unnecessary changes and distributing load request among different service operations when necessary.

Our framework has been tested and validated using different prototypes implemented in both simulated and real environments. The results were favourable with the research objectives and indicates a major gain in the use of the proposed proactive techniques in the execution and adaptation of web service compositions.

Chapter 1

Introduction

Business processes can be defined as a set of structured activities coordinated to accomplish a particular organisational goal. Due to the complexity of current computer-based applications, the rapid changes in market conditions and regulations, the dynamic creation of business alliances and partnerships, and the need to support the changing demands of users, business processes need to be more flexible, adaptable, and versatile in order to remain competitive in an increasingly competitive world. In this context emerges the *Service-Oriented Computing* (SOC), a distributed computing platform that envisages software as a temporary service rather than permanent property.

SOC extends previous computing paradigms with a new set of design principles, standard technologies, governance considerations, and defines *services* as the fundamental unit or basic building blocks to support the development of business processes. *Services* are loosely-coupled and autonomous computer-based entities owned by third parties and representing different functionality, which can be combined to realise applications and implement business processes. More specifically, SOC envisages how services can be used to implement the structured activities

defined in business processes.

Services can be implemented and accessed locally but it is the ability to locate and access third party services distributed over the web that spurred the rapid growth of the service orientation. *Web Services* are formally defined by the World Wide Web Consortium (W3C) as "a software system designed to support interoperable machine-to-machine interaction over a network" [19]. Web Services are defined through formal contract documents that exposes the available service operations, in a similar way to a traditional application programming interface (API).

The process of creating a coordinated flow of activities using distributed services in order to accomplish a certain task or implement a business process receives the denomination of *service composition*, and the resulting system is defined as a *Service-Based System* (also known as *Service-Based Application*) [2, 46]. More precisely, a Service-Based Application is the resulting service composition created as a coordinated aggregation of services and associated with the automation of a parent business process[48]. The terms service composition, service-based systems, and service-based application, are used as synonyms in this work.

Service-based systems have a great advantage when using services to implement activities required by business processes. However, by requiring such services through partner providers, the system loses control of these components. This can be an advantage, since the maintenance and implementation of the service are not the responsibility of the service-based system. On the other hand, these external services may suffer degradation in its performance over time, become faulty or unavailable, directly affecting the service-based system.

In fact, Web Services are offered and located in a very dynamic environment

where changes are almost certain and therefore should be taken into consideration. In other words, service based systems must continuously evolve in order to adapt to the dynamic execution context of business processes.

The constantly changing environment of business processes creates a complex problem for service-based systems. As outlined in [43][45][107], a major research challenge in SOC is the support for autonomic composition of services in which service compositions need to adapt autonomously and automatically to new situations.

Generally speaking, changes can be introduced to service composition through a mechanism that supports either *static adaptation* or *dynamic adaptation*. While the first mechanism focuses on remodelling the service composition during *design time*, the second mechanism focuses on adaptation during *runtime*. Dynamic adaptation is usually preferable to static adaptation due to the ability to change service-based systems without stopping their execution.

Moreover, methods to support adaptation of service compositions are usually classified depending on how to deal with possible internal system faults, namely: *reactive* and *proactive* approaches. *Reactive* approaches adapt the system by reacting to faults that occur and are observed during the system execution. *Proactive* approaches attempt to avoid faults by predicting their future occurrence[64].

Regardless of the strategy employed, the most desired feature of self-adaptable service-based systems is arguably the ability to improve, or at least maintain, the system reliability while meeting certain Service Level Agreement (SLA). In software engineering, reliability can be defined as the ability of a system to deliver its main functions under routine or unexpected situations[96].

In other words, an adaptable service-based system is designed to be executable

under certain constraints or system requirements and be tolerant to changes in the execution context. Certain events, such as the unavailability of certain service operations can cause the service composition to halt if a candidate service is not identified and use as replacement. Other events, such as an overloaded network can impact the performance of the composition which may lead to nonconformity with the SLA previously agreed between the involved parties.

The focus of the work described in this thesis report is on *dynamic and proactive adaptation of Web service compositions*. More specifically, the work in this thesis report investigates the identification, detection and prediction of the need for adaptation as well as ways to autonomously reconfigure the service composition during its execution time.

In the work described in this thesis report, proactive adaptation of service composition is defined as the combination of the preventive and perfection adaptation types. The preventive adaptation is concerned with the detection of the need for changes and enacting the required changes in a composition, before reaching an execution point in the composition where a fault or failure would occur, impacting the correct execution of the composition as a whole. The perfection adaptation aims to optimise the service composition even when it is running correctly.

Instead of just predicting and preventing unwanted situations in the service composition, however, we expect to continually improve and adapt a service composition even when there are no expected faults in the composition. Optimisation can be performed, for instance, by choosing faster service operations while reducing the costs related to the execution of the composition as a whole.

In this thesis report, we describe a framework for proactive adaptation of ser-

vice compositions named ProAdapt. The framework extends current approaches for dynamic service composition by proactively and individually identifying the need for adaptation for each parallel running instance of service composition while avoiding unnecessary changes and distributing load request among different service operations when necessary. Reactive adaptation is also supported and exploited in case where proactive adaptation is not possible.

ProAdapt performs monitoring and analysis over different events of interest in order to prevent possible execution failures and to improve service composition continuously, even when there are no expected or observable issues. Examples of these events are: (a) unavailability or malfunctioning of a deployed service operation, (b) QoS deviations, (c) evolution of requirements, and (d) emergence of better services. The reconfiguration of running instances is performed by replacing a single service operation or a group of service operations in the composition, by another service operation or by a group of dynamically composed service operations.

Different prototypes were developed to illustrate and evaluate the proposed framework. The analysis of the collected data shows that the architecture and methods defined for ProAdapt can efficiently improve the execution of service compositions in terms of performance and reliability. Overall, we believe that the work conducted in this thesis was satisfactory with respect to the objectives and hypotheses raised (See Section 1.4).

In what follows, we present some formal definitions required to a better understanding of the topic of this thesis. More specifically, Section 1.1 presents a customised view of the *Web Service Architecture*. Section 1.2 goes beyond the basic architectural model and presents the current work done in the implementation

of business processes as compositions of orchestrated Web Services.

The insights are then used to summarise the motivation and research challenges of this work in Section 1.3. The hypotheses and the main objectives of this thesis are presented in Section 1.4. The contributions are included in Section 1.5. Finally, Section 1.7 presents the outline of the thesis report.

1.1 Web Services Architecture

The concept of Web Services, and technologies to support them, have much evolved since the term was first publicly introduced by Bill Gates at the Microsoft Professional Developers Conference in Orlando on the 12th of July 2000. Although many definitions for the term Web Services exist, they are mostly related to the task of delivering and integrating business processes as services over the web [19].

Businessmen tend to prefer a looser definition, describing Web Services as any set of functionality delivered over the Internet, while software engineers are more inclined for a precise definition on the use of standards to support interoperability between software systems [2, 33].

The distinction becomes clear when surveying practical applications. Commercial banners, search engines, pools, and chat area are examples of external services usually consumed by websites in the form of plugins. While these are arguably the most apparent applicability of service components, they are usually based on different standards, such as URL commands and JavaScript.

On the other side, usually hidden from customers, services that provide functionality such as network traffic statistics, online payment processing, stock quotes,

and many other behind-the-scene functions, are usually accessed through formal standards including XML and SOAP. The work in this thesis report is oriented towards the software engineering point of view for Web services.

The necessity of Web Services technologies appears as a result of a growing demand of properly defined interaction between organisations (online businesses) and the lack of support from the software component paradigm in this context.

While components are helpful when it comes to rapidly software development and maintenance, the interoperability across organisations introduces a real challenge. Moreover, Web Services do not share the tangibility characteristic of software components, in the sense that Web Service implementation is not available for the system requesting to use it. Such characteristic is very interesting for business organisations because it permits a complete control of the provided services.

It is clear then that Web service technologies focus on improving the software-to-software communication, rather than techniques to implement actual functionality. The Web Service Architecture (WSA) dictates the foundation of such a communication model. The work described in this thesis adopts a variation of the basic WSA defined in [19].

The main component of the WSA, as well as the basic interaction between them are presented in Figure 1.1. The service requester is the party that wishes to make use of the functions offered by the service provider. The interaction between them is flexible enough to allow either the pull or push strategy. In the pull strategy the requester will contact the provider on demand, whenever its functions are required. In the push strategy however, the requester subscribes itself to the provider and waits for income messages. Finally, the service discovery

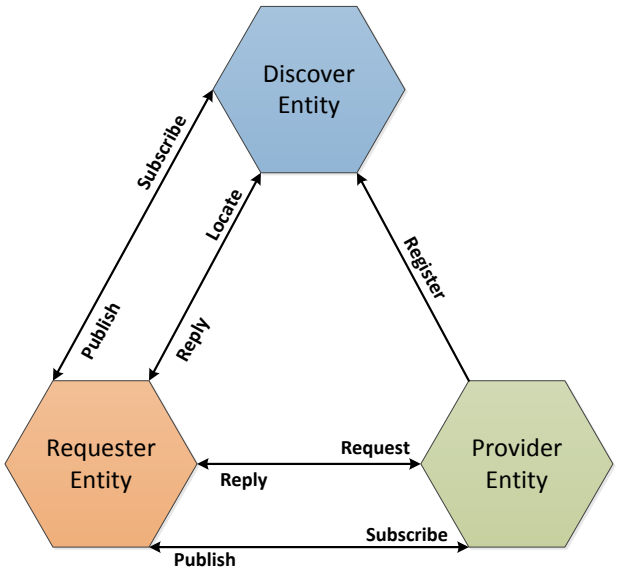

Figure 1.1: Basic Web Service Architecture

engine provides a way to register and locate services.

Before the requester and provider parties start exchanging messages, both of them have to implement a specific agent. The provider entity uses a concrete software agent to implement the abstract notion of a service. The requester entity requires an agent to work as a bridge between a software system (or human) attempting to interact with the desired service, and the provider itself.

The service interface is exposed in a document named Web service description (WSD), which contains detailed information about how to construct request messages for each operation available in the service, as well as how to interpret response messages. There may be also a need for exchange the expected semantic of the service which includes the expected order to call different functions or any other agreement between parties.

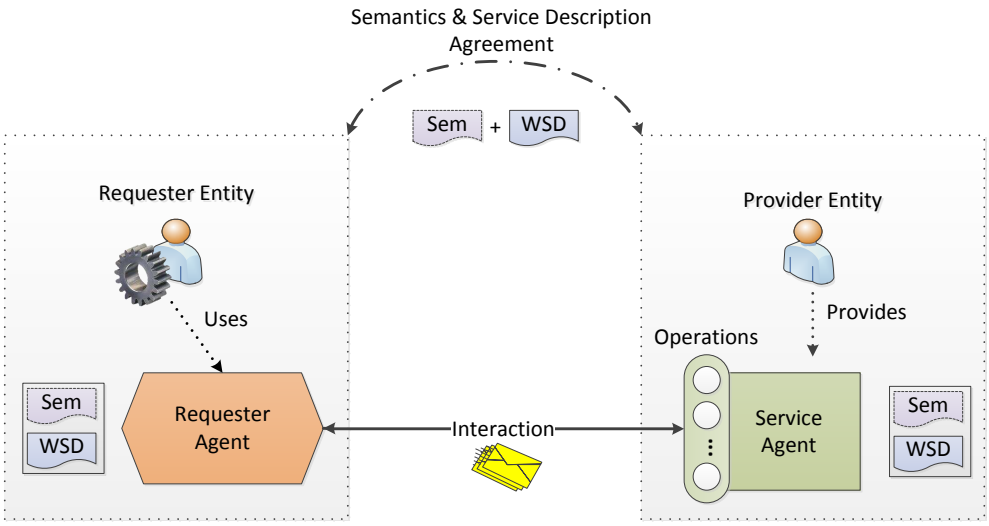

Figure 1.2: Web Service Interaction Model

As a model for these elements, Figure 1.2, adapted and extended from [19], presents a basic illustration of this software-to-software interaction model. It includes the basic elements that compound the Web service infrastructure, which revolves around the requester and provider entities. We define below the main components of Figure 1.2.

Service Agent : A Web service agent is a piece of software which implements the abstract set of operations defined in a service. It is also responsible for identifying the requested operation and for receiving and sending messages over the communication channel. A provider may decide to update its service agent to improve security or performance for instance, or just rollback for a previous working version in case of a problem, while maintaining the same functionality. In such case, the service itself remains the same, and all changes may go unnoticed by requester entities.

Requester Agent : The Web service interaction model defines standards for software-to-software communication, which means that a person cannot interact directly with a service. A requester agent provides an interface between a person (or software systems) and the desired service. It is also responsible for creating messages and wrap them into the right network protocol, as described in the WSD.

Requester Entity : A requester entity is the party interested in making use of Web service. It can be a software system which uses the service to provide some of its internal functions or a person or organisation that which to access the service directly. In either case, a requester entity will use an agent to interact with a Web service.

Provider Entity : A provider entity is the party interested in implementing a service through an agent. It can be an individual or organisation and is independently responsible for maintaining and make available its services agents. It is usually also the entity responsible for the advertising of services at some global Marketplace using standards such as UDDI [12] and DAML-S ServiceProle [99].

Web Service Description : A WSD is a document usually written in the Web Services Description Language (WSDL)[32] which provides a machine-readable specification of how the service can be invoked, the messages and types it expects, and what data structures it returns. The document has an abstract and a concrete section. In the abstract section a WSD defines the operations, types, and messages. The concrete section contains the binding information; the protocol and network address to make use of the service.

The XML-based Web Service Description Language is arguably the most well-know solution for WSDs. Some extension such as DAML-S [23], however, attempts to extend the service description with elements to answer what is the actual functionality available, instead of only how the operations are accessed. As another example, SA-WSDL [49] provides a mechanism to add annotations to WSDL components that can reference semantic models using arbitrary languages, which can be placed both within or outside the WSDL document.

Behaviour : The behaviour of a Web service is the shared expectation about the exchange messages between request and service agents. It generally defines the required ordering of operations to be invoked or some generic constraints over the set of available operations. The Web service behaviour may be implicit or formally registered to be processed by interested parties.

Communication Channel : The communication channel is the physical and logical transmission medium in which messages and documents are exchanged between requester and provider. The physical transmission medium is usually hidden from both parties. The logical transmission medium is the set of protocols chose to wrap the exchange messages or documents, such as Transmission Control Protocol (TCP), HTTP, and SOAP. It is important that both requester and provider support the same set of protocols, which is defined in the WSD.

The presented architecture, together with the interaction model provide an intelligent way for Service-Based Applications to invoke Web services, however, it is not enough for complex interaction involving composite services. The next sec-

tion describes how such complex interactions are accommodated by using service compositions.

1.2 Services Composition

When only atomic Web services are considered, that is, when service implementations (Service Agents) are enclosed and do not rely on external components, the basic infrastructure involving *Requesters*, *Providers*, and *Discovers*, is adequate. As Web services become more complex, however, their internal implementation will usually involve invocation of other Web services.

These distinct Web services receive the denomination of composed service [2], and the process of developing a composite service is called service composition[46]. For reasons of simplicity, from this point onwards, this report refers to composite services and service compositions as interchangeable synonyms.

While an atomic service is self-contained, requiring no connections to other resources to provide its operations, a composite service relies on interactions with external resources as a way to deliver its functionality. In fact, service compositions are not limited to connections with atomic services, and are recursively defined as an aggregation of atomic and composite services[117]. This new design pattern enables high sophisticated interactions suitable to implement complex logic while keeping a simple external interface.

More specifically, services compositions can be used to quickly implement business processes using partners' services as a way to implement functions required, while the composition itself hides the internal logic and is exposed as a default Web service using standards such as WSDL [32]. From the *Requester*

Entity point of view, this is an interesting approach, since it is usually concerned with receiving the required functionality, regardless of how it is implemented.

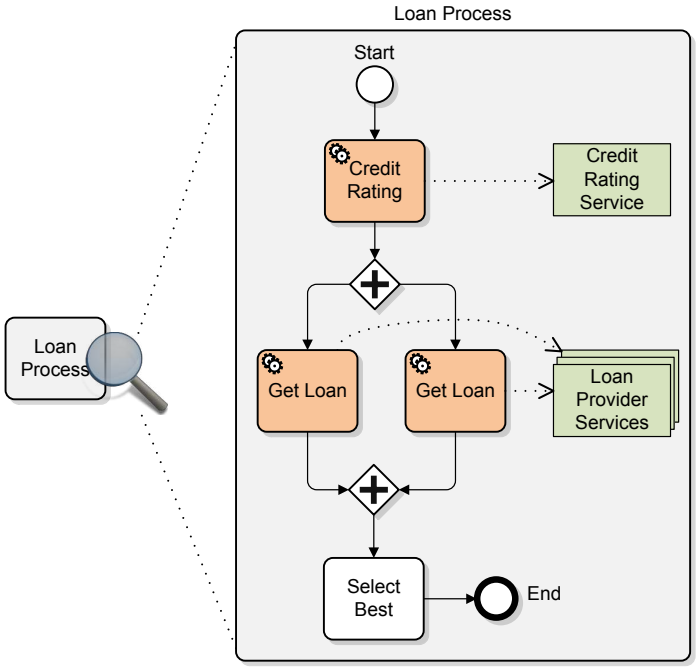

Figure 1.3: Service Composition: Loan Process

In engineering, the general flow of plant processes and equipment are usually illustrated using a process flow diagram (PFD) [124], which display the interaction between major equipment without getting into minor details. Designers of business processes and service compositions follow a similar approach using a Business Process Diagram (BPD)[60], which is a graphical representation composed of a flow of basic and structured activities. Basic activities are atomic basic units for the construction of the process, such as invoking Web Service operations. Structured activities are used to provide the logic flow of the process, and are recursively defined as an aggregation of basic and structured activities. The Business Process Model and Notation (BPMN) [30] is arguably the most

well-know standard for business process modelling.

In order to illustrate the use of BPDs, Figure 1.3 presents a Loan Process service composition using the BPMN graphic notation. The process starts when a client submits a loan application, and finishes with a loan offer. The internal process comprises three phases. First, the composition invokes an external service to receive the credit rating of the client. Next, the composition sends the client application to two different loan service providers. Finally, the composition decides for the best loan offer and returns it to the client.

BPMN provides a very powerful notation to describe the general properties and behaviour of service compositions, however, considering that it provides only the process skeleton, BPMN is not enough to actually make a process executable [54]. BPMN knowingly misses details not required to describe the process flow, such as data and links definition. Because of that, in order to make service composition executable, designers have to choose between (a) languages specifically designed for it, or (b) proprietary BPMN extensions aimed at fulfilling the required gaps.

Two languages appear as the candidates most frequently used in the description of executable processes, namely (a) the Web Services Business Process Execution Language (WS-BPEL) [73], and the XML Process Definition Language (XPDL) [116]. WS-BPEL, or BPEL for short, is an executable language based on the Pi-Calculus mathematical model and defined by the Organization for the Advancement of Structured Information Standards (OASIS) [113]. It is designed to specify business processes with Web services. XPDL is a general purpose business process definition language based on Petrinet models and standardized by the Workflow Management Coalition (WfMC) [138].

When choosing the best language to describe executable business processes, there are various arguments in favour of BPEL or XPDL. For instance, BPEL is arguable better fitted to describe distributed and collaborative processes, which is expected since its goal is the web service orchestration. On the other hand, XPDL is better defining standalone processes, since it is target at defining the process diagram. For designers looking for interchange business process definitions with detailed graphical information, XPDL may be the better choice, because it was design to include all aspects of a BPDs such as X and Y position of the nodes, while BPEL does not define in itself graphical aspects, and is targeted most exclusively to automatic interpreters. Regardless of the advantages and disadvantages of each standard, the fact is that BPEL has been winning the battle for the preferred language in the definition of executable business process for composite Web services. The above claim is supported by recent research and publications, e.g. [28][55][86], recognising BPEL as the *de facto* standard for Web services orchestration.

The prototypes developed for the ProAdapt framework use service compositions defined in BPEL. Executable languages such as BPEL provide all the detailed information required to enact business processes, but without a way of putting these processes into practice, they would not make much sense. A business process execution engine provides the execution environment and the infrastructure for business process execution [58]. Execution engines are responsible for enacting the various activities defined in the business process.

In order to execute a service composition defined in a language such as BPEL, it is necessary to employ a Work Flow Management (WFM) System [65], which is the component, or process, responsible for the automation of activities defined

in the service composition. Over the past few years, a number of execution engines have been proposed for different environment and languages, such as Silver [59], jBPM [125], Apache ODE [114], and IBM Websphere [77]. Silver is a very peculiar SOAP and BPEL execution engine for mobile devices that offer support for Java 2 Platform Micro Edition (J2ME). jBPM is an open-source project that allows the execution of business processes specified using BPMN with additional annotations. IBM Websphere is a SOA platform with run-time support for both Enterprise Service Bus and BPEL processes. Apache ODE is an open source project targeted specifically at BPEL processes, with good community support, easy installation, and service deployment procedures.

Given a proper way to represent the service composition and an engine to execute it, there is still the problem of how to select the operations to form the composition. In other words, we must know how to find services that satisfy the functional requirements of our business process and how to proper choose between different service candidates. The answers to these two questions are strongly related to the adaptation of the composition itself. More details regarding this subject can be found in Section 2.3.

1.3 Motivation and Research Challenges

The previous sections presented the overall context in which this thesis is inserted, including some of the challenges and motivations behind the topic of dynamic adaptation of service compositions. In this section, we summarise these motivations and research challenges and present some new insights that inspired the worked proposed in this thesis.

The Service-Oriented Computing paradigm has attracted great interest from industry and research communities around the world. Service integrators, developers, and providers are collaborating to address the various challenges in the area. Various approaches and tools have been proposed to support different areas of SOC such as (a) languages to describe services, (b) service design and development, (c) service discovery, (d) service composition, and (e) service management and monitoring.

Dynamically adaptable systems have also been the focus of study in several areas of computing such as software engineering, robotics, control systems, programming languages, and bio-informatics. The software engineering community attaches great importance to this topic as demonstrated by regular workshops held in conjunction with main conferences in the area (e.g. ICSE, FSE/ESEC, SEAMS, ICSOC).

To deal with the challenges of SOC dynamic and demanding business environment, service compositions need to be (a) self-configuring, compositions that are able to automatically identify, select, and combine new services with which to interact; (b) self-optimising, compositions that can select the best services with which to interact in order to become more efficient; and (c) self-healing.

Despite the advances in the area, more work is required to support the development of software systems based on dynamic composition of software services, enabling the provision of new products to the market in a rapid and efficient way. To be successful, these dynamic compositions should be able to adapt to various situations, including (i) changes in or emergence of requirements, (ii) changes in the context of the composition and participating services, (iii) changes in functional and quality aspects of services in compositions, (iv) failures in services in

compositions, and (v) evolution of services.

Based on the need to support dynamic evolution and changes of business activities, users' demands, and service availability, it is important to provide approaches that consider the various autonomic aspects together (e.g., self-configuring, self-optimising, self-healing, self-adapting) [118] in a proactive way, predicting problems before they occur.

As defined above, proactive adaptation of service composition constitutes the detection of the need for changes and implementation of changes in a composition, before reaching an execution point in the composition where a problem may exist. Such a problem may cause the process to generate erroneous, unexpected behaviour, or even to halt.

Examples of such proactive actions include (1) the prediction that a service used in a composition may be malfunctioning, unavailable, or no longer compliant with various aspects of the system, followed by the implementation of necessary changes in the composition even before this service is invoked during the execution of the composition; (2) the identification of the parts of a service composition that cannot fulfil a new requirement, again before execution of such parts; and (3) the identification of compositions parts that need to be changed due to the existence of a new better service, before the execution of these parts.

Ensuring QoS in Web services is a critical and significant challenge due to the dynamic and seemly unpredictable nature of their context of execution. There is often more than one business process available to execute a required task, which means some level of competition, both for network resources and attention of the market. Ensuring the correct execution of service compositions, while satisfying the QoS requirements, is considered an important differentiating point

for businesses.

The overall adaptation aim for service compositions is to permit the correct execution of business process and maintain or improve some system requirements such as reliability, performance, and general aspects of quality of service (QoS). In fact, the QoS parameters of Web services and compositions are one of the main concerns for service providers. The reason is that business opportunities may be lost, or penalties may be due if service compositions do not satisfy the expected performance requirements of their customers. The strong relation between user satisfaction, revenues, and the performance of a system was observed in a number of previous research done in the field.

In research conducted by *Amazon*, it was observed that when the time to generate the pages was increased by around 100ms, there was a drop on sales of about 1%[92]. In the session *What Google Knows* presented by Marissa Mayer at the Web 2.0 conference in 2006, she explained how a delay of 500ms when generating search results could lead to a traffic and revenue drop close to 20% [101].

The impact of degraded QoS values on business revenue, however, is not limited to the largest online retailer or the giant of internet-related products and services. In the competitive stock market, the TABB Group estimates that a response time of about 5 milliseconds behind the competition could cause a broker to lose $4 million in revenues per millisecond[75].

In summary, although services compositions appear as a solution for seamless integration of business processes over the Internet, this integration does not come without a price. Service compositions use resources from numerous partner services and need to be prepared for any possible changes induced by these part-

ner services. Such preparation comes in the form of self-adaptable and dynamic service compositions.

Overall, the existing approaches that deal with such issue are fragmented and in their initial stage of development, leaving space for further investigating the topic of adaptation of service compositions.

1.4 Research Objectives and Hypotheses

This thesis explores techniques to achieve autonomous execution of Web service compositions with a focus on the adaptation process. The general hypothesis of the work is the following:

Proactive adaptation of service compositions can improve the performance, reliability, and general conformance with system requirements of service compositions when compared to traditional static and reactive adaptation approaches.

The above general hypothesis can be broken down into the following sub-hypotheses:

- *It is possible to proactively identify the need for adaptation in service compositions by constantly monitoring the execution environment, systems requirements, and the status of the service composition itself.*
- *It is possible to use a local service repository and proactively replace candidate operations in service compositions.*
- *It is possible to adapt service compositions in parallel to their execution without stopping the business process.*

- *It is possible to avoid unnecessary changes in service compositions when identified problems can be compensated by parts of the service composition yet to be executed.*

Given the above hypothesis, the general aim of the research is *to support a dynamic and proactive adaptation of service compositions through the use of monitors, candidate service replacements, and techniques for prediction of problems in order to improve the reliability, performance, and conformance of business process.*

The above general aim can be broken down into the following measurable objectives:

Objective 1: Literature Review

To provide a literature review and analysis of works in relevant areas of the research topic. The review needs to include topics such as service composition, monitoring, discovery, adaptation, and failure prediction techniques.

Objective 2: Prediction of Problems

To design and implement mechanisms to support the detection and proactive prediction of events that may require adaptation of service compositions.

Objective 3: Events Analysis

To deliver techniques to help understanding potential faults and to identify the parts in a service composition that may be affected by detected or predicted faults.

Objective 4: Adaptation Approaches

To analyse and specify relationships between the different ways of adapting service compositions and the circumstances that may trigger adaptation.

Objective 5: Adaptation Framework

To specify an adaptation framework including techniques for monitoring, detection and prediction of events that may require adaptation, and enforcement of changes in a dynamic and proactive way. udies to demonstrate and evaluate the work.

Objective 6: Evaluation

To develop scenarios and evaluate proof-of-concept tools that will be created to support the techniques and mechanisms created for the Adaptation Framework considering medium to large scale case studies.

1.5 Contributions

The ProAdapt framework includes the following major contributions:

Response Time Modelling: The framework offers a new technique to support service operation response time modelling based on exponentially weighted moving average (EWMA) function [67].

Signature mismatches: The framework considers dependencies that may exist between the signature of service operations within a composition in order to avoid adaptation of the composition that may lead to mismatches of the signatures of the operations. In order words, the approach can avoid

a replacement due to signature mismatches or modify other parts of the composition to treat the mismatches.

Independent Execution Instance Adaptation: Different from existing service composition adaptation approaches, ProAdapt is able to reconfigure single execution instances independently, acting in response of specific needs for each execution instance, without interfering with other execution instances currently running. ProAdapt supports adaptations that are required for specific contexts of each execution instance.

Parallel Adaptation: ProAdapt supports the adaptation and sharing of events that may require adaptation in parallel of running instances, allowing for faster and potentially more efficient adaptation procedures.

Load Balancing: The ability to handle execution instances individually, together with the sharing of information in parallel enables ProAdapt to distribute the load between multiple service operations simultaneously.

Behavioural Compensation: In a extension of the ProAdapt framework we propose a solution to deal with missing behaviour of candidate replacement services by compensating such missing behaviour with other candidates, which can be new services or the ones already deployed in the composition.

Following the list of major contributions, this work also includes some other important additions to the field of service composition as presented below.

Spatial Correlation: ProAdapt employs a spatial correlation method to proactively identify potentially unreachable operations, services and providers

by inspecting messages of network protocols such as TCP/IP, HTTP, and SOAP.

Adaptation based on different classes of events : The framework supports adaptation of service composition triggered by four different classes of situations, namely C1: problems that cause the composition to stop its execution; C2: problems that allow the composition to continue to be executed, but not necessarily in its best way; C3: evolution of requirements; and C4: emergence of better services.

N-M Operations Replacement: ProAdapt allows changes in service compositions performed by (a) replacing a single service operation in the composition by another service operation or by a group of dynamically composed service operations (replacement of types 1-1 or 1-n), or (b) replacing a group of service operations in a composition by a single operation or by a group of dynamically composed service operations (replacement of types n-1 or n-m).

Simulated Web Service Infrastructure : The second prototype developed for the research includes a complete Web service infrastructure layer for the network simulator three (NS-3) [62] that can be used by other researches to perform experiments for service-based systems.

The contributions above lead to the publication of the following papers:

- Aschoff, R.R.; Zisman, A., "QoS-Driven Proactive Adaptation of Service Composition,". In Proceedings of the 9th international conference on Service-

Oriented Computing(ICSOC'11), Springer-Verlag, Berlin, Heidelberg, 421-435. 2011 DOI=10.1007/978-3-642-25535-9_28.

- Aschoff, R.R.; Zisman, A., "Proactive adaptation of service composition," Software Engineering for Adaptive and Self-Managing Systems (SEAMS'12), pp. 1-10, 4-5 June 2012 doi: 10.1109/SEAMS.2012.6224385.

We are also preparing a journal paper to be submitted to the TSE journal and have been invited to submit a chapter in the book Engineering Adaptive Systems. Moreover, we are planning to publish the ideas and results of the extension of our framework (See Chapter 5). The list below summarises the planned publications.

- Parallel Adaptation of Service Composition
- Behaviour Compensation of Service Compositions
- ProAdapt: A Proactive Adaptation Framework for QoS-Aware Dynamic Service Composition

1.6 Research Methodology

The research process is usually defined in two clear stages. First, the research questions are made, and thus some process is followed to answer those questions. This section describes the process undertaken to find the answers to the dynamic adaptation problem investigated in this report. Our research process was based on methods and good practice guidelines defined in the literature, such as the works presented in [42], [81], and [83].

In the work presented in this thesis, we follow a research process composed of eight-steps, namely: (1) Research Problem Formulation; (2) Literature Review; (3) Objectives Definition; (4) Research Design; (5) Data Collection; (6) Analysis of Data; (7) Generalisation and Interpretation; and (8) Preparation of Final Report. These steps are better described below.

Research Problem Formulation

The research problem formulation is the first step followed in the research process. This step must include an analysis of the importance of the topic of adaptation in the context of service composition. In this context, the specific area of concern is described, including what are the challenges or conditions to be improved.

Literature Review

Reviewing the literature is an important step to establish a theoretical framework and maintain a methodological focus. The critical reading of related work can be used to acquire insights for the proposal of new theories or methods, as well as to locate the research within a context, relating the findings to what has been done.

Objectives Definition

The goals of the research must be defined clearly and specifically to inform the reader about what the work described in this thesis attempts to attain. This step includes three level of abstraction, namely: (a) a general hypothesis, (b) sub-hypotheses, and (c) objectives.

Research Design

It is in the research design that we define the path that our research will follow in the context of how to gather and process the required information for

the thesis. In such case, we need to revisit the hypothesis and objectives and define the method for data collection and analysis. In our case, we decide to use a quantitative research method, which is useful to analyse proposed hypotheses through measurable variables.

Collecting Data

Considering that the research design is defined, together with the hypotheses and objectives, the next step is to collect data from the experiments in order to draw inferences and conclusions.

Analysing Data

Analysing the data or evaluating the results is the process used to extract information from the collected data and presenting it in order to test the hypotheses and confront the objectives of the research. Considering that the research is a work in progress, this step can be also used to revisit the hypotheses and objectives

Preparation of Final Report

Finally, the last step is to write the thesis with the results of the research conducted.

1.7 Outlines of this Thesis

The remaining of this thesis is structured as follows.

Chapter 2 - Literature Review

This chapter situate the current study within the body of literature and provides a representative review of works related to the research topic of this thesis. It includes a general discussion regarding adaptation in the context

of software engineering as well as a review of some of the works more closely related to the topic of dynamic adaptation of service compositions.

Chapter 3 - ProAdapt Adaptation Framework

This chapter introduces the current stage of the adaptation framework developed in this thesis to support dynamic adaptation of service compositions. The chapter presents an overview of the main concepts of the framework and the architecture created to support these concepts. The chapter also includes details of the process of adaptation and the techniques used to analyse events and predict failures.

Chapter 4 - Experiments and Evaluation

This chapter presents the prototypes developed to verify the concepts, techniques, and methods of the adaptation framework. For each of the prototypes, the analysis of the collected data is presented and discussed.

Chapter 5 - Behavioural Compensation Extension

This chapter presents an extension for ProAdapt to support behavioural compensation. It includes a running example with which the extension is explained and also the results and evaluation of the experiment performed.

Chapter 6 - Conclusions and Future Work

This chapter highlights the current contributions of the research done hitherto and presents the future plans for the work.

Chapter 2

Literature Review

Software systems often go through various stages in their development life-cycle in order to meet the defined requirements under the expected operating running environments. However, it is arguably impossible to anticipate the requirements and behaviour of all users or create a single best configuration for the system that works all the time. Moreover, requirements and especially the running environment changes over time, potentially invalidating previous agreements. This nonstatic environment forces software systems to support continuous adaptation in order to avoid unwanted behaviour or loss of opportunities.

There are various works in the literature dealing with adaptation approaches in the context of software systems, however, it is important to note that the focus of this research is adaptation specifically targeted at service composition. Nevertheless, some core concepts are common and important for a better understanding of the topic. The remaining of this section starts by introducing adaptation in the general concept of Software Engineering, and progressively address the approaches targeting service compositions.

2.1 Software Engineering Adaptation

The adaptation of software systems has been extensively studied in various branches of the software engineering field, including control engineering [134][44], software architectures [82][47], self-adaptive systems theory [29], autonomic computing [78], and fault-tolerant computing [123][112].

In control engineering, adaptation is concerned with the use of sensors to monitor the devices being controlled and actuators that are able to make corrections on the system according to the reasoning performed with the monitored values. Different adaptation techniques, such as gain scheduling [133], automatic tuning[148] and continuous adaptation [129] have been used in controllers for more than twenty years [7].

An example of adaptation in the context of control engineering is the method presented in [129] to reduce the acoustic feedback in hearing aids that contain a substantial amount of gain. The authors use a continuous adaptation approach, together with a delay in the forward or cancellation paths of the hearing aid plant to achieve acoustic feedback reduction of more than 15 dB, increasing the maximum insertion gain of a hearing aid using the approach. The approach is somewhat similar to the work present in this thesis in the way that it constantly adapts the filter coefficients based on the monitored signal, while our approach constantly adapts service compositions based on monitored QoS values.

In a similar way to control engineering, self-adaptive systems are able to autonomously evaluate their execution context and change their own behaviour in order to meet or improve previously defined requirements. For example, a software system can be designed as a multimodal display based on changing context

and user disabilities[131]. In other words, it is possible to select among possible output formats, such as sound, image, and text, in order to present information based on user preferences. A website can self adapt to increase the font size when presenting information for elderly users, or play sounds instead of presenting texts in case of blind users.

In a more ambitious goal, autonomic computing refers to the self-managing computing model named after the human body's autonomic nervous system. The main overall aim of autonomic computing is the design of systems able to manage themselves given high-level objectives specified by humans while keeping the complexity of the system itself invisible to the user. To accomplish these ambitious goals, IBM has proposed a conceptual guideline architectural named MAPE[35]. Figure 2.1 illustrate the MAPE model, which is consisted of a central knowledge base surrounded by four modules: *Monitor*, *Analyse*, *Plan*, and *Execute*.

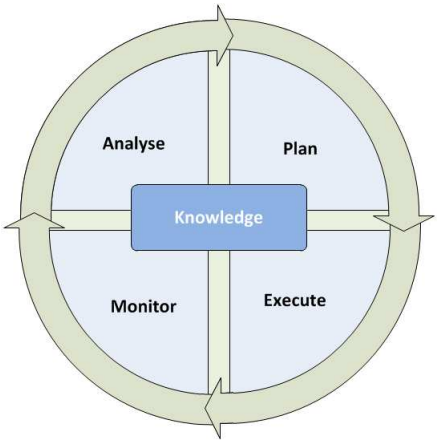

Figure 2.1: MAPE Architectural Guideline

The monitoring process is responsible to collect information about the managed resources and execution context. The Analyse module extract and correlate

information from the monitored data to model adaptation situations. The planning phase uses policies to guide different adaptation procedures. Finally, the planned actions are executed through controlled mechanisms. The MAPE model, however, is purposely only to be used as a guideline. It is up to the system designer to decide how each component is going to be implemented and where they are going to be positioned.

Regardless of the software domain, perhaps the main reason that motivates the need for adaptation is the faults that invariably occur in software systems, especially distributed systems. In many application contexts, such as in business process, the reliability of the overall system must be far higher than the reliability of its individual components[80].

Software reliability can be defined as the probability that the software will be functioning without failure under a given environmental condition during a specified period of time[141]. In other words, some systems are designed so that even in the event of faults risen by internal components, the overall system must still attempt to accomplish its goal while meeting the defined requirements.

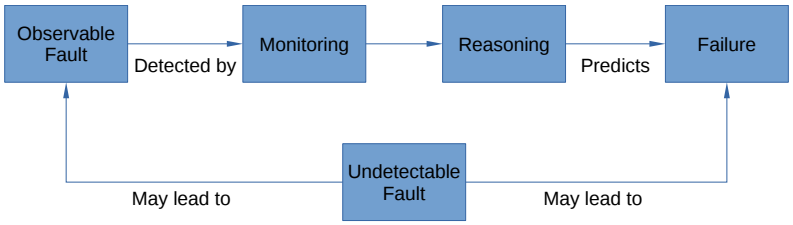

Figure 2.2: Failure prediction process performed for fault tolerant systems

The failure and fault terms are thus key to the understanding of system reliability. A failure is defined as a deviation from the conformance with the system requirements specification for a specified period of time, while faults are failures

incurred by internal component or interacting systems [61].

Figure 2.2 illustrate this relation between faults and failures in the context of fault tolerant systems. Generally speaking, as depicted in the picture, fault-tolerant systems compute the probability of one or more observable faults causing a system failure. A direct interpretation of such a system is that hidden or undetectable faults can easily cause system failures since they are not computed by the reasoning process. If those undetectable faults lead first to observable faults, then the fault tolerant system can work as expected.

By predicting failures, fault-tolerant systems can adapt accordingly to prevent such failures to happen and thus ensuring the system reliability. In the context of service-based system, failures of external partner services are identified as internal faults by the service composition. For example, whenever a service operation deployed in a service operation does not meet its expected QoS aspects, or cannot be invoked for some reason, a fault is observed.

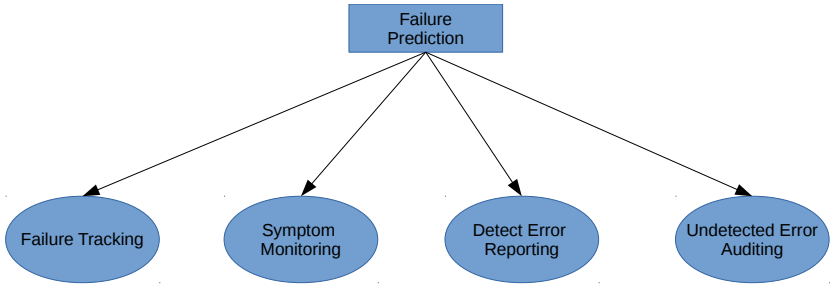

Figure 2.3: Classification of failure prediction techniques

As previously described, the ability to predict and prevent failures is of great importance to ensure the system reliability. At the same time, the continuous grow in complexity and size of the computer systems make almost mandatory the use of a solid fault management method. The remaining of this section presents an

overview of the four major branches of failure prediction techniques as presented in [126] and depicted in Figure 2.3.

Failure Tracking

Failure tracking is the method used to assert the probability of future failures based on the occurrence of previous failures. The work in [37] presents an offline Bayesian predictive approach which attempts to predict the probability distribution function of future failures given the occurrence of present and past failures. The approach is applied to the Markov process model known as Jelinski-Moranda [97] and besides been a complex offline method can also be used during runtime.

Failure Tracking can also be accomplished by exploiting the fact that failures can usually occur next to each other, either in time or in space. An example of work that examines such relation is [52], which presents a method for temporal and spatial correlation of failures in distributed system. This approach has some resemblance to our technique to predict faults in operations, service and providers (see Section 3.3.1.2).

Symptom Monitoring

Symptoms of a software system can be seen as side effect of errors. Before a failure can be observed in a system, there are some symptoms that can be detected and exploited to predict, prevent or at least mitigate these failures. In other words, failure-prediction techniques based on symptom monitoring analyse the system behaviour and context in order to detect symptoms that can be perceived as prelude of upcoming failures.

An example of symptom monitoring is presented in [88] through the form of a function approximation. The work attempts to predict the resource utilization of Apache web servers by collecting system variables such as memory use and then

building an auto-regressive model to predict the time of resource exhaustion.

The work in [104] uses instead a technique known as time series prediction. Generally speaking, monitored values are seen as sequence of data points in time. Time series forecasting is performed by modelling previous measured values to compute future values. In the case of the work in [104], the system variable is the memory usage of J2EE applications, which is monitored every twelve minutes. Future memory usage and resource exhaustion is predicted by combine rough set theory with wavelet networks to the collected data.

The previous approach retains some similarities to our own regarding the prediction of data values using time series forecast. As described in Section 3.3.1.1, we use exponentially weighted moving average to predict the expected service operation response time.

Detected Error Reporting

In some cases, collecting and reasoning about system symptoms promptly is not practical or indicated. In such cases, failure prediction can work using as input some sort of discrete system error report. More precisely, failure prediction using detected error reporting can be generalised as the reasoning function $f(r_n) = P_(n + k)$, where r_n is the error report given at time t_n and $P_(n + k)$ is the prediction of what may happen at time $t_(n + k)$.

This approach is employed in [84] to generate failure warning in high performance Linux clusters. The authors extended the Open Source Cluster Application Resources (OSCA) by introducing high availability (HA-OSCA). The proposed method collects and aggregates hardware sensitive information, such as temperature, voltage and power supply, and uses thresholds to predict and prevent failures.

Undetected Error Auditing

In the previous branches of failure prediction, the approaches focus on analysing an event that is observed or logged as a direct result of the use of a particular function of the system. In the case of undetected error auditing approaches, the idea is to proactively look for incorrect or undesired system states, even for those parts that are not actually in use at the moment.

This search for undetected errors, or latent failure is exploited in [9]. The authors propose a method to detect and predict latent sector errors in disk drive-based storage systems. The contribution of the paper is in the fact that such latent errors can be detected before the corresponding disk sectors are accessed and thus, potentially avoiding system failures.

This section discusses topics and works in the general context of software engineering for self-adaptable systems. Many concepts have been used in the ProAdapt framework, such as the precise definition of faults and failures, and the control loop of the MAPE architectural model. Moreover, two important concepts of the ProAdapt framework were designed as the result of the research done, reviewing general techniques for failure prediction. The spatial correlation of availability and the function approximation of the response time were both envisioned by adapting some ideas reviewed as general adaptation and failure prediction. The next section closes the gap by targeting specifically on adaptation approaches designed to work with service compositions.

2.2 Service Composition Adaptation

The highly dynamic nature where business processes for Web services operate, which usually requires access to an infrastructure without centralised governance, exacerbates the adaptation issue of service-based systems. In fact, the adaptation of service compositions constitutes a major research challenge for service-based systems[43][45][107].

Service compositions require access to numerous partner Web services that might evolve over time, changing their structures, behaviours, and quality or just becoming unavailable, potentially forcing the composition itself to adapt to these new circumstances. An adaptable service-based system is thus defined as the one that continuously monitors and modifies itself in order to satisfy new requirements and to adjust to new conditions imposed by changes of the executing environment [63].

Such definition for adaptable service-based systems can be viewed as an extension of what is expected from a fault tolerant system (see Section 2.1) by considering adaptations of service composition triggered not only by observable faults, but also extra events of interest. More precisely, the adaptations of service compositions revolve around four main objectives, namely (a) recovery, (b) context, (c) interoperability, and (d) optimisation [13].

The *recovery* is concerned with compensation actions for partially executed service compositions. The execution *context* of service compositions changes relatively often and adaptation strategies must act accordingly, with actions such as replacing partner services to accommodate changes. While standards proposed for Web services greatly reduce *interoperability* issues, the heterogeneity of Web

services creates a challenge. Web services may use different protocols and interfaces, which could cause signature mismatches between the service composition and provider entities. Finally, *optimisation* is the need to continually improve the general system performance.

In Chapter 1 an overview of the general adaptation classification is introduced and briefly discussed, which is here illustrated by Figure 2.4. In this section we revisit the previously presented concepts and expand the discussion to introduce some of the various approaches that have been proposed to accomplish the aforementioned adaptation goals.

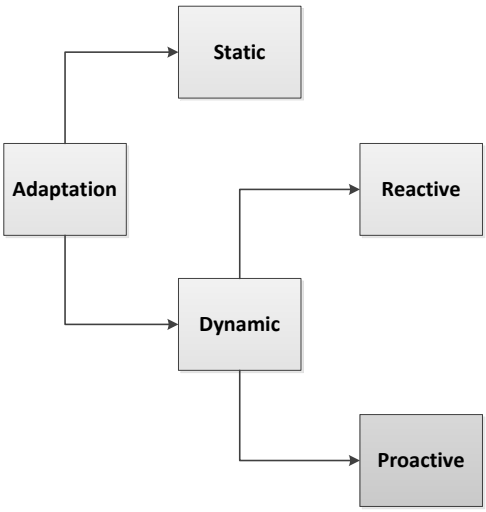

Figure 2.4: Adaptation Approach Classification

As shown in Figure 2.4, the first level of categorisation defines adaptation in static and dynamic ones. Dynamic adaptation approaches can be further categorised as reactive or proactive.

While static adaptation focuses on assembling the pieces of the service compo-

sition during design time or while the system is not running, dynamic adaptation occurs while the system is up and running. Dynamic adaptations can be classified as reactive, when the idea is to correct some observable problem, or proactive, when there is an attempt to predict such problems and thus act before they happen.

Reactive adaptations aim to resolve the internal faults that can happen in distributed systems and specially service-based systems in order to allow the correct execution of system as a whole. This can be performed in a number of ways but usually the faulty activity is identified and replaced. For instance, a service composition may not receive a reply from a particular service operation. In this case, a reactive adaptation must retry to invoke the operation or replace the identified unreachable operation with another candidate service operation.

A Proactive adaptation on the other hand, aims to predict and prevent future faults or failures of the service composition. As defined in [126], the prediction of faults and failures is concerned with the identification of the occurrence of a problem in the near future based on an assessment of the current state of the system. More specifically, in the scope of service-based systems, fault prediction is concerned with the assessment of what is the impact of service misbehaviour, or of a group of services, in other parts of the service composition, while failure prediction is concerned with the impact of observed and predicted faults in the service composition as a whole.

The dynamic and competitive environment of business processes demands continuous evolution of the business processes and their implementation. Given this evolution demand, another example of proactive adaptation is to improve the service composition even when there is no fault, failure or degradation of the

expected requirements. For instance, a proactive adaptation can continuously search for new candidates of service operations in order to optimise the quality of service characteristics of the service composition. The presented classification, however, should be view only as a guideline. In fact, an adaptable service-based system may employ more than one strategy to cover different scenarios and requirements.

2.2.1 Static Adaptation

The first approaches for adaptable service-based systems have focussed on the particular problem of automatic service composition during design time. In other words, these approaches try to generate an executable service composition with concrete service operations based on an abstract template and predefined requirements while the system is not in operation. More precisely, the initial approaches for automatic service composition are mostly based on anticipated interactions of services and matches between input and output functionality [95][107][16].

In [8] a framework is presented which support adaptation of service-based applications targeted at mobile resource-constrained devices in the heterogeneous Beyond-3G Networking (B3G). The approach uses a Java programming model for adaptive applications named Chameleon [14], which allows adaptation with respect to a dynamically provided execution context based on *generic code*. This generic code is used to generate a set of different Java components providing different ways of implementing a provider/consumer application. The main disadvantage of the approach is the fact that adaptation happens at discovery time and cannot evolve during run time.

The major drawback of these adaptations focused on the design time is the fact that they only work when the system is not in operation. This condition means that for any changes required in the system, it needs to be stopped, recompiled with the required changes, and only then return to operation.

Automatic service composition while the system is not running is an important step to help service-based system designers to produce an executable service composition with concrete operations. It is also important when there are major changes to the system and there is a need to identify new interactions between partner services. Nonetheless, these approaches do not fit well when the system is already running due to the great reaction time. In other words, it takes a long time to adapt the system and all current requests to the business process must be finished or dropped, which decreases the overall performance and reliability of the system.

2.2.2 Reactive Adaptation

The solution to the above problem is to enable automatic service composition during the system runtime, what is usually called dynamic service composition[1][25][27][34][120][121][15]. More precisely, approaches for dynamic service composition are able to identify, aggregate, and replace compatible services operations during execution time.

Initial approaches to dynamic service composition have been proposed to support adaptation of service compositions in a reactive way, where faults are identified and corrected at the same time as the execution engine is running, triggering changes in service composition based on predefined policies [10], self-healing of

compositions based on detection of exceptions and repair using handlers [36], and KPI analysis [76].

In [22] the authors present a framework based on planning techniques which identifies recovery activities for context changes and constraint violations that occur at run-time in order to guarantee that business goals are still achieved. The work consists of a goal-based approach for adaptation where processes are modelled as Adaptable Pervasive Flows (APFs) [21] and monitors are responsible for triggering adaptation on demand. APF is a workflow-based paradigm for the design and execution of pervasive applications. It allows the workflow to be described as a set of abstract activities in terms of the goal they need to achieve. During run-time, the workflow can thus be refined in an executable process considering the execution context, available services, and workflow goal. In order for the approach to work, the available services and goals must be manually annotated.

Two similar rule-based adaptation approaches based on monitors are proposed in [4] and [11]. The former is a context-based adaptation approach for service compositions that uses negotiation and repair actions. In [11] the authors present a strategy to re-organize a service composition in order to make it suitable for monitoring and re-configuration triggered by changing in the execution contexts. The main drawback in both cases is that the adaptation strategy becomes constrained by the monitor rules and adaptation actions that must be defined during design time.

Another reactive approach based on dynamic re-configuration is presented in [145]. The approach consists in repairing failed services with available candidate services and ensuring the new composition still meets the user specified end-to-

end QoS constraints. The approach uses an inspection algorithm that designed to take into consideration the re-configuration regions of the service composition. In this case, it is possible to focus the adaptation only in the affected region, in which case the business process can be less affected by adaptation. The approach also uses *extended constraints* of QoS values as a way to relax the original constraints of regions where some sub-process that meets the criteria is not found.

The work presented in [6] describes an approach to automatically recover from system failures by using AI planning. The main goal is to capture the system state after a failure, and by using the plan, reach an acceptable recovered state. The AI planning chosen is based on three artifacts: domain, initial state, and goal state. The domain is immutable, and encodes the semantics of the system. The initial and goal state may change, and are the current system state and the desired system state respectively.

The work presented in [6] reduces the downtime associate with traditional recovery techniques, by reducing the time needed to go from a failure state to a recovered state. This is accomplished by using an AI plan that dynamically constructs a knowledge base about status of failure plus different paths to get in recovered states. However, despite it is successful in fast recovery from failure status; the approach does not take into account failure prediction, what could improve the overall system performance. Moreover, the domain artifact used in AI planning cannot be changed during runtime, or in other words, it does not support changes in the business environment.

[93] presents a simple way to dynamically compose Web services by changing the semantic of WS-BPEL language while maintaining the syntax unaltered. The authors defend that by following an established specification such as BPEL no

additional tasks are need from programmers. The solution works by intercepting calls to BPEL partners and, based on specific criteria, looking through the repository and redirects the call to the best available partner.

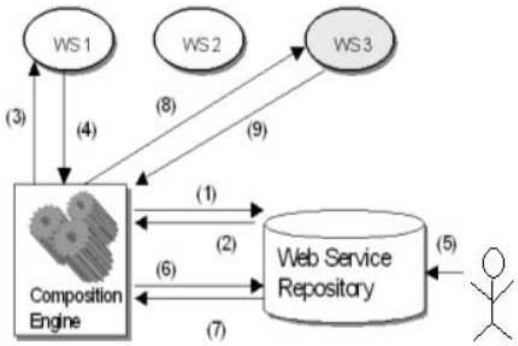

Figure 2.5: Example of ruintime adaptation using dynamically composed Web services

Figure 2.5, summarises the work in [93] through an example which contains an initial serial composition of two Web services (WS1 and WS2). The first steps performed are to check the repository for changes (step 1 and 2). Let us suppose that there were no changes in the repository, then the engine proceeds by invoking WS1 (steps 3 and 4). Before every partner invocation the engine checks for changes in the Web service repository, which means that before invoking WS2 it is verified one more time (steps 6, 7). Now, suppose that a new partner (WS3) was registered at the repository and that it becomes more appropriate to the composition than WS2. In this case, the engine is able to perceive this change and then acts to alter the correspondents call (steps 8 and 9).

The main advantage of the approach presented in [93] is the ability to perform runtime adaptation without syntactic changes in the BPEL language. The engine is very simple and allows adaptation without system interruptions. Moreover,

the work presents an intuitive way to implements a Web service repository by grouping semantically equivalent Web services and using XML base files instead of complex databases. However, the approach lacks the ability to adapt to changes at business level, in other words, the solution considers a static specification of the business process and its requirements.

Another problem of the solution presented in [93] is the assumption that every semantically similar partner has the same interface. It is a difficult assumption and the authors are already working on how to solve this by introducing semantic Web services. The same is valid to the structure of the repository. Grouping partners only works because of the assumption of identical interfaces. Moreover, it seems that this repository was thought to work only inside a company, since we cannot expect the kindness of external companies to register in their process. On the contrary, we should expect a tool to look in public repositories and extract the desired information. Moreover, in the approach, policies are analysed operation by operation (one-to-one) and the engine does not consider QoS of the entire process. This procedure could lead to optimisation of local problems that does not correspond to an optimal global solution. This is particular true when considering SLAs with fields that are traditionally inversely proportional, like cost and estimated response time.

This one-to-one limitation is handled with an approach for replacing faulty services in a group while maintaining QoS constraints [91]. The idea is to define re-configuration regions to release planning overhead. Not only the work presents a way in which services may be replaced using one-to-one, one-to-many, or many-to-one service mappings, but by defining hierarchies regions and by searching replaceable services only within these regions, re-configuration overhead is lowered

(see Figure 2.6).

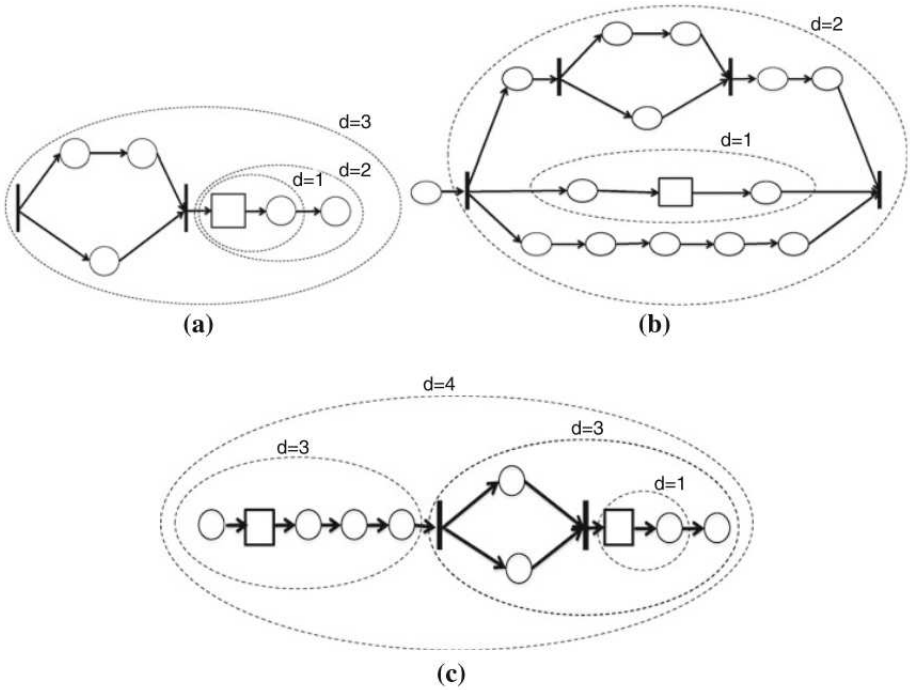

Figure 2.6: Reconfigurable Regions (extracted from [91]).

Moreover, the idea of organizing services into classes facilitates replacing procedures. The approach enables changes at BPEL structure to be performed automatically, however, context changes are not considered. Additionally, as stated before, single service failure does not necessarily means overall SLA disagreement.

In the above approaches the strategies for adaptation are concerned with the replacement of a service in the composition by another service or a group of services. These approaches are fine when the candidate services are compatible, or in other words, when the interoperability is not affected. As described in Section 1.1, in order for a request entity to interact with a provider entity, both parties must agree on the interface and behaviour of the desired service.

These negotiations usually occur during design time, when a requester agent is configured according to the expected behaviour of the desired service agent. In a dynamic environment, however, due to the heterogeneity and independent development of Web services, some interaction incompatibility may arise. To ensure Web service interoperability in such dynamic execution context constitutes a research challenge [20][89][139].

The work in [79] suggests two ways for dealing with mismatches in service compositions, namely the creation of stand-alone adaptors and the use of aspect-oriented adaptation by transforming the data types of the operation signatures, instead of identifying replacement operations that can accept the dependency mismatch. The approach relies on the use of patterns for capturing the recurring differences and providing solutions to these differences.

As it was to be expected, the main disadvantage in the case of reactive approaches is that correction procedures start only after a fault has already occurred. Furthermore, there is the possibility of a fault to induce an irreparable system state. For example, in order to recover from an unavailable operation, the adaptation approach may change the unavailable operation for another candidate operation, however, the time lost trying to invoke the unavailable operation and the time spent with the adaptation may well cause a nonconformity with the service level agreement predefined for the composition.

2.2.3 Proactive Adaptation

To overcome the above problem, some approaches have recently been proposed to support adaptation of service compositions in a proactive way [40][87][105][135].

These approaches extend the functionality provided by adaptable system based on reactive adaptation with the prediction and prevention of general unwanted behaviour in the execution of the service composition.

A proactive adaptation approach may work by observing a single internal fault in order to predict and prevent future faults or by monitoring the system status in order to assess the risk of a first fault. Nevertheless, dynamic service composition techniques which use proactive adaptation methods usually outperform reactive adaptation approaches due to the ability to avoid faults.

Some approaches have been proposed to support multilayered monitoring and adaptation of service compositions [57][122][144]. The work in [122] uses adaptation taxonomy and templates (patterns) created during design time to represent possible solutions for adaptation problems.

The work in [57][144] dynamically identifies cross-layered adaptation strategies for software and infrastructure layers. In [110] the authors propose approaches based on aspect-oriented to support adaptation of service compositions with support for QoS aspects.

One of the first works to use a proactive solution for dynamic service composition was PREvent [87], which was designed to support prediction and prevention of SLA violations in service compositions based on event monitoring and machine learning techniques. The prediction of violations, however, is calculated only at defined checkpoints in a composition based on regression classifiers prediction models. It is important to support changes in compositions due to problems that may occur in any part of the composition, as supported by ProAdapt.

In order to predict the probability that a change will actually effect the running service the work presented in [111] presents a methodology named change impact

probability (CIP). In order to compute the CIP, the authors provide a grading model for QoS values and changes that depends upon the degree of influence on the SLA. Next, a prediction is made to assess how long the service will remain at a given QoS level. Another model is then used to compute the possible start time of each service in the composition. Finally, this information is used to compute the CIP function and thus the probability of a change affecting the service composition in execution. The main problem of this work is that it does not consider structural changes of the service composition or the requirements defined for the whole composition. It also needed to better define a threshold for the CIP function.

The works in [105][135] advocate the use of testing to anticipate problems in service compositions and trigger adaptation requests. The approach in [135] supports identification of nine types of mismatches between services to be used in a composition and their requests based on predefined test cases.

The work in [105] is similar to the work in [111] commented above in a way that it tries to diminish the impact of unnecessary changes triggered by proactive adaptation using a combination of online testing and monitoring technique in order to determine failure probability within a confidence interval. The approach is constituted of five main steps as presented in Figure 2.7, namely (1) *Determine Representative Data*, (2) *Determine Current Confidence*, (3) *Execute Tests*, (4) *Predict Failure Occurrence*, (5) *Decide on Proactive Adaptation*.

The first step determines which of the data points collected so far are representative of the service that is being observed. This is important because during the execution of online tests, the service might have changed or new monitoring data might have been collected (from the SBA instances running in parallel).

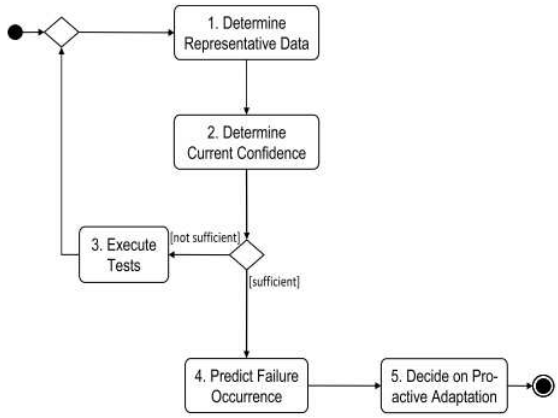

Figure 2.7: Steps performed by to achieve proactive adaptation with confidence

This means that some of the data will not be representative anymore or that new, representative data should be considered. In step 3, test cases are generated and executed in order to gather additional, representative data points for failure prediction. After the previous steps have established a set of representative data that exhibits the required confidence for failure prediction, Step 4 predicts the actual occurrence of the failure. Step 5 decides on the actual proactive adaptation of the SBA instances. The decision on such an adaptation is based on the predicted failure probability from Step 4. For example, proactive adaptation is triggered if the prediction is above a predefined threshold.

The solution takes into consideration two different approaches to start the described steps. The first one considers triggering step 1 as soon as the monitoring process discovers a failure. This strategy has the clear disadvantage of delaying the adaptation, but it reduces the cost related to testing since it is triggered when the potentials need for a proactive adaptation occurs. The second approach triggers step 1 after each change of a partner of the SBA. Here, one the contrary of

first approach, adaptation can be performed early, on the other hand, the testing routines could be intensive and the collected data may never be used. Moreover, the creation of test cases is not easy and the paper does not specify how test cases are created for modified compositions. Additionally, the work is focussed only on service binds, which means that structural changes in a workflow are not considered and new services cannot be dynamically found.

Online testing is also employed in [127]. The work presents an online testing and monitoring framework focused on dynamic selection of service operations by using quality prediction. Such proactive quality prediction is performed by selecting test cases and performing online testing in parallel of the execution of the service-based system to gather additional evidence for failure on top of the monitoring process.

Similar to the approach presented in this report, in [18] the authors advocate that the management of service compositions during runtime needs to consider the structure of a composition and the dependencies between the participating services, and propose an approach that determines the impact of each service in a composition on its overall performance.

The solution presented in [106] uses a combination of proactive adaptation techniques along the phases of the service life-cycle. The core concept is to use these techniques to determine deviation from requirements based on monitored failures. In the approach, functional and nonfunctional requirements are formalized and the service composition also needs to be formalized in order to support automated checks. The approach attempts to solve the problem of whether the failure of a single service leads to a violation of the composition as a whole.

The work described in [106] reasoning is that when the expected behaviour

of some service operation invocation deviates from its assumption the past monitoring data together with the assumptions about the not yet invoked services operations and the service composition specification is checked against the general requirements (SLA).

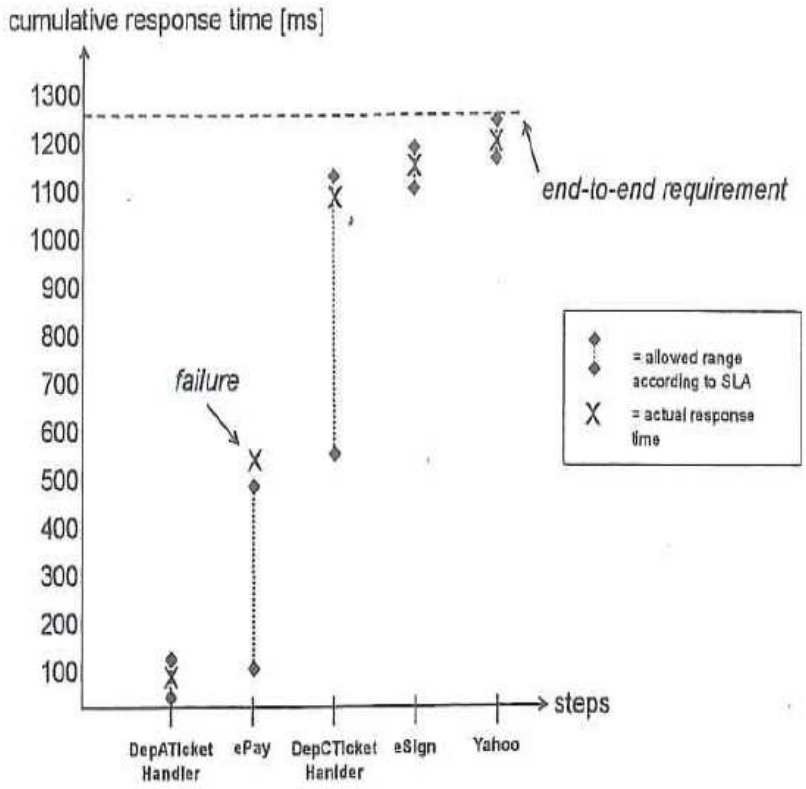

Figure 2.8: Requirements Monitoring Steps performed by to achieve proactive adaptation with confidence (extracted from [106])

If the requirements are still satisfied, the composition can continue its execution, as depicted in Figure 2.8, otherwise it must be adapted. The approach manages to cover adaptation along service life-cycle and formalisation of func-

tional and nonfunctional requirements allow proactive runtime check of service compositions. However, the solution covers only one-to-one maps and does not specify how options to replace services are discovered. Moreover, it does not consider different adaptations for multiple parallel executing instances.

A solution that combines proactive adaptation and reactive adaptation is presented in [72]. The approach presents a new model to represent service composition in order to include execution state of each service within the composition in order to provide self-protecting and self-healing. The work firstly introduces the notion of an execution instance as a memory structure to record information concerning a single execution of the composite service.

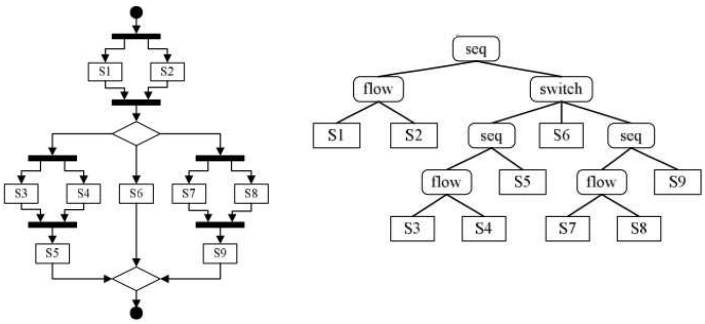

Figure 2.9: Structured view of an example of execution instance (extracted from [72])

Figure 2.9 presents a workflow and tree view of an execution instance as defined in [72], where leaf nodes are referred to partners in WS-BPEL, while other nodes present control logics. The execution instance maintains information defined as attributes on nodes and edges as *QoS Feature* and *Execution Feature*. A QoS feature is a set of quality parameters associated with service nodes and control nodes. An execution feature contains control and execution information related to edges. The notion of execution instances is also used in the work

described in this thesis to support independent adaptations. The execution instances used by ProAdapt, however, are a bit more flexible to support structural changes, faster computation of candidate operations, and faster verification of SLA satisfaction of logic regions and the composition as a whole.

The work presented in [72] constitutes a two-stage adaptation model that includes *How*, *Where*, and *When* adaptation actions take place. The adaptation is performed by three actions, namely (a) request redirection, (b) request reconstruction, and (c) execution instance revision. The request redirection (a) is the action involved in proactive adaptation and triggered by new service request. The request reconstruction (b) is performed in a reactive way and triggered by arrival of a failed or error response. The execution instance is revised (c) both in proactive and reactive way by updating the QoS features.

More specifically, the required service is invoked directly if it is a dependable one. The request is redirected on condition that an alternative service can be found (the execution instance is updated accordingly). In the case of undependable or no alternatives, the approach suggests the review of quality setting (with a user) or the termination of the current execution. The adaptation process can be triggered by a service change either if a recommended service is founded to be unusable in the future execution or if a service is identified as possible threat to break constraints defined in the SLA.

The combination of a proactive approach with a reactive approach presented in [72] proved to be useful when dealing with QoS deviations. The defined concept of execution instances provides a way to formally represent system requirements as well as to store dynamic system state. However, the work does not specify how a service community could work. This may have a great impact on the solution,

specifically regarding the time to select a substitute service. Moreover, it does not consider changes at composition itself or the service composition SLA as a whole.

This previous sections present a comprehensive review of the related work to this thesis in the specific context of adaptation for service compositions. By introducing the related work in a simple but efficient categorisation, it was easier to present the approaches in a close to evolutionary way. Starting from static approaches, passing from reactive adaptation and finally ending in the dynamic and proactive adaptation approaches. By analysing the previous discussed works, it seems that while there are some contributions about how to dynamically adapt service compositions, these ideas are still scattered and cannot be envisioned in practice yet. The next section addresses the problem of services selection, regardless of the other adaptations concerns involved.

2.3 QoS-Aware Service Selection

The previous sections presented an overview of the various adaptation approaches for service compositions. Despite the differences in how and when the need for adaptation is identified or changes actually enacted, most adaptation approaches rely on the selection of different concrete operations to perform the various business process tasks. It is common among adaptation approaches to assume that service operations are functionally equivalent and can be replaced by each other, despite the interoperability issues that may arise. However, even under such assumption, it is unrealistic to consider that different services have the same QoS values and choosing candidate services with different QoS values for a single ser-

vice operation may direct effects the service compositions[137].

The number of functionally equivalent services with different QoS values such as cost, reliability and response time have been consistently growing recently [140]. The problem of how to select and combine services to meet the QoS requirements of the business process has being one of the main issues of service compositions.

An important discussion is then on the assessment of such QoS values. Service providers, as the entities responsible for maintaining and making available its services agents, sometimes shares the QoS information along with the service description using techniques such as reputation [71] or fidelity [70]. The provided values, however, are usually considered as unreliable because such values are based on subjective estimates from customers in different execution contexts.

When it comes to third party evaluation of QoS values for services, testing services directly to obtain such values [94] [100] and relying on previous feedback is usually what is employed [142].

The QoS-based service selection work presented in [94] uses statistical variances and applies linear programming to decide which particular operation should be selected based on end users constraints. The approach works as follows. Given a set of service operations with the same functional properties, a QoS matrix is formed where each row represents a candidate operation, while each column represents one of the QoS criteria for the particular operation.

The work presented in [94] assumes that QoS criteria can be collected via active monitoring and users feedback. The active monitoring is similar to our approach. It considers that each service operation execution can be actively monitored by the service requester. The difference in the case of ProAdapt is that we consider not only the execution time of the operation, but the availability

as well, and uses the monitored information to model and predict the behaviour of each deployed service operation. In [94], only service selection is considered, the composition, and thus, the impact of combining services with different QoS values is not considered. Another point to note is that the approach uses a maximum value to normalise the QoS matrix. This maximum value is debatable, and creates uneven distributions based on nonexistent values. Moreover, different QoS values are considered the same, even when they have clear different range of values. This is one of the reasons why the normalisation performed by ProAdapt considers the current minimum and maximum observed QoS values to compute a factor between zero and one (see Equation 3.14).

A different approach which targets the prediction of QoS values based on ranking computed with users reports is presented in [136]. The approach creates the ranking by analysing QoS values advertised by service providers and also the perception of the real quality of monitored services by consumers. This values can be misleading due to false rating shared by both service providers and service consumers. However, the approach employs a trust and reputation model to address this problem.

Another example of trust models used to categorise quality aspects of Web services is proposed in [38]. The approach establishes different levels of trust for service costumers and provides an estimate of the quality aspects of a service-based on a weight function that takes into consideration the trust levels and different levels of importance for specific metrics of interest. Similarly, in [90] a service selection approach is proposed to classify service QoS values regarding the trustworthiness of provided data. The work considers both data coming from service providers and feedback from users. In the case of data coming from the

providers, the observed QoS values are used to revise the informed values, while in the case of user feedback a weight function is used.

In a slightly different approach, the authors in [39] present an approach for decomposing the overall QoS values provided for composite services into their composing services. The approach considers value provided for the composition as a whole as well as trusted and known QoS values of the internal services in order to make estimations. The biggest limitation of the approach is that the internal logic of the business process must be disclosed, which is usually not the case.

For composite services and in situations where a ranked list of expected values is not enough, we can try to find the probability distribution of the desired QoS value [68]. The work presented in [68] uses the knowledge of each probability distribution of services used in a composition to compute the probability distribution of the service composition as a whole. Such distributions can be used to better understand the expected behaviour of the quality value of a composite service and to assess the probability to meet the desired service level agreement. A problem with such approach is that it relies too much on given QoS data from the component services. This data must be gathered somehow and updated frequently to produce the desired results.

Considering a set of service operations with their respective QoS parameters, the next step in the service selection process is the actual choice of the operations that will compose the executable business process. This optimisation problem has been broadly discussed in the literature and is known as service composition problem [3] [103] [69] [98].

Formally speaking, giving a service composition S_k with a set of abstract

operations $L_k = \{A_1, A_2, ...A_n\}$ and for each A_x a set of candidate concrete operations $R_x = \{C_x^1, C_x^2, ..., C_x^m\}$; The problem of service composition is to find a configuration $G = \{C_1^a, C_2^b, ..., C_n^r\}$ that satisfies defined requirements, such as KPI and QoS.

$$NC(S_k) = \prod_{i=1}^{n} m_i \tag{2.1}$$

The number of possible configurations for a composition is given by Equation 2.1, where S_k is a given service composition with n abstract operations and m_i is the number of candidates operations for the specific abstract operation A_i.

Essentially, Equation 2.1 indicates that the number of possible configurations grows very fast in relation to the number of candidates and abstract operations. Verifying all possible combinations using a brute-force algorithm can be very time consuming. As surveyed in [132], different approaches have been proposed to handle the problem of service composition by using different techniques, such as linear integer programming [143][66], mixed integer programming [5][128], or genetic programming[24][53][85].

A linear integer programming method is a mathematical optimisation approach to a problem that can be modelled with all unknown variables as integers. In [143] such technique is used in an attempt to find the global optimal solution for a service composition. The idea is to find the best solution for every possible execution path and then merge the results together. This merging procedure attempts to solve the conflict created when two different optimal paths use different operations for a common branch, but a global optimal solution is not guaranteed.

In the case of mixed integer programming the restriction upon the domain of

the unknown variables are relaxed to just some of them. In order words, only some variables are required to be integers. An example of mixed integer programming can be view in the general optimal approach presented in [5]. The approach considers compositions with internal loops and presents a strategy to deal with the problem of no valid solution by using negotiation techniques. Differently from our work, stateful Web services are considered, but the approach fails to model conditional activities.

Another global optimisation approach based on mixed integer programming is proposed in [128]. Unlike [143] and [5], the work presented in [128] completely considers conditional activities and does not need to compute all possible paths, since it models the conditional activities with probabilities. The approach is able to compute an optimal solution in a reasonable time, but it fails to consider the dependencies that may exist between service operations. Moreover, the solution would require modification to be able to find a solution for a running instance.

While these previous integer programming approaches attempt to find the best solution to the service composition problem, some other approaches target heuristic suboptimal methods. A hill-climbing algorithm, where the basic idea is to start with a first solution to the problem and then repeatedly improve it until some constraints are met, is presented in [103] while generic algorithms are used by [24],[53], and [85].

The previous approaches can be very efficient in finding optimal solution for the problem of service composition, however, they present at least two major problems. First, they fail to consider the dependencies between services operations. An exception is [53] which is one of the first works to consider the interface matches between services as part of the problem to find an optimal service

composition. Second, they lack the simplicity to be used on running execution instances.

In this section, various concepts related to service selections are discussed, such as service discovery, categorisation, and composition. Even though some approaches could potentially be handled as black boxes in a service-based adaptation system, they are usually very attached to the base adaptation model and goals. A clear difference on the approach used in the ProAdapt framework is that such composition, as well as the identification of the need for a new configuration, needs to be computed as fast as possible. Even more than when comparing with other adaptation approaches dues to the fact that more verifications are to be performed and more adaptations are possible. The next section presents a general discussion regarding the adaptation topics reviewed in this chapter.

2.4 Discussion

The current approaches for dynamic service composition are still fragmented and in their initial stages of development. More specifically, the majority of the approaches reviewed try to focus in specific points of the adaptation challenge and do not offer a complete framework to support the implementation of the proposed ideas. In other words, while there are some contributions about how to dynamically compose Web services, these ideas must be integrated in a framework that supports the specification of business processes, execution of service compositions, identification of candidate services, rebinding of deployed operations, and dynamic changes in the composition workflow logic.

Existing approaches have at least four common drawbacks which are listed

below.

General Adaptation Scope - In existing approaches, when it is necessary to adapt a service composition, the changes executed in a composition are reflected in all instances of this composition. For instance, consider an invoke activity I defined for a service composition that requires a service agent A. If for a particular instance of a service composition, A needs to be changed by another service agent B, all instances of the service composition would have to use B as well. This is a clear limitation when trying to handle issues that are particular to a specific instance of a service composition.

One should not confuse adaptation of instances of execution with the dynamic adaptation of service compositions while the processes are still executing. While the first focus on a particular session or execution instance, the other is simply the ability do adapt without stopping the service composition or execution engine.

For example, in the case in which the time to execute an operation O in a service composition instance SC is greater than the expected time, causing a nonconformity of the SLA for composition SC, it is necessary to adapt the composition instance to respect the SLA; i.e., the adaptation may require other operations yet to be executed to be replaced. However, this delay in the execution time of operation O may not cause problems in other composition instances that use O. Therefore, changes in all composition instances that use O are unnecessary.

Prediction Based only on QoS Values - As outlines above, there are some proactive approaches to adapt service compositions, but they are mainly

focussed in predicting the impact of QoS values that deviate from the contract (SLA). However, there are other important points to cover such as the ability to predict the availability of a certain service operations and the ability to predict the impact of the need for adaptation in other parts of the service composition and also other compositions deployed in the same execution engine.

Limitation to Fault Tolerance - Existing approaches are limited to support adaptations due to faults in service compositions. The ability to enable the continued execution of a business process even in the presence of internal faults while still satisfying system requirements is perhaps the most desirable feature of service composition adaptation approaches. However, there are other situations in which no fault is observed or expected in a composition, but the system should adapt or could benefit from adaptation. For instance, adapting composition due to new requirements or optimising the composition when new better candidate services become available are both desirable features that have no relation to internal faults.

User Interaction - There is no question that complex and large business processes inadvertently require the interaction of the user, either to provide input for the system or to execute required tasks. Such interactions with users are usually not covered by adaptation approaches. Future adaptation approaches should be able to support activities that describe the user interaction due to its relevance to real business processes. This particular activity, however, is usually poorly supported by service composition execution engines and is not covered by ProAdapt. We recognise the importance

of the user interaction, but ProAdapt does not currently supports it.

From the four drawbacks listed above, ProAdapt covers three of them. Missing only support for user interaction. We are, However, considering to extend our research and add support for user interaction (see Section 6.1).

2.5 Summary

This chapter presents a general map of the research field and position the work described in this thesis within the context. This is done by first presenting the general topic of adaptation in the context of software engineering, where the concepts of adaptation in fields such as control engineering, software architectures, self-adaptive systems theory, autonomic computing, and fault-tolerant computing are discussed.

With the overall adaptation systems presented, the chapter reviews the related work in the area of adaptation for service composition, and specifically discusses the topic of service selection. Finally, the chapter presents a general discussion regarding the surveyed approached.

The work described in this thesis complements current approaches by providing a fully operational framework to support dynamic adaptation of service compositions tackling three out of the four issues presented above. The user interaction issue is the only that is still to be covered. However, this is already a work in progress that will be discussed in Chapter 6 (Future Plan).

Chapter 3

ProAdapt Adaptation Framework

As discussed in Chapter 1 and Chapter 2, there are still some open challenges regarding the topic of dynamic adaptation of service compositions. In this context, this chapter presents a new proactive adaptation framework named ProAdapt. The framework aims to improve the performance, reliability, and general conformance of business processes with system requirements when compared to traditional static and reactive adaptation approaches.

ProAdapt extends current approaches for dynamic service composition by proactively and individually identifying the need for adaptation for each parallel running instance of a service composition, while avoiding unnecessary changes to the service composition, and distributing load request among different service operations when necessary. It is an event-driven and QoS-aware fault tolerant service composition framework that uses monitoring and prediction techniques to prevent possible execution failures, and to continuously improve service composition.

The framework supports adaptation of service composition triggered by *four*

different classes of situations:

- **C1** - Events that cause the composition to stop its execution. This class includes events such as unavailability or malfunctioning of a deployed service operation;
- **C2** - Events that allow the composition to continue to be executed, but not necessarily in its best way. When the network link is congested, for example, the response times of some operations may be greater than expected. Such fluctuations in the response time may require adaptation in order to comply with SLA parameters of the composition;
- **C3** - Evolution of requirements. The service provider may decide to update the expected QoS parameters of the deployed service compositions. If the current composition violates these new values, adaptation is required;
- **C4** - Emergence of better services. Even when the system is running without any problems, some events, such as the availability of new cheaper and faster services can be used to improve the service compositions.

ProAdapt allows the reconfiguration of service compositions performed by (a) *replacing a single service operation in the composition by another service operation, or by a group of dynamically composed service operations* (replacement of types 1-1 or 1-n); or (b) *replacing a group of service operations in a composition by a single operation, or by a group of dynamically composed service operations* (replacement of types n-1 or n-m). Furthermore, ProAdapt considers dependencies that may exist between the signatures of service operations within a composition

in order to avoid changes in the composition that may lead to mismatches of the signatures of the operations.

The chapter is structured as follows. Section 3.1 presents a first insight of Proadapt framework. In Section 3.2 ProAdapt's architecture is presented and each component detailed. Finally, the adaptation process itself is described in Section 3.3.

3.1 Overview

As mentioned above, ProAdapt is a proactive adaptation framework to support dynamic reconfiguration of services compositions execution instances in order to improve the performance, reliability, and general conformance of the compositions instances with their requirements. To accomplish these ambitious goals ProAdapt continuously monitor and analyse events of interest generated in the execution environment in order to detect and prevent faults that could potentially lead to compositions failures (see Section 2.1 for a review of faults and failures).

Figure 3.1 summarises the general adaptation process adopted by ProAdapt. As show in the picture, various events are continuously monitored and reported to be analysed. During the analysis phase, the framework verifies the reported events, together with running execution instances, templates, and default binding information, to assess the need for adaptation and to perform any required preprocessing, such as marking service operations as potentially unavailable. Finally, during the decision and enacting phases any executions instance, template, or binding information can be modified.

One key aspect of ProAdapt is how the *execution engine* supports service

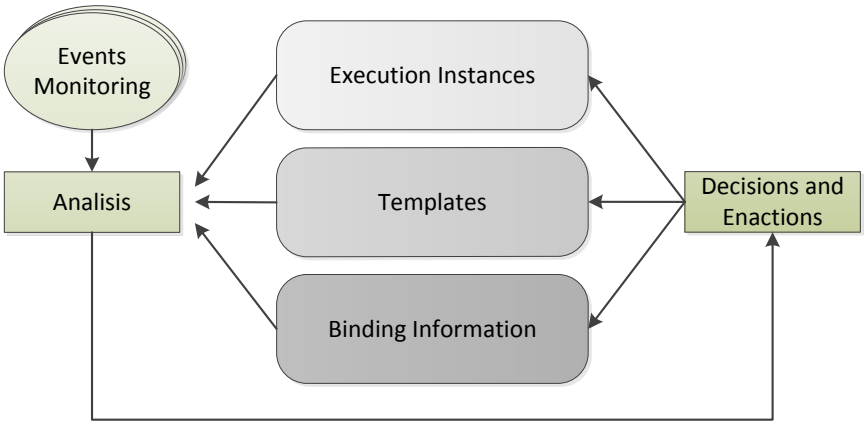

Figure 3.1: ProAdapt Overview

compositions. As described in Section 1.2, an execution engine is the piece of software responsible for the execution of business processes described in the form of executable service compositions. Different service compositions can be deployed in an execution engine, and for each request of a particular composition, a *private session* must be maintained in order to individually and correctly parse input and output parameters.

In the same way that a Web Service Description (WSD) contains the abstract part for the general definitions of the web service and a concrete part for the binding information (see Section 1.1), for each service composition SC_n deployed in an execution engine, we have (a) an abstract *composition template* T_n consisting of the workflow logic and (b) a set of binding information containing each deployed service operation S_{T_n}.

The abstract template T_n contains invoke activities pointing to abstract web service definitions. While executing a services composition SC_n, the execution engine uses the binding information S_{T_n} to identify the actual concrete operation

to be invoked. Without a way to dynamically update the binding information, compositions are bound to use the same set of concrete operations, which results in great issues when such operations degrade their performance or present any fault. Thus, the binding information available for each deployed service composition is usually the main concern of adaptation frameworks.

This general approach of changing the binding information S_{T_n} for the service composition SC_n, however, is limited and presents some problems. First, it means that any change performed in the binding information S_{T_n} will have impact in all private sessions created for the service composition SC_n. In other words, micro adaptations focussed in particular sessions are not possible, reducing the reliability of the business process in certain circumstances. Second, because changes in the template itself are not covered, this approach only supports replacing the binding information of one concrete operation with another.

This means that it is not possible to use a group of candidate operation as replacement of a single deployed operation, or use a single candidate operation as replacement for a group of deployed operations. The common adaptation approaches focus only on faults enclosed to a service composition context. In other words, faults observed for a private session are not shared across service composition in an attempt to prevent potential faults and failures. Moreover, because the usual approach is concerned with the reaction to internal faults, other events of interest for the adaptation process, such as the appearance of better services or new requirements, are not covered.

In order to solve these issues, instead of limiting adaptations to the binding information of service compositions triggered by internal faults, ProAdapt uses independent adaptable service composition *execution instances* that may be up-

dated not only by monitoring internal faults but any event of interest, including faults and changes in the execution context or requirements. In addition, simmilarly to the work presented in [26], ProAdapt supports workflow evolution, accepting changes in the workflow for running instances. Differently from the proposal in [26], however, we are able to cope with exceptional or unplanned event that affects only a specific instance of a workflow, but our strategy is limited to finding a set of service operations that are semantically equivalent to logic regions defined in the workflow.

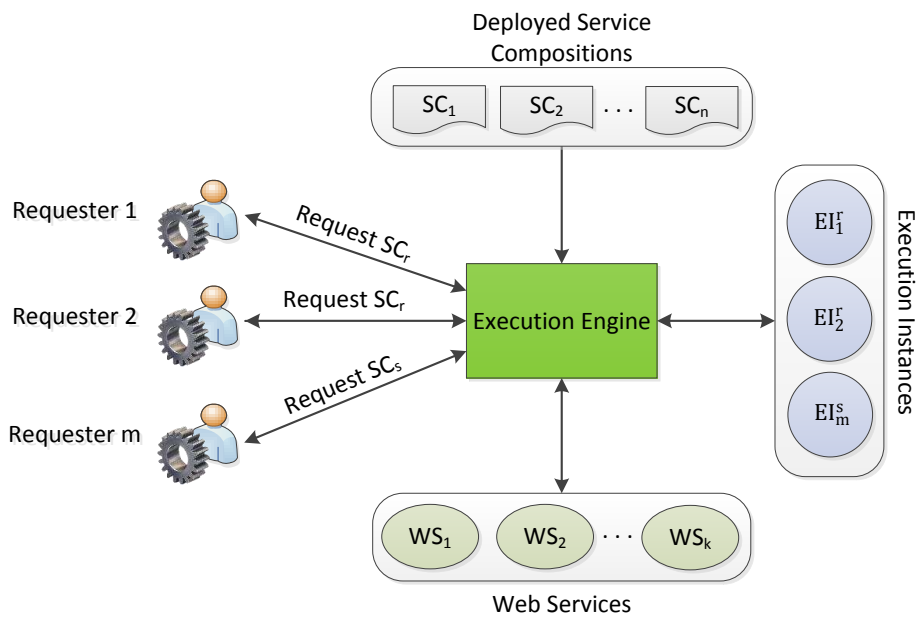

Figure 3.2: Illustration of the Execution Engine accessing Execution Instances of Service Composition.

Figure 3.2 illustrates the concept of independent execution instances used in ProAdapt. As show in the figure, for each request m of a deployed service composition SC_n, an execution instance EI_m^n is created.Then, the execution engine can invoke web services according to what is defined inside the execution instances.

Unlike current execution engines that rely on a general templae and binding information for each service composition, as described above, each of our execution instances EI_m^n includes a copy of the composition template T_n the binding information S_{T_n}. In this context, for reasons of performance and optimisation, a S_{T_n} is still maintained for T_n, but each execution instance EI_m^n contains its private template T_m^n and binding information $S_{T_m}^n$.

With this modification, adaptations enclosed to a particular instance are now possible, without losing support for general or parallel adaptation. For example, considering three execution instances (a) EI_1^r, (b) EI_2^r, and (c) EI_1^s for the service compositions SC_r and SC_s, changes in the local template for (a) T_1^r or its binding information $S_{T_1}^r$ have no impact in the local templates for (b) T_2^r and (c) T_1^s or the binding information $S_{T_2}^r$ and $S_{T_1}^s$.

It is possible, however, to flood the need for adaptation across parallel execution instances of the same or different service composition by accessing their particular template and binding information. Moreover, considering a set of deployed service compositions $\{SC_1, SC_2, ..., SC_n\}$, future execution instances can benefit from previous processed information by changing the default templates $T_x, 1 \leq x \leq n$ or the default binding information $S_{T_x}, 1 \leq x \leq n$, which are both used to create the execution instances.

Faults are recurrent events for service compositions, and as explained in Section 2.1, these faults may lead to dissatisfaction and can result in lost opportunities for business processes. Reacting to faults after they are observed incurs in obvious penalties, thus, preventing and avoiding faults is a very important feature for adaptable service-based systems. Perhaps the most important feature of ProAdapt is indeed the ability to *proactively identify potential fault points*

in execution instances based on the monitored events and prevent such faults to occur. Moreover, even in the absence of faults, ProAdapt may decide to proactively optimise service composition. Resulting in faster, cheaper or more reliable processes.

ProAdapt currently supports the *prediction of faults and failures caused by QoS deviation and unavailability of operations*. Service composition execution instances must comply with predefined SLA parameters, such as maximum response time or cost for the whole composition. QoS deviations of deployed operations, mainly in the form of delayed response time, can lead to nonconformance of the SLA.

In order to handle this issue, ProAdapt uses a QoS aggregation function that is able to predict the expected QoS parameter values for the whole execution instance and thus trigger adaptation when detected a possible SLA nonconformity and, therefore, trigger the need for adaptation when a possible SLA discrepancy (i.e., nonconformance) is detected. To prevent invoking unreachable operations (unavailable or broken network link), the framework supports the prediction of unreachable operations using failure spatial correlation[51].

In this work, the spatial correlation is considered as the strong availability correlation between operation, services and providers. In other words, if an operation is unavailable in one part of a service composition, it will most likely be unavailable for other parts of the composition that also use the same operation. Moreover, if the whole service or provider is unavailable, it would be wise to proactively replace all deployed operations offered by the unavailable service or provider to avoid the almost certain faults.

Regarding adaptations necessary due to unavoidable oscillation of the ob-

served QoS values of service operations, ProAdapt attempts to avoid executing changes in a composition in the situations in which QoS faults generated by one or more deployed operations can be *compensated* within the composition by other deployed service operations.

Example: when the observed response time of an operation is greater than its expected response time, the approach verifies the implication of this fault in the execution instance as a whole in order to assess the need for adaptation. This verification considers service level agreements (SLAs) specified for the whole composition and QoS parameter values obtained from previously invoked operations as well as expect values for the yet to be invoked operation in order to identify if a QoS fault can be compensated or accommodated. ProAdapt is ultimately concerned with the correct execution of the composition under defined constraints. Internal faults can be overlooked if the service level agreement (SLA) defined for the composition remains satisfied.

The next section presents the ProAdapt architecture and describes its main components.

3.2 Architecture

In the case of proactive approaches for dynamic service composition, situations that may trigger the need for adaptation should be predicted before they lead to faults and failures of the service compositions. As defined in [126], the prediction of faults is concerned with the identification of the occurrence of a fault in the near future based on the assessment of the current state of the system. In the scope of service compositions, the prediction of faults and failures is concerned

with the assessment of the impact of a service, or group of services, misbehaviour, performance degradation, or unavailability in the composition as a whole.

ProAdapt is a proactive adaptation framework consisting of four main steps, namely (i) events monitoring, (ii) analysis and prediction of the events, (iii) decision of actions to be taken due to the events, and (iv) execution of the actions. In order to support these four steps, an architecture was developed for the ProAdapt framework with six main components, namely: (i) *Composer*, (ii) *Execution Engine*, (iii) *Adaptor*, (iv) *Service Discovery*, and (v) *Event Analyser*. An overview of the architecture of the framework is shown in Figure 3.3.

All the components of the framework had to be implemented from scratch due to the lack of support from current technologies. In fact, we were able to extend an existing execution engine to suit our need but only for the first prototype of our framework (see Section 4.1).

In summary, a system administrator specifies some business process and related SLA to be deployed in the execution engine. When clients request these business process through the execution engine, it uses the composer to create an execution instance that is particular to each request. All events produced by the execution engine, such as successful or failed attempts to invoke an operation are reported and analysed by the event analyser. Depending on the context and generated events, the need for adaptation is identified and the adaptor component updates the running execution instances. Details regarding each of the main components of the framework are described below.

The *composer* is responsible for parsing business processes descriptions represented in different languages (e.g.; WS-BPEL [115]), and their associated Service Level Agreements (SLAs), in order to generate the associated *Execution*

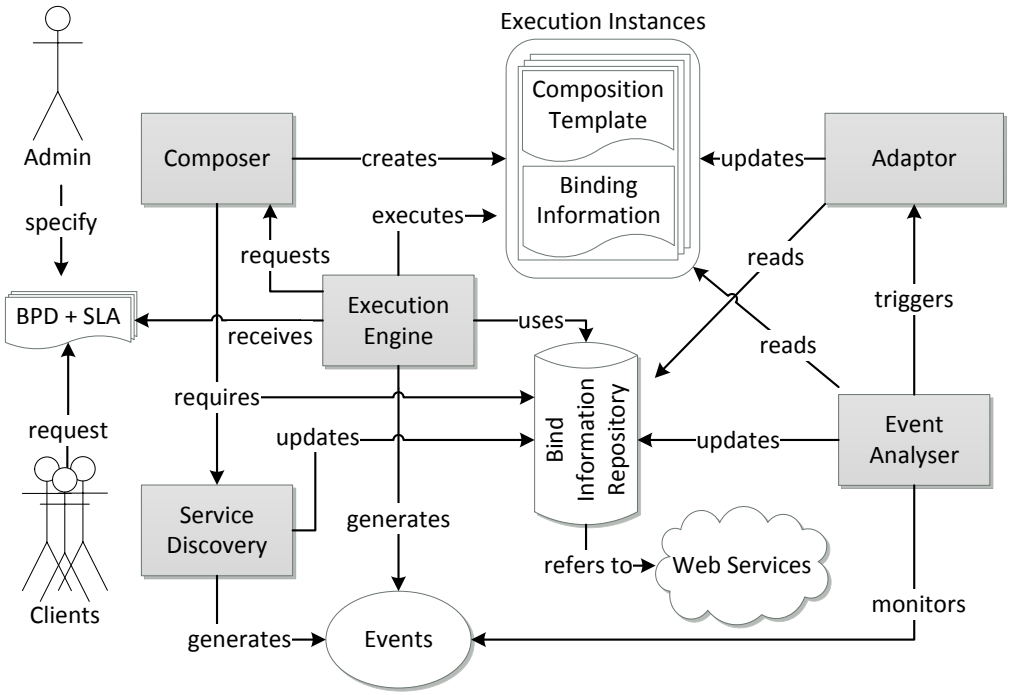

Figure 3.3: Architecture overview of ProAdapt framework

Instances. The current approach is limited to compositions that use stateless services; i.e.; compositions with service operations that can be replaced at any point during its execution since there is no need to keep track of the interaction-specific data as each subsequent interaction with a stateless service does not depend upon the outcome of the previous interaction.

As explained in Section 3.1, the framework considers SLA parameters for the whole composition in order to verify if QoS discrepancies require the need to execute changes in a composition. Therefore, the SLA parameter values will be represented in the internal representation of the business process.

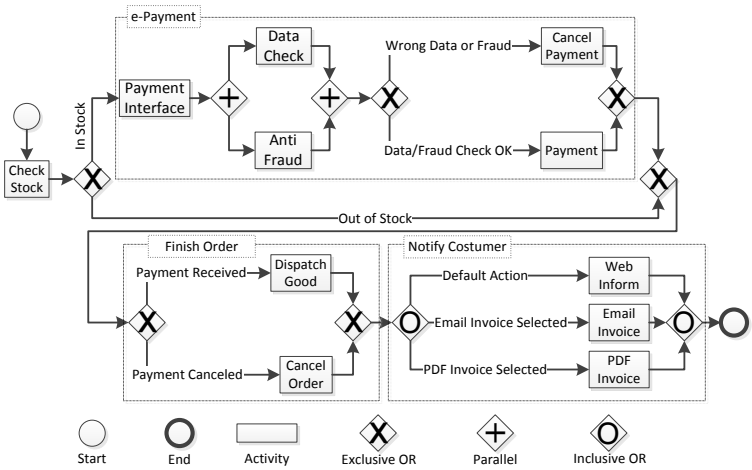

Figure 3.4: Example of a business process for ordering of goods

Figure 3.4 presents an example of a business process model using the BPMN notation for ordering goods online. In Figure 3.4, the required operations of the business process are represented as activities. As shown in the example, when a customer makes an order, the system checks the stock and in case of availability of

the required goods, the system displays the interface for executing the payment. In the case of a successful process execution, the payment is received and the goods are dispatched to the customer. Otherwise, both the payment and the order are cancelled. At the end of the business process execution, the customer receives the transaction status from a web interface. In the case of a successful execution, the customer receives an invoice by email or link to PDF file.

There are four different abstract activities or regions defined for Figure 3.4. The *Invoke Activity*, which is the final level of abstraction before the actual binded service operation, and three logic control regions, namely *Parallel*, *Exclusive OR*, and *Inclusive OR*. The parallel region is self explanatory, each sub-activity within the region will be executed in parallel. For instance, *Data Check* and *Anti Fraud*. The distinction between the *Exclusive OR* and the *Inclusive OR* is that the last admit more then one path to be executed. While *Dispatch Good* and *Cancel Order* will never be executed together, *Email Invoice* and *PDF Invoice* may or may not be executed together.

The *Execution Engine* is the component responsible for running the execution instances. The execution engine receives a request for a particular business process mapped to a service composition, requests an execution instance from the composer and uses the logic and binding information provided by the execution instance to manipulate input and output parameters.

As explained in the previous section, this work recognises that different private sessions may face distinct circumstances which may temporary and potentially affect only a single session. Moreover, the current state of each session is certainly different, and adaptation procedures may well take into account these different states.

In the framework, when a business process deployed in the execution engine is requested, the associated composition template and default binding information is used to generate an *execution model instance* of the business process. An execution model instance extends the composition template with information about the (i) execution flow, (ii) deployed endpoint service operations and their locations, (iii) state of a service operation in a composition (e.g., executed, to be executed, and executing), (iv) observed QoS values of a service operation after its execution, (v) expected QoS values of a service operation, and (vii) SLA parameter values for the service operations and the composition as a whole.

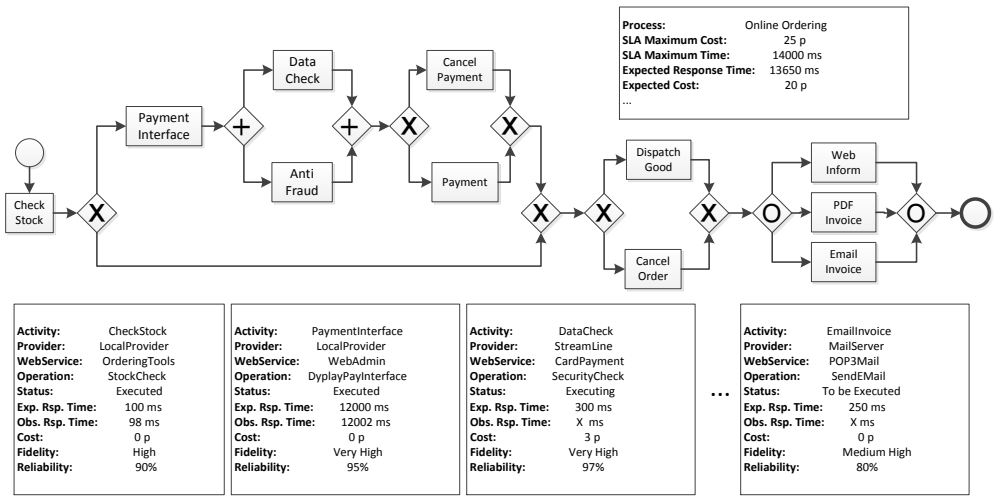

Figure 3.5: Execution model instance for the business process.

Figure 3.5 shows an example of part of an execution model instance for the business process presented in Figure 3.4. Figure 3.5, shows the deployed endpoint service operations for various activities, the QoS values for the whole execution model instance and for the deployed endpoint service operations, the execution

status of the service operations, and some expected and observed QoS values. As shown in Figure 3.5, the service operations executing the activity of *CheckStock* and *PaymentInterface* are offered by the same services provider *LocalProvider*. The *DataCheck* activity is offered by service provider *StreamLine*. The expected cost and response time values for the whole execution model instance are based on the deployed operation endpoints and their status.

The *monitor* verifies the service operations used in the instantiated execution models and the possible replacement candidates' operations in other to identify any events that may lead to situations C1, C2 and C4 described in Section 3.1. As shown in Figure 3.3, monitors can either intercept the messages exchanged between the *Execution Engine* component and the *Web Service Operations* or check the operations directly, for instance, using online testing. The current implementation of the framework focuses only on the first strategy, intercepting calls to the services, observing when new services become available, and checking the QoS values of the operations. It also communicates with the adaptor component to inform about observed events.

The *service discovery* component identifies possible candidate service operations to be used in the composition, or to be used as replacement operations in case of problems. The work assumes the use of the service discovery approach [130][146] which has been developed to assist with the identification of candidate service operations. This approach advocates a proactive selection of candidate service operations based on distance measurements that match functional, behavioural, quality, and contextual aspects. The candidate service operations are identified in parallel to the execution of the compositions based on subscribed operations, and are organised in descending order of best matches. The identi-

fied candidates' operations are used to create and adapt execution models by the adaptor component.

The *adaptor* together with the *Event Analyser* are the main components of the framework. Together, they (a) receive events from the execution engine and monitor components (situations C1, C2, and C4), and composer component (situation C3); (b) predict and analyse problems that may exist in the composition based on these events; (c) identify the need for adaptation of the current execution model instances and/or templates; (d) decide on the actions to be taken; (e) makes necessary changes in the execution model instances and/or templates; and (f) informs the execution engine about changes made in the execution model instances.

The decision of actions to be taken and changes in the compositions are made based on the identified events and the list of available replacement candidate service operations identified by the service discovery component. The next section covers the adaptor component in more details.

It is important to note that in out current framework, the monitor is seen as a sub-component of the event analyser, however, an external monitor could be attached to the framework and generate events in parallel, such as new status of service operations.

3.3 Adaptation Process

In the framework, the different steps for proactive adaptation of service composition are supported by the use of techniques for the prediction of QoS aspects; the analysis of dependencies between service operations in a composition; and

the consideration of groups of service operations in a composition flow instead of isolated operations.

As stated in Section 3.1, the adaptation process may be triggered by situations of types C1 to C4. When an event from one of classes C1 to C4 occurs, the monitor, the execution engine, or the composer components notify the adaptor. The adaptor analyses the event, predicts faults and failures, verifies the need for adaptation, and enforces the necessary changes.

The analysis of events is concerned with the implications of such events in the deployed service compositions, mainly the need for adaptation of current running and possible future execution instances. As discussed in Section 3.2, for each request m of a deployed service composition SC_n, an execution instance EI_m^n is created from the respective composition template T_n using an available list of candidates' operations. This process is optimised by maintaining a default combination of candidates' operations S_{T_j} for each deployed composition template and customising it per execution instance. An important concept of the analysis phase is the scope or impact of events analysed.

Section 3.2 explains how execution instances are independently adaptable, without interfering with other instances. This does not mean, however, that the analysis of an event is enclosed to a particular instance. The possibility of constraining the required measures to a single instance is a feature of the framework, not a mandatory employed solution. In other words, any event of interest may potentially trigger the process of adaptation over several instances in parallel.

Another important task of the analysis phase is to update the execution instance with the monitored QoS values. An execution model instance can be up-

dated in several ways during its execution by: (a) changing the status of its operations (e.g., from "to be executed" to "executing" and "executed"), (b) changing observed and expected QoS values of the operations, (c) changing the operation endpoints, and (d) changing the composition workflow. The execution engine and analysis step are responsible for points (a) and (b) cited above, while points (c) and (d) are the results of actions enacted by the adaptor component.

Considering that the analysis of the events indicates the need for adaptation, the phase of decision and execution of actions uses virtually the same process, regardless of the class of situation observed or the execution instance status. This is because the decision and execution of actions are performed using the same algorithm.

The algorithm takes into consideration all information kept by the execution instances along with the set of available candidates' operations. More precisely, all required information to adapt the execution instance, such as expected QoS parameter values, predicted failures, and dependencies between operations, can be obtained reading only the execution instance and the set of candidate service operations.

The dynamic service composition algorithm is responsible for reorganising the composition workflow by replacing service operations in order to meet the defined requirements. The approach supports four types of possible replacements, namely (a) one-to-one, (b) one-to-many, (c) many-to-one, and (d) many-to-many.

In other words, adaptations of the various execution instances can be performed by changing a single deployed service operation by a candidate operation (a), or a group of candidates' operations (b). It is also possible to replace a group of deployed operations by a single candidate operation (c) or a group of can-

didates' operations (d). The method also supports the verification of signature dependencies between service operations in order to avoid inconsistent compositions that cannot be executed.

We define that an operation O_2 has a signature dependency with an operation O_1 (i.e., O_2 depends-by-signature on O_1), when the output parameter (or its part) of O_1 is used as input parameter (or its part) in O_2.

Example: consider operations *PaymentInterface():Info* and *DataCheck (customer:String, cardNumber:String):Boolean* in Figure 3.4. Assume *Info* a complex data type with parameters *time:Time, local:Coordinate, name:String, cardNumber:String*. In this case, operation *DataCheck()* depends on operation *PaymentInterface()*, since *DataCheck()* uses part of the output parameter of *PaymentInterface()* as one of its input parameters.

The solution employed in this work for the case of signature dependencies is to replace the operations with candidates' operations that would satisfy the signature dependency, if they exist. This process of solving the signature dependency may be recursive. This is because candidates' operations chosen to solve a signature dependency in a particular point of the execution model may cause signature dependency issues in other parts of the composition.

3.3.1 Analysis of Events

As aforementioned, one of the key concepts built in the ProAdapt framework is the ability to monitor the execution context and predict faults in order to prevent composition failures, such as inability to complete the execution and nonconfor-

mity with SLA values. The framework currently supports two techniques for failure prediction.

One of the failure prediction techniques is concerned with the impact of QoS faults of internal components in the composition as a whole. More precisely, this technique identifies QoS deviations of operations used in the execution instance of a service composition and determines the need for adaptation based on impact of such deviation on the composition as a whole.

Currently, the framework supports the prediction of the nonconformity with SLA values for the whole composition using operations response time and cost values. The second failure prediction technique supported by the framework is concerned with the prediction of unreachable operations, services, and providers. Moreover, the framework supports the analysis of events and trigger of the need for adaptation in parallel with all execution instances.

The next sections provide details of the analysis performed with QoS events in order to prevent a nonconformity with the SLA and the analysis of events observed from the communication protocols to predict unreachable operations and avoid composition failures, and presents the benefits of the parallel adaptation.

3.3.1.1 QoS Analysis

In recent years we have experienced an increase in the amount of available web services with diverse features. Services that offer the same functionality are normally chosen based on their different quality of service (QoS) parameters. However, some of these parameters change constantly due to situations that are beyond the control of both the requester and the provider of the service, such as the availability and response time.

An important research problem for adaptable service-based system is concerned with the handling of these oscillations of QoS parameters in order to maintain the service level agreement defined in the service composition. In service oriented computing, the ability to perform adaptation by considering the various QoS values is usually known as QoS-aware dynamic service composition[109][31].

From the various existing QoS values defined for service operations, the response time is arguably one of the most important; at the same time it is perhaps the best example of a parameter that constantly changes its values over time and between calls. This is due to two primary reasons. First, the performance in terms of response time of a service operation varies a bit according to the load and computational resources available for the operation to execute. Second, web services are provided over a best effort network communication channel (Internet). In other words, there are certain circumstances that may cause small or even severe delays in service operations, directly impacting the service compositions that are invoking such operations.

The work in this thesis concentrates on the analysis and prediction of the response time of service operations as key QoS parameters, in order to meet SLA values defined for service compositions. In addition, our framework identifies the relationship that exists between the cost to use service operations and their performance in terms of the response time. In summary, ProAdapt supports the analysis of two strongly connected QoS parameters: response time and cost.

In order to guarantee compliance of the SLA response time and cost values in a service composition execution instance, it is necessary to consider (i) the aggregation of the response time values of the participating operations in a composition; (ii) the various processing times of the execution engine that encompass

activities such as marshalling and unmarshalling of messages, multiplexing and demultiplexing of the communication channel, and process scheduling (see Sections 1.1 and 1.2); and (iii) the time for the adaptor to identify and analyse events of interest and perform changes in the execution model when necessary.

Given a service composition SC_n and the respective composition template T_n, the response time for a related execution instance EI_m^n is given by the function below.

$$T(n,m) = \bigsqcup(n,m) + E(n) + A(n,m) \tag{3.1}$$

where:

- $T(n,m)$ is the time to complete the execution instance m from the service composition n;
- $\bigsqcup(n,m)$ is a function that returns the aggregated values of the observed and expected response time of each service operations $O_1, O_2, ...O_r$ deployed in EI_m^n;
- $E(n)$ is the expected overhead time of the execution engine for the composition template T_n;
- $A(n,m)$ is the time required by the adaptor for the execution instance m of the service composition n.

The aggregated response time $\bigsqcup(n,m)$ can be seen as the expected minimal time to complete an execution instance. Such minimal time could only be achieved if we did not have to incur in overhead from the execution engine and no

adaptation process would take place. Given that it is necessary to have time to orchestrate and pass messages around $E(n)$, or to adapt the execution instance when required $A(n, m)$, the aggregated response time cannot be used alone. While the adaptation process for a single execution instance is performed in parallel to the execution or adaptations of other execution instances, it creates a single restriction for the instance that is being adapted. The operations not yet executed cannot start. Thus, the need to include the adaptation time in order to compute the expected response time for the service composition instance as a whole.

From the Equation (3.1) above, $E(n)$ is the only element that does not depend on the execution instance itself. In practice, there may be some small variation in the expected overhead time of the execution engine for two execution instances of the same composition, but our experiments show that such time is negligible or difficult to predict. In case where there is no deviation from the expected QoS values, there is no need for adaptation, thus, the expected overhead time of the engine can be computed using the Equation (3.2) below.

$$EPC(n, m) = T(n, m) - \bigsqcup(n, m) \tag{3.2}$$

As shown in Equation (3.2), the expected overhead time of the execution engine is the observed completion time of the execution instance T(n,m), minus the aggregated function of the observed response times of the operations for the finished execution instance EI_m^n.

Since this time may vary depending upon available resources and number of parallel execution instances, ProAdapt continuously update the value with each new completed execution instance. Such value is given by the Equation (3.3)

below.

$$E_x(n) = \begin{cases} EPC(n,m), \text{if } x = 0 \\ \frac{E_{x-1}(n)+EPC(n,m)}{2}, \text{if } x > 0 \end{cases} \tag{3.3}$$

where, E_x is xth computation of the expected overhead time, $A(n,m) = 0$ and EI_m^n has finished its execution.

In other words, the expected overhead time is computed as the exponentially weighted moving average based on the difference between the minimal time to complete an execution instance $\lfloor\rfloor(n,m)$, and the observed final execution time after the completion of the execution instance.

From the three parameters of the Equation (3.1) above, the adaptation time $A(n,m)$ is the only one that may not impact the final calculation, that is, no adaptation was required. When changes are required and enacted, however, the adaptation time is then logged and the expected response time for the whole execution instance updated. This means that if another adaptation is required, this previous logged adaptation time will be considered when attempting to find a valid configuration to guarantee compliance with the SLA.

The aggregation of the response time values of the service operations in the execution model considers different execution logics in a model such as *sequence*, *parallel*, *conditional selection*, and *repeat logics* [128]. In the example in Figure 3.4, *DataCheck* and *AntiFraud* represent a parallel execution logic activity, while *DispatchGood* and *CancelOrder* are conditional activities.

A simplified view of the logic regions within an execution context, as well as the logic template of the execution instance can be obtained recursively as show

in Equation 3.4.

$$LR = \begin{cases} IO, \\ SEQ(LR, LR, ..., LR), \\ PAR(LR, LR, ..., LR), \\ XOR(LR, LR, ..., LR, BE), \\ IOR(LR, LR, ..., LR, BE), \\ RPT(LR, BE) \end{cases} \tag{3.4}$$

In Equation 3.4, a logic region (LR) is recursively defined as (a) a single invoke operation (IO); (b) a set of logic regions in sequence (SEQ); (c) a set of logic regions in parallel; (d) a set of logic regions in a conditional exclusive or (XOR); (e) a set of logic regions in a conditional inclusive or (IOR); (f) a repeatable logic region (RPT); those last three controlled by some Boolean expression (BE). The logic regions b-f are known as control logic regions or control activities, since they are responsible for the work flow of the business process.

In order to better explain the concepts presented in Equation 3.4, the steps below presents a top down approach to identify these regions in the case of the Goods Online Process defined depicted in Figure 3.4 (with Boolean expressions hidden for simplicity).

1. $LR_1 = SEQ(IO_1, LR_2, LR_3, LR_4$, where LR_1 represent the top level region, which is formed by a sequence of an invoke operation ($IO_1 =$ *Check Stock*), and three other logic regions LR_2, LR_3, LR_3;

2. $LR_2 = XOR(IO_2, LR_5, LR_6)$, where LR_2 is the *e-Payment* exclusive con-

ditional logic region. IO_2 is the invoke operation *Payment Interface*, and LR_4 and LR_5 two other logic regions;

3. $LR_5 = PAR(IO_3, IO_4)$, where IO_3 and IO_4 are *Data Check* and *Anti Fraude* respectivaly;

4. $LR_6 = COND(IO_5, IO_6)$, where IO_5 and IO_6 are *Cancel Payment* and *Payment* respectively;

5. $LR_3 = XOR(IO_7, IO_8)$, where LR_3 is the *Finish Order* exclusive conditional logic region. IO_7 and IO_8 are *Dispatch Good* and *Cancel Order* respectively;

6. $LR_4 = IOR(IO_9, IO_10, IO_11)$, where LR_4 is the *Notify Costumer* inclusive conditional logic region. IO_9, IO_10, and IO_11 are *Web Inform*, *Email Invoice*, and *PDF Invoice* respectively.

Figure 3.6 below presents graphical representation of the steps described above. As shown in Figure 3.6, there are 11 operations logically organised between five control logic regions of four different types (parallel logic, inclusive conditional, exclusive conditional, and sequential logic).

Since we assume that every logic region ultimately culminates to an invoke operation, the process of computing the aggregated response time $\bigsqcup(n, m)$ for a EI_m^n can thus be broken down to computing the expected response time of the concrete service operation deployed for the particular invoke operation region, and recursively compute the expected response time of the ascendant regions.

Example: in the case of the *Goods Online Process*, the expected response

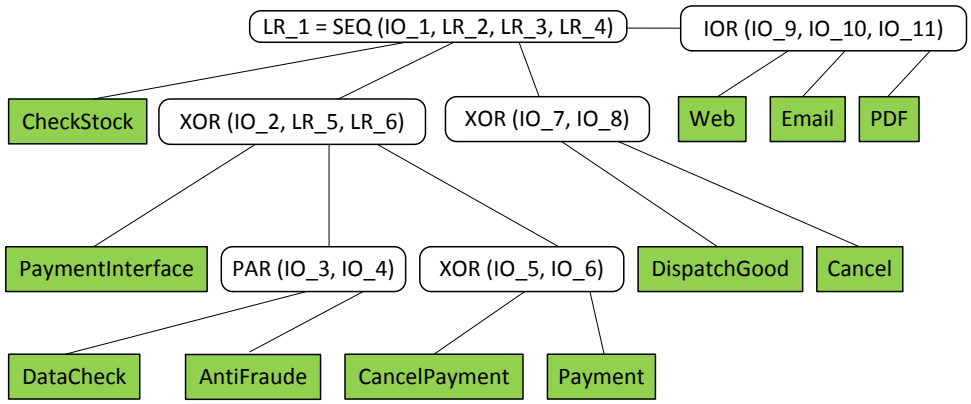

Figure 3.6: Simplefied view of the regions breakdown structure of the Goods Online Process

time of both *CancelOrder* and *Dispatch Good* would be calculated, prior to computing the expected time of the exclusive conditional logic region *Finish Order*.

We can define the region response time as:

$$RRT(LR) = \begin{cases} IO_{RT}(IO_n), LR \rightarrow \text{Invoke Operation} \\ SEQ_{RT}(LR_1, LR_2, ..., LR_n), LR \rightarrow \text{Sequence} \\ PAR_{RT}(LR_1, LR_2, ..., LR_n), LR \rightarrow \text{Parallel} \\ XOR_{RT,BE}(LR_1, LR_2, ..., LR_n), LR \rightarrow \text{Exclusive OR} \\ IOR_{RT,BE}(LR_1, LR_2, ..., LR_n), LR \rightarrow \text{Inclusive OR} \\ RPT_{RT,BE}(LR_1), LR \rightarrow \text{Repeat} \end{cases} \tag{3.5}$$

where RRT(LR) is the generic function to compute the region response time of the logic region LR, and $IO_{RT}, SEQ_{RT}, PAR_{RT}, XOR_{RT,BE}, IOR_{RT,BE}, RPT_{RT,BE}$

are the concrete functions to compute, respectively, the expected response time of *invoke operations*, *sequential control logics*, *parallel control logics*, *exclusive conditional control logics*, *inclusive conditional control logics*, and *repeatable control logic*. While the invoke, sequence, and parallel activities or logic regions are self contained, the exclusive and inclusive conditional activities as well as the repeat control logic require a Boolean expression (BE), as expected.

Table 3.1 presents the summary of how to compute the aggregated response time for each of these logic regions presented above.

Table 3.1: Summary of the aggregated response time for logic regions.

Logic Region	Aggregated Time
SEQ	$\sum_{i=1}^{n} RRT(LR_i)$
PAR	$\max(\{RRT(LR_i) : i = 1, \ldots, n\})$
XOR	$\max(\{RRT(LR_i) : i = 1, \ldots, n\})$
IOR	$\max(\{RRT(LR_i) : i = 1, \ldots, n\})$
RPT	$k * RRT(LR_1)$

In the case of a sequence of logic regions such as $SEQ(LR_1, LR_2, ..., LR_n)$, the aggregated response time is calculated as the sum of the response times of the logic regions in the sequence. Likewise, the cost is computed as the sum of the aggregated costs of the logic regions in the sequence. Equation(3.6) formalises this definition for the response time.

$$SEQ_{RT} = \sum_{i=1}^{n} RRT(LR_i) \tag{3.6}$$

In the case of parallel and conditional logic regions, the aggregated response time is calculated as the maximum expected response time of the descendant logic regions, as defined in the Equation (3.7) below. The use of the maximum

function for conditional regions could be seen as a pessimist approach, however that is not the case. The composition needs to be reliable and thus, even in the event of the most time consuming path, complete its execution while meeting the SLA. The cost for the conditional logic is calculated similarly, but in the case of parallel logic, the cost is computed as the sum of the aggregated costs of the logic regions.

$$PAR_{RT} = XOR_{RT} = IOR_{RT} = \max(\{RRT(LR_i) : i = 1, \ldots, n\}) \tag{3.7}$$

The repeatable logic control region is the trickiest one. In order to compute the expected response time, one solution is to start with one execution of the logic region, and then, adjust the factor as more execution instances are finished in order to best estimate the average number of times a repeatable region is executed (k in Table 3.1. It could also be possible to annotate the activity to provide information regarding the expected number of execution, in which case the solution would converge faster.

In the case of endless loop a different approach must be made. In this case the process clearly does not completely terminate and we cannot expect to include all executions of the loop in the general SLA expectation for the whole composition. In this case a design decision must be made. For instance, if one knows a priory that a endless loops is going to be used, such activity can be designed to be included in the QoS considerations just once.

Finally, the aggregated response times and costs of the operations in a service composition instance are calculated based on expected values of the operations

not yet executed, and the observed values of the operations already executed. In other words, $IO_{RT}(IO_n)$ is either the observed response time collected for an already executed operation, or an expected value of the response time of the related operation in case it is yet to be executed or is still executing.

In the case of the expected cost of the regions, the aggregation process can be done by using the values informed by the service providers only. This is due to the fact that the cost of a concrete operation does not tend to change over time frequently, and it is fair to assume that changes must be informed by the service provider. Even then, the aggregation of the cost must be performed to check if the execution instance complies with the SLA. Any replacement in the execution instance, however, triggers another verification, in order to ensure that the change still complies with the SLA when considering the execution instance as a whole.

When it comes to the response time, however, things are not static. As outlined in [40], the response time for an operation request combines the time for executing the operation and the network time for sending the operation request and receiving its response. During the experiments conducted in this thesis, it was observed that there are also other times that should be considered such as the time of marshalling/unmarshalling a request, and the time that a request may need to stay in a queue in both client and server sides. The response time of an operation can be defined as the Equation (3.8):

$$RT(O_n) = PT(O_n) + DT(O_n) \tag{3.8}$$

where:

- $PT(0)$ is the processing time for an operation O, which is given by service providers;
- $DT(0)$ is the deliver time which is the time to send a request and receive a response after it been computed.

As shown above, the response time is considered a variable parameter that can be affected due to changes in the network and system resources. In fact, both processing and deliver times may suffer variations as described above. These variations or their causes, however, do not need to be separately estimated for the needs of this work, only the response time as a whole. Therefore, in order to identify expected response time values, it is necessary to use techniques that predict or approximate the behaviour of a function with random parameters. The exponentially weighted moving average (EWMA) [108] is the technique employed for this prediction since ProAdapt does extensive use of this prediction technique in order to predict and prevent failures, and due to the fact that the expected response time value of an operation changes greatly, but relatively smoothly over time. It is relatively simple, does not consume too much memory or processing powers, and is very fast to compute.

Given that OB_t^n is the observable response time of the operation O_n in the time t, the exponentially weighted moving average of the response time is given by the Equation (3.9) below:

$$\begin{aligned} EWMA_0^n &= PT(O_n) \\ EWMA_t^n &= \alpha * OB_t^n + (1-\alpha) * EWMA_{t-1}^n \end{aligned} \tag{3.9}$$

where:

- $EWMA_t^n$ is the expected response time value of operation O_n at time t;
- the coefficient α is a constant between 0 and 1 that controls the degree of importance of old observations. A lower α discounts older observations slower.

As shown in Equation (3.9), the initial EWMA value is started as the processing time informed by the service provider. Such value is then updated with each execution of the operation. As a second degree of cautiousness, ProAdapt analyses the standard deviation of the observed values to create a boundary of the expected result, as presented in Equation (3.10).

$$B_t^n = EWMA_t^n + \beta * SD_{t,x}^n \tag{3.10}$$

where B_t^n is the boundary value of the expected response time of operation O_n at time t, β is as a constant parameter, and $SD_{t,x}^n$ is the standard deviation of the observed response time series of O_n at time t using x previous observations.

3.3.1.2 Availability Analysis

When the *execution engine* invokes any service operation defined in an execution instance, there are three possible outcomes, namely (a) successful, (b) delayed, and (c) faulty. In the case of a successful invocation (a), the execution instance is updated with the observed QoS values and no additional procedure is required.

Due to various reasons, such as congested network link or degraded performance, the expected reply from service operations may be delayed. An operation

invocation is said to be delayed (b) if the reply message is not received within the expected response time for the operation as defined in the previous Section 3.3.1.1.

Finally, when an operation is temporarily unreachable, attempts to invoke the operation will generate a faulty outcome (c). This occurs if (1) the expected reply message is lost, (2) the reply is discarded because it does not arrive before a maximum timeout set for the operation, or (3) an error message is received instead of the expected reply.

For all the cases (1-3) classified as faulty outcome (c), the *execution engine* must try to execute the invoke activity again, either using the same deployed operation or another candidate supplied by the adaptor. Invoking operations that are out of reach can cause a lot of problems for business processes. Predicting the unavailability of these operations in order to prevent fault invocations can save time and business opportunities.

The technique for failure prediction supported by the framework aims at proactively detect invoke activities which most likely will generate a faulty outcome, and thus, provide alternatives to replace the potential faulty operations.

ProAdapt employs a method to predict availability of service operations based on spatial correlations between operations, services, and providers. The idea is to proactively detect invoke activities which most likely will generate a faulty outcome and use the adaptation process to find alternatives for these operations.

More precisely, the technique uses events generated in case of a faulty outcome, such as error messages, and the strong relationship of availability between operations of the same service or provider, in order to identify other operations deployed in the execution instance with a great likelihood of generating a fault.

Example: when an operation O becomes unavailable, the process looks for

other references to O in the logic flow of the execution instance and mark them all to be replaced. This operation may be only temporarily unavailable or faulty within a service S, which means that other operations serviced by S can still be invoked. In the case such service S becomes unavailable, the process marks all other operations of S been used in the execution instance to be replaced, since these operations cannot be reached if the entire service is unavailable.

In the case above, the other services of the same provider may still be reachable, thus no action is performed to prevent the access to them. However, when a provider P is unavailable, the process considers all services and operations deployed in the execution instance that are provided by P to be replaced. These different situations are identified according to internal and external messages exchanged while attempting to invoke a service operation.

As the example above may suggest, the framework defines three levels for detection of spatial correlation, namely operations, services, and providers. The levels are described in more details below.

Operation Level - The same service operation O may be used in different points of a service composition. The spatial correlation may be used in execution instances that have such recurrent operations to prevent the call of future occurrences of a given operation if it has already been identified as faulty at any point of the execution instance. More precisely, after an operation O has been identified as unreachable, for each invoke activity that uses O, a new candidate operation must be identified. This event is basically identified by a message (e.g. SOAP Fault Element) sent from the web service indicating that the required operation is unavailable. In addition, *monitors* can constantly update the new status of a service operation in parallel, making it available for running execution instances

or new execution instances. This is expected since the information regarding the status of service operations is kept in a centralised way and can be updated independently.

Service Level - When it is possible to identify that a faulty outcome was caused by an unavailable service, it is wise to mark to be replaced all operations in the composition offered by the service identified as unavailable. In this case, the service provider itself can inform that the required service is not reachable by replying with error messages (e.g. HTTP Error 404). The spatial correlation is direct since it is not possible to reach an operation of an unavailable service. The existence of operations offered by the same service in a composition is common, as is the temporary unavailability of services, making the spatial correlation at service level very useful. The unavailability of a service S reachable by the provider P is detected by error messages produced by P informing that S is not deployed or available in P.

Provider Level - When a service provider is not available, depending on the configuration of the machine or cluster in which the provider was available, different error messages may be observed, such as *connection exception*. In other words, even when the provider is not available, there are still messages to be observed in order to detect such situation and differentiate from the other levels. When an unreachable provider is detected, the prediction technique operates in a similar way as in the levels presented before. More precisely, for a provider P identified as unreachable, the process identifies all operations provided by P and mark them to be replaced by the adaptor process.

Both techniques for prediction and aggregation of QoS parameters described in Section 3.3.1.1 and the technique for predicting the availability of service oper-

ations described above can be seen as preprocessing activities for the adaptation process. The techniques may change the status of service operations within or across multiple execution instances and trigger the need for adaptation, but the actual decision and reconfiguration are performed by the adaptation process described in Section 3.3.2.1.

3.3.1.3 Parallel Analysis

As described in Section 3.1, ProAdapt was designed to analyse events and trigger adaptation for one or multiple execution instances. The adaptation of multiple execution instances for events observed for a single execution instance is possible because of two techniques used in ProAdapt.

(1) First, before invoking any service operation, the execution instance requests the status of such operation and in case it is out of reach an adaptation is requested. By carefully examining the two previous sections and the preprocessing phase, it becomes clear that even when for a single and independent execution instance an operation is marked as potentially unavailable, or updated with a severe degraded response time, these actions are reflected across all execution instances that uses the same operation. The decisions taken for each execution instance, however, will depend on the state of execution of each one of them. In such a manner, ProAdapt is proactively avoiding the attempt to invoke unreachable operations and thus saving time and computational resources.

(2) The second technique can be viewed as an extension of the first one. It consists of a parallel analysis of the impact of observed events in an execution instance, in other execution instances that are bound by the same events (e..g, use the same unavailable operation). This parallel approach provides two outcomes,

namely (a) parallel triggering of adaptation, and (b) load balancing.

The parallel triggering of adaption, as the name suggests, is a mechanism to flood the adaptation need across multiple execution instances in parallel. This approach is different from the status checking presented above because an adaptation may be triggered before reaching the execution point where the potential fault would be observed.

In a first analysis, the advantage of this parallel method may be difficult to see, since the faulty operations can already be avoided by using the proactive status checking together with the QoS and availability analysis presented in the previous sections. The time in which the adaptation is triggered, however, may be different using the parallel approach.

More precisely, the standard proactive adaptation may have to wait until the point in the workflow where an unavailable or degraded operation would be invoked in order to replace it and reconfigure the composition. On the other hand, if the need for adaptation is triggered in parallel, an execution instance may reconfigure itself early in the process, which potentially means more adaptation options and high probability of success in adapting under given constraints.

As defined in the previous chapter, giving a service composition S_k with a set of abstract operations $L_k = \{A_1, A_2, ...A_n\}$ and for each A_x a set of candidate concrete operations $R_x = \{C_x^1, C_x^2, ..., C_x^m\}$; A valid configuration $G = \{C_1^a, C_2^b, ..., C_n^r\}$ is the one that satisfies defined the SLA and the number of possible configurations for a composition is given by Equation 2.1.

Considering V_t the set of size t of valid configurations at a given time, the question is whether the parallel approach is able to improve the probability of finding a configuration belonging to V_t.

Equation 2.1 considers that all abstract operations are free to be composed, however, the execution instance status changes over time and some operations are blocked. In other words, the operations that are executing or already finished the execution cannot be changed.

If we consider T_k^s the set of all possible combinations for the service composition S_k when s operations are free, the size of T_k^s equals to $NC(S_k)$ (see Equation 2.1) when $s = n$, that is, when all activities are free. If the candidate sets $R_1, R_2, ...R_n$ have roughly the same size, the sizes of the sets $T_k^s, T_k^{s-1}, ...T_k^1$ decreases exponentially.

$$P(x > 0)|x = |V_t \cap T^s| \tag{3.11}$$

The probability of successful adaptation is related to the probability of having at least one element in the intersection between the set of valid configurations (V_t) and the set of possible configurations given the free operations (T^s). In other words, if x is the size of the intersection set between V_t and T^s, the probability of successful adaptation is the probability of x been greater then 0 (see Equation 3.11).

The set of valid configurations is the result of the execution context and candidates' operations. However, the parallel approach allows early adaptation, increasing the number of free operations, and thus augmenting the probability of finding an element in the aforementioned intersection.

3.3.1.4 Load Balancing Analysis

When performing changes in a service composition, the load of a particular invoke activity can be distributed over different candidate service operations in order to increase or maintain the total throughput of the composition. If a deployed service operation is unavailable, and there are no candidates with the same expected throughput, a combination of more than one candidate operation can be considered.

In ProAdapt a load balancing technique is used to verify if the throughput of the service compositions can be maintained as initially specified for the compositions. The throughput specified for a service composition is reflected in the activities and their deployed operations in the composition. The throughput of each service operation in an execution instance is calculated and compared with the maximum accepted throughput value, in order to avoid overloading the use of the deployed operations.

This is done by using a throughput counter for each deployed operation. When an execution instance is created, the counters associated with the operations are incremented; when the operations are invoked during the execution of an instance, their associated counters are decremented. The maximum accepted throughput value of an operation is maintained in the bind information repository to allow the composer and adaptor components know which operations can be used in an execution instance, without causing operation to be overloaded.

Considering a service composition reduced to a sequence of logic regions $S_u = \{R_1, R_2, ..., R_n\}$, T_x the value of the throughput of the operation in R_x with the lowest throughput, and E_x the number of execution instances executing

operations within R_x. Equation 3.12 shows the theoretical maximum number of concurrent execution instances M_u for a service composition reduced to a sequence of logic regions S_u.

$$M_u = \sum_{i=1}^{n} T_i \tag{3.12}$$

Equation 3.12, however, cannot be used to estimate the total number of concurrent execution instances regardless of the point of execution. In other words, the execution instances must be distributed such that the number of execution instances executing operations within R_x must be not more than the value of the throughput of the operation in R_x with the lowest throughput ($E_x \leq T_x$).

In fact, the achievement of the maximum theoretical throughput of the composition depends on the ration of incoming request. If the average response times of the operations across logic regions are roughly the same, and new compositions' request are uniformly distributed, we can assume a close to even distribution of the number of execution instances executing operations within R_x. Equation 3.13 summarizes the new maximum value of concurrent instances under this scenario.

$$M_u = T * n \tag{3.13}$$

When ProAdapt verifies and increments the counters of the deployed operations during the creation of the execution instances and decrements the counters after the operations are executed, it is actually using a safe approach to distribute the load between operations, similar to the dedicated communication channel achieved by circuit switching for ATM networks [50]. The drawback is that it limits the amount of concurrent execution instances without actually exploiting

the maximum throughput of the deployed operations. An heuristic improvement is to consider not T_x when creating instances, but $n * T_x$, where n is the number of sequential logic regions.

The counters registering the use of service operations are used both to (a) avoid faults by checking the load before invoking an operation, and (b) proactively create a plan of execution in order to avoid the need for adaptation.

After all the preprocessing performed during the analysis phase, the next step is to take the decisions regarding the actions to be executed in order to accommodate the required changes. The next section describes the Decision and Execution of Actions phase of the adaptation process.

3.3.2 Decision and Execution of Actions

For any of the cases C1-C4 described in Section 3.1 there may be a need of replacing one or more service operations in the composition template or execution instances. This section describes the various steps in the adaptation process when a service operation needs to be replaced. The decisions and actions performed by the adaptor are concerned with the assessment of whether a new set of operation endpoints need to be identified, selected, and used to change the composition.

In order to follow the steps in the adaptation process, consider O a service operation in the execution model instance that needs to be replaced. This work assumes that a candidate service operation for O is an operation, or a group of dynamically composed operations, identified by the service discovery component (see Figure 3.3) that can provide the same functionality of O. For instance, in the example in Figure 3.5, a candidate replacement operation for *StockCheck()*

is an operation that verifies the existence of certain goods in stock, even if this operation has different signatures, quality or contextual characteristics.

It is also assumed that a candidate operation O' fully matches an operation O in the execution instance if O' and O have the same structural (signature) and the QoS values of O' do not violate the SLA values for the whole composition (functional match is already a criteria for an operation to be a candidate operation).

In ProAdapt, when selecting a candidate replacement operation, the process gives preference for matches in the operation signatures, instead of QoS values that do not violate the SLA values of the composition. This is because it is more cumbersome to make changes in the execution instance when there are other operations in the model that depend on the signature of the one to be replaced. More precisely, the matching of operation signature is a pre-requisite for the matching of QoS values. Mismatches of operation signatures require the use of either adapters or the replacement of conflicting operations. Moreover, differences in QoS values may be compensated by other operations in the model to be executed, or by trying to identify other operations that could compensate the QoS aspects. Furthermore, differences in QoS do not cause the model to stop its execution.

The process considers the various execution logics (regions) in a model such as sequence, parallel, conditional selection, and repeat logics. The execution logics are used (i) to calculate the aggregation of expected QoS values of operations in the model, and (ii) to identify a group of operations in the model that may need to be replaced by an operation or a group of dynamically composed operations. During the adaptation process, the sizes of the considered execution logics are

incrementally increased if necessary. When a replacement solution cannot be found for one execution logic E_l, the process considers the next larger execution logic that contains E_l.

Example: in Figure 3.4, if a solution cannot be found for the operations for activity *PaymentInterface*, the next execution logic to be considered is the parallel logic with activities *DataCheck* and *AntiFraud* together with activity *PaymentInterface*. The reason for considering execution logics (or their increments) is to allow a better performance for the adaptation process since it is faster to analyse smaller regions in a model, and find a solution for that region, instead of considering the composition as a whole.

For each operation O in the model involved in the adaptation process, a set of candidate replacement operations $SetCand(O)$ is identified for O ordered in descending order of best matches. The candidate replacement operations can either have (i) full match with O, (ii) match with only the signature of operation O, or (iii) no match with the signature of O.

During the selection process, the operations in $SetCand(O)$ that are considered as replacement options for O are operations that match at least the input parameters of O (as explained above, the operations in $SetCand(O)$ provide at least the same functionality of O, and are not marked as faulty or unavailable. An operation O' in $SetCand(O)$ is considered as a replacement operation if the data types of the input parameters of O' are subsumed under the input parameters of O. This is because it is necessary to guarantee that the input parameters of O' (yet to be executed) will be able to be defined by values that are available from the parts of the model that have been executed. In the case of O being the first operation in the execution instance, these values will be defined with the inputs

of the initial execution of the composition.

In other words, for each invoke activity of the execution instance, a set of candidate replacement operations is identified by the service discovery tool, based on functional and behavioural matching [130][146]. The identified candidates' operations are ordered by the weighted sum of the normalised response time and cost values of an operation, as per the Equation 3.14. The weights are used to specify priorities in QoS values. The ordered list of candidate services is used as part of an optimisation heuristic employed in the selection process described in Section 3.3.2.1.

Given an abstract invoke operation I_i and a set of candidate concrete operations O_n^i, where O_1^i, O_2^i, and O_3^i are example of candidates' operations for I_i the ranked order of these operations is given by the Equation 3.14 below.

$$V(O_n^i) = wRT * NormT(O_n^i) + wC * NormC(O_n^i) \tag{3.14}$$

where:

- O is a candidate service operation;
- $NormT(O_n^i)$ is the normalised value for the expected response time of the candidate operation O_n^i;
- $NormC(O_n^i)$ is the normalised value for the cost of the candidate operation O_n^i;
- wRt and wC are weights used for response time and cost, with $wRt + wC = 1$.

Considering m the number of candidate operation for an invoke operation I_i, the normalized value for the expected response time is given by the Equation 3.15 below.

$$NormT(O_n^i) = \frac{B_t^n - \min_{0 \leq x \leq m} B_t^x}{\max_{0 \leq x \leq m} B_t^x + \min_{0 \leq x \leq m} B_t^x} \tag{3.15}$$

where: B_t^n is given by Equation 3.10

Likewise, the normalized value for the cost of a candidate operation is given by the Equation 3.16 below.

$$NormC(O_n^i) = \frac{C(O_n^i) - \min_{0 \leq x \leq m} C_x^i}{\max_{0 \leq x \leq m} C_x^i + \min_{0 \leq x \leq m} C_x^i} \tag{3.16}$$

where: $C(O_n^i)$ is the cost of the candidate operation O_n^i for the invoke operation I_i.

For situations in class C1 (malfunctioning or unavailability), O may be just one of the operations that need to be replaced, given that the availability analysis (see Section 3.3.1.2) can identify other correlated operations to be replaced. In this case, the process temporarily replaces these other unavailable operations by a virtually created candidate, so that they can be considered when executing the adaptation process. As described below, the process relies on expected QoS values for all the operations in the model to be executed to verify violations of the SLA values of the whole composition. These temporary replacement operations must be changed during the execution of the adaptation process.

The overall idea of the process is to replace O with an operation in $SetCand(O)$, by traversing the set of candidates' operations in order, and choosing an operation in the set that has the best match with O and does not cause problems to the rest

of the execution model, if possible. Examples of these problems are violations of SLA values for the whole composition, or operation signature dependencies that cannot be resolved. The process terminates when either a replacement operation for O is identified in $SetCand(O)$, or all the operations in the set are considered without success. When analysing if it is possible to replace O by an operation in $SetCand(O)$, other operations not yet executed in the model may also need to be replaced to resolve SLA violations or operation signature dependencies. The adaptation process is recursive; it tries to identify replacement candidates' operations for those operations.

In the situation in which operations that match the signature of O are identified, the process verifies if any of these operations can be used to replace O by analysing if there are violations of the SLA values for the whole composition. If the SLA values are not violated, the adaptation process uses the replacement operation to change the execution model, and terminates (see Algorithm 3.1 and Algorithm 3.2 for a more formal definition).

If the SLA values are violated, the adaptation process identifies the list of operations in the model that have not yet been executed. For each of these operations, the process tries to identify combinations of candidate replacement operations that could maintain the SLA values of the whole composition, recursively. The order in which the operations in the list are considered takes into account execution logics (regions) in the model. When necessary, the execution logics are incrementally enlarged, until a combination of replacement operations is identified, or all remaining operations in the model are considered. If a combination of replacement operations is found, the execution model is changed, and the adaptation process terminates.

In the situation in which no operation that matches the signature of O can be used, or they did not exist in the set of candidates' operations for O, the adaptation process identifies within the list of operations in the execution model that have not yet been executed, the operations that have a signature dependency with the replacement operation for O. For each of these dependent operations, the process verifies if the replacement operation can be used and accepted by the dependent operation, or if replacement operations can be found for a dependent operation.

In the case in which signature dependencies between operations could not be resolved (i.e., no replacement operations can be found for a dependent operation that accepts the replacement of operation O), the process verifies if operations in the execution model can be replaced as a group by a candidate replacement operation, or a set of dynamically composed operations. A group of operations is considered based on execution logics, by adding one operation to the group at each time. If candidate replacement operations are found, the execution model is changed. If not, or there are no more operations to be added to compose an execution logic, the model cannot be adapted, and the adaptation process terminates.

The next section presents the method employed in ProAdapt to support dynamic adaptation, which uses the information provided during the analysis phase to generate a valid configuration for the service composition.

3.3.2.1 Dynamic Composition

As stated in the previous sections, ProAdapt supports the monitoring of different classes of events in order to predict faults, failures, and identify the need

for adaptation of one or various execution instances. Independently of the class of situation analysed, preprocessing performed or granularity of the adaptations required, the decision phase uses the same process to compute a new valid configuration for execution instances.

As states in Section 1.2, formally speaking, service composition is the process of composing services to provide a desired functionality under defined constraints. Dynamic service composition is thus the process of composing and modifying service composition during runtime. In general terms, since the functionality of a service composition is already defined during runtime, the adaptation of the composition templates and execution instances is considered an optimisation problem based on the selection of appropriate combinations of candidate service operations that satisfy the SLA parameters of a composition.

In order to avoid considering all possible combinations of service operations, which is time consuming, ProAdapt does not try to identify the best combination, but it looks for some valid combination given current constraints (e.g.; the candidates' operations, the QoS parameters of the operations that have already been executed, and the SLA parameter values of the whole composition).

In addition, ProAdapt uses a technique known as Intelligent backtracking, which is used to reduce search in combinatorial feasibility problems by using information derived from small sets of infeasible constraints discovered in one part of the search space to avoid searching other, similar, regions [41]. The idea is to ignore those combinations that are identified as invalid in the middle of the selection process. This can happen when candidates' operations used in a combination cause the composition to be invalid, independent of other candidates' operations in the combination.

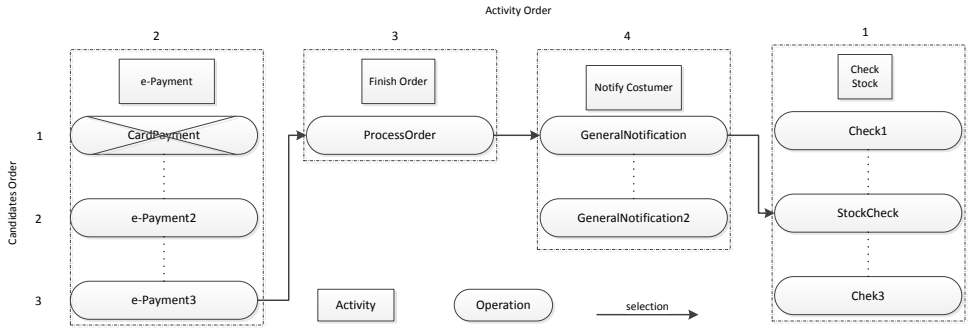

Figure 3.7: Example of a scenario for the selection of operations.

Figure 3.7 presents an example of a possible combination of operation endpoints for a simplified version of the execution model instance shown in Figure 3.5. More specifically, suppose that the model in Figure 3.5 has already been modified with activities *e-Payment*, *NotifyCustomer*, and *FinishOrder*, represented as boxes in Figure 3.5. Each activity in Figure 3.7 has a set of possible candidate operation endpoints in order of best match. As shown in Figure 3.7, activity *CheckStock* has three candidates' operations while activity *FinishOrder* has only one candidate operation. The numbers on the top of the activities represent the order in which these activities are executed in the composition. In the example, suppose operation *CardPayment* becomes unavailable. Consider that for activity *CheckStock* operation *StockCheck* has already been executed.

In the first time of the selection process, the approach identifies combination *e-Payment2*, *ProcessOrder*, *GeneralNotification*, and *StockCheck*. However, suppose this combination does not satisfy the required SLA values for the composition. In this case, the selection process marks the combination as invalid and continues to try for another combination. Suppose the second selected combination satisfies

the required SLA values and does not generate signature mismatches. In this case, the selection process identifies *e-Payment3*, *ProcessOrder*, *GeneralNotification*, and *StockCheck* as the new valid combination.

In addition, the selection process considers a special candidate which represents an artificial best combination of the available candidates for the particular abstract operation. More precisely, this artificial best candidate will have the best QoS parameters of all candidates and no constraint. This approach is useful to discard paths that are known to be invalid since early stages, and thus save time to find a valid configuration. Moreover, only operations that are free and available are considered.

In other words, the process retrieves the status of an abstract operation in the execution instance. An abstract operation is *allowed* to be replaced only if it belongs to the future path of operation to be invoked. Operations executed, in execution, or not reachable due to conditional flows, are not considered for adaptation. Similarly, operation that are marked as unavailable in the *Bind Information Repository* are not considered as candidade replacement.

The basic process for service selection comprises two procedures (a) Adapt and (b) Reconfigure. The first one sets a temporary environment for the reconfiguration procedure, while the last is the recursive backtrack algorithm to select candidates while satisfying constraints.

Algorithm 3.1 presents a pseudo-code for the Adapt procedure. The input of the algorithm is a service composition instance. In the first part of the algorithm, from line 2 until line 4 some local variables are set. The first step is to extract the template or workflow of the received instance (line 2). Next, from the extracted template, we get the list of all abstract operations deployed in the

Algorithm 3.1 Adaptation Algorithm

```
1: procedure ADAPT(instance)
2:     template ← instance.getTemplate()
3:     listAbstractOp ← template.getListOfAbstractOp()
4:     deployedMap ← instance.getDeployedMap()
5:     for all abstractOp ∈ listAbstractOp do
6:         candList ← bindInfoRepository.get(abstractOp);
7:         specialCandidate ← createSpecial(candList);
8:         candList.insertLast(specialCandidate);
9:         deployedMap.update(abstract, specialCandidate);
10:    end for
11:    compositionOK ← Reconfigure(instance, bindInfoRepository, 0)
12:    return compositionOK
13: end procedure
```

template, regardless of the logic structure. Finally, we get the map between the list of abstract operations defined for the template and the deployed candidate operations chosen for the instance.

The next part of the algorithm is a for loop that iterates over all abstract operation of the execution instance and creates a special candidate which represents an artificial best combination of the available candidates for the particular abstract operation. In line 6 the algorithm creates the local variable *candList* to hold the list of all possible candidates available in our *Bind Information Repository* (see Section 3.2).

More precisely, this artificial best candidate will have the best QoS parameters of all candidates and no statical constraint. This approach is useful to discard paths in the Backtrack algorithm that are known to be invalid since early stages, and thus save time to find a valid configuration.

Line 14 calls the actual reconfiguration process and uses it returned value to inform the status of the attempted adaptation or instance creation.

Algorithm 3.2 Reconfigure Instance Algorithm

```
procedure RECONFIGURE(instance, candsMap, index)
    listAbstractOp ← template.getListOfAbstractOp()
    abstract ← listAbstractOp.get(index)
    absSize ← listAbstractOp.size()
    candSize ← candsMap.size()
    allowed ← abstract.isToBeExecuted()
    compOK := false
    if allowed then
        attempt := 0
        notFound := true
        deployedMap ← instance.getDeployedMap()
        while (attempt < candSize − 1) & notFound do
            candidate ← candsMap.get(attempt)
            if candidate is available then
                deployedMap.update(abstract, candidate)
                candidate.incrementLoad()
                if index = absSize then
                    compOK ← checkConstraints(instance, candsMap)
                else
                    compOK ← checkConstraints(instance, candsMap)
                    if compOK then
                        compOK ← Reconfigure(instance, candsMap, index + 1)
                    end if
                end if
                notFound := compOK
                if notFound then
                    deployedMap.update(abstract, candsMap.get(candSize − 1))
                    candidate.decrementLoad()
                end if
            end if
            attempt := attempt + 1
        end while
    else
        if index = absSize then
            compOK ← checkConstraints(instance, candidatesMap)
        else
            compOK ← Reconfigure(instance, candidatesMap, index + 1)
        end if
    end if
    return compOK
end procedure
```

From the first line that set up the local variables for Algorithm 3.2, lines 6 and 7 deserve special attention. Line 6 retrieves the status of an abstract operation in the execution instance. An abstract operation is *allowed* to be replaced only if it belongs to the future path of operation to be invoked. In other words, operations executed, in execution, or not reachable due to conditional flows, are not considered for adaptation. If the abstract operation is not *allowed*, the procedure returns the status of the attempted configuration (line 31) or proceeds to the next index (line 33). Otherwise, the procedure will recursively attempt each candidate (line 22), until a valid configuration is found or all indexes have been tried.

3.4 Summary and Discussions

This chapter presents a proactive adaptation framework named ProAdapt, which was designed and developed as part of work undertaken in this thesis. The chapter starts by presenting the main goals and characteristics of the framework and highlighting the classes of situation handled. Then, the general concepts of ProAdapt are covered followed by the details regarding the components of the proposed architecture. The chapter also includes details of the process of adaptation and the set of strategies used by ProAdapt to predict the need for adaptation and improve the performance and reliability of service compositions.

The ProAdapt framework constitute various components that perform different tasks in order to accomplish the major goal of improving the performance, reliability, and general conformance with system requirements of service compositions. The implementation of these various components does not follow a

complete decoupled paradigm due to performance optimisations, nevertheless, the proposed components or techniques could easily be adapted to be used by other frameworks, enhancing out contribution.

We have spent a great deal of time prototyping the various proposed solution due to the complexity of the pieces of software involved. As mentioned before, standard execution instances do not provide a way to differentiate between execution instances or to dynamically update service compositions. Some modifications of the available tools were necessary to provide the required context for our experiments, independently of the proposed solution. For instance, since we were dealing with multitasking processes with high load, we even implemented a process scheduler for the java virtual machine in order to handle large amounts of parallel service composition requests. Another example of great implementation effort was to provide a minimum infrastructure to work with service composition on a simulated environment (NS-3) since the tool does not support the use of services by default.

Various drawbacks presented on the surveyed approached were tackled, resulting in different strategies such as the distribution of the load request amongst multiple operations, the ability to perform isolated and parallel adaptations and avoid unnecessary changes. Overall, we believe that we produced a good framework for proactive adaptation of service compositions.

Chapter 4

Experiments and Evaluation

In order to demonstrate and evaluate the ProAdapt framework, some prototypes were implemented. A total of three prototypes of the framework were implemented in an iterative way.

The first prototype (*Prototype I*) was implemented to demonstrate that the framework could predict the need for changes in service compositions due to QoS aspects such as response time and cost values, and unavailability of operations, services and providers. The first prototype proved all of our sub-hypothesis to be true, however, there was space for improvement, leading to two new prototypes.

The second prototype (*Prototype II*) expanded *Prototype I* to support all the different tasks executed by the adaptor component due to the four different classes of situations that may trigger the need for adaptation (C1 to C4). Even though all sub-hypothesis were already understood as true from the first prototype, the prototype gives another step into a formal adaptation framework, as specified in the fifth objective.

Finally, the finds of both *Prototype I* and *Prototype II* lead us to believe

that the framework could be further improved. (*Prototype III*) supports parallel adaptation of service composition that have overlapping service operations and load balancing between candidate operations.

General Assumptions and Observations

- **BPEL:** All three prototypes assume service compositions expressed in WS-BPEL.
- **Stateless Services:** Operations can be changed at any time of the execution process of the service composition
- **Logic Regions:** During our experiments we have not considered repeat regions, such as *While* or *For*. The framework, however, can support such activities, since it is only a matter of defining the aggregate function for each of them (see Section 3.3.1.1).
- **Requirements:** Even though the framework supports the evolution of requirements such as the cost or overall response time of the whole composition, such changes in the requirements were not considered in our experiments. However, these requirements are considered for every new instance created and the experiments consider changes of internal QoS aspects, thus we believe very little would be added by including experimenters specifically target at requirements evolution.
- **Response Time:** During our experiments, the response times used during the analysis are collected from the request agent perspective. In the case of response time for the whole composition, that time can be interpreted as

measured from the end user. While in the case of invoked operations, such time is collected from the execution engine point of view.

In the following we describe each of the three prototypes, the experiments executed for each prototype, and the results of these experiments.

4.1 Prototype I

The main goal of the first prototype was to verify the ability to dynamically identify the need for adaptation in service compositions and to replace service operations during run time.

Prototype I was implemented in Java. In this prototype the execution engine and the adaptor components were implemented as a single component for simplicity. The prototype assumes service compositions expressed in BPEL4WS exposed as Web Services using SOAP protocol. The prototype also assumes operations in the compositions and user requests emulated using SoapUI [74], and exposed through Apache Axis 2 [119].

In *Prototype I*, candidate service operations are registered in a local memory-based database, which is assumed to be maintained by the service discovery tool. An external file containing the web service description could also be used to populate the local candidate database.

To execute the composition instances an adaptation of the Apache ODE Execution Engine was used. Since Apache ODE does not provide a way to proactively and dynamically adapt composition instances, we had to modify the tool to support our needs.

Basically, we changed Apache ODE to include additional information as annotations in the request message regarding the service operation been invoked, such as the associated service composition, user session, and logic region. Additionally, a proxy was implemented to route the request messages to the desired providers. Without such modifications, there would be not way to replace a deployed service operation for a particular abstract invoke activity, or replace candidates for individual instances.

In order to evaluate the work in *Prototype I*, a service composition with 12 invoke activities was developed and mapped to 12 distinct service operations, distributed within sequence and parallel execution logics. Figure 4.1 presents the service composition used in the evaluation.

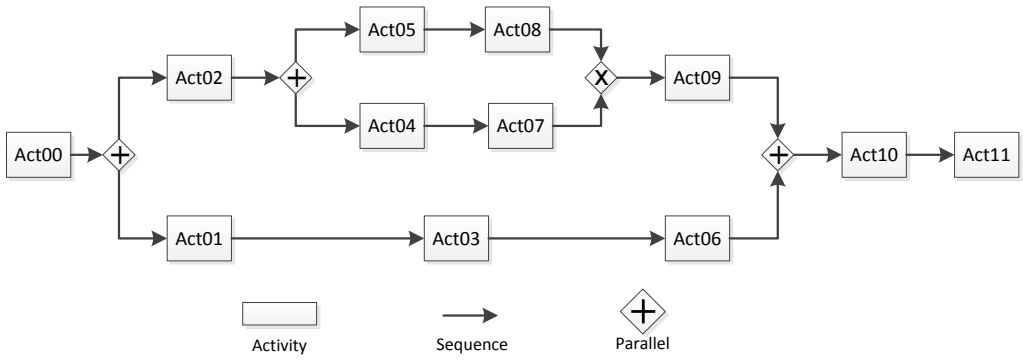

Figure 4.1: Experiment service composition

For all experiments concerning *Prototype I* (see Section 4.1.1), the 12 operations used in the composition had the same syntactic description, but each operation implemented a different functionality. These operations were configured to require two input parameters and produce one output result. Each operation had different associated costs and processing time values, as summarised in Table 4.1.

The experiments were conducted using five different machines, namely: (a)

Table 4.1: Configuration of experiment environment.

Machine	Configuration	Services Operations	Cost (pence)	Processing Time (ms)	Network Links (Mb/s)
Client (C)	Turion 1GHz 2GB RAM	-	-	-	C-A = 3.0
Adaptor (A)	Core 2.33 GHz 3GB RAM	-	-	-	-
Provider P1	Pentium 4.3 GHz 1GB RAM	S0:O00,O04,O08 S1:O01,O05,O09 S2:O02,O06,O010 S2:O02,O06,O010 S3:O03,O07,O011	100	150	A-P1 = 1.0
Provider P2	Core 1.86 GHz 2GB RAM	S0':O00,O04,O08 S1':O01,O05,O09 S2':O02,O06,O010 S3':O03,O07,O011	150	100	A-P2 = 1.5
Provider P3	Pentium 3.0 GHz 3GB RAM	S0":O00,O04,O08 S1":O01,O05,O09 S2":O02,O06,O010 S3":O03,O07,O011	300	50	A-P3 = 3.0

client machine responsible to create simultaneous requests to the service composition, simulating several concurrent users; (b) adaptor engine machine connected to three service providers; and (c) one machine for each service provider P1, P2, P3. The machines were connected using a network switch that allowed fine control of the communication link characteristics such as bandwidth control.

Figure 4.2 presents a summary of the testbed configuration including the speed of the network links between the machines. We used different speeds for the network links to emulate bottleneck situations that may occur when using an Internet environment.

Each service provider contains four different services, with each service implementing three different operations (see Table 4.1) that can be used in the composition created for the experiment (see Figure 4.1). More specifically, each

Ox presented in Table 4.1 is a service operation that implements the required functionality of the abstract invoke activities $Actx$ presented in Figure 4.1.

The operations in the four services in provider P1 are similar in terms of their functionality to the ones in providers P2 and P3, in order to simulate possible candidate replacement operations. We assumed different costs and processing time values for the operations in the various providers as summarised in Table 4.1.

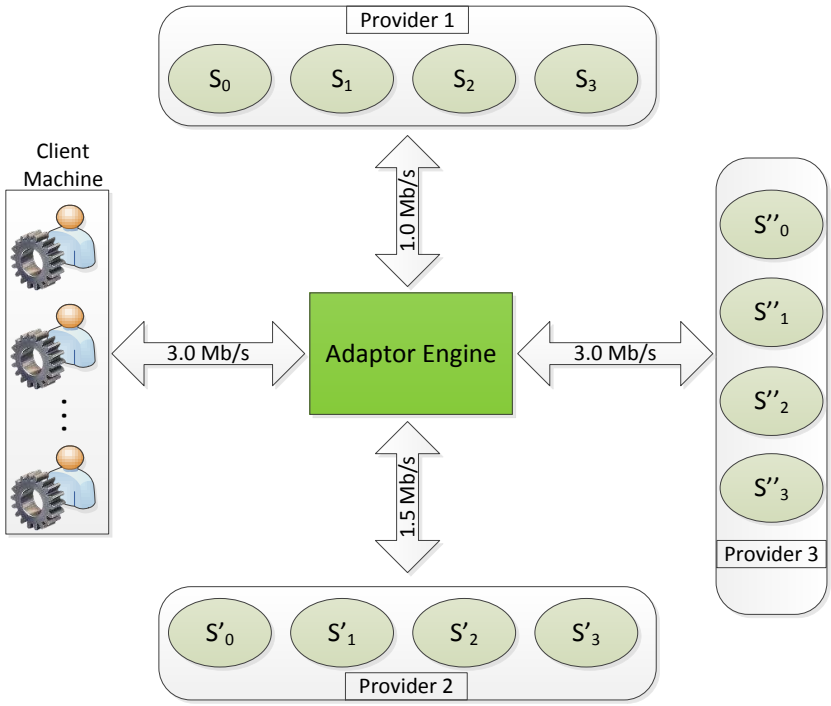

Figure 4.2: Testbed Configuration for Prototype I

It is a well-known fact that in a real scenario the operations used in a composition may be from different providers. In the experiment, for the initial execution model a decision was made to use all operations from the same provider (P1) to enforce a more realistic bottleneck situation. This does not invalidate the experiments since it is important to consider the network capacity between the adaptor

and providers.

Moreover, the simulations were performed considering the parameters shown in Table 4.2. The size of each request or reply message in an operation was around 60 bytes. The expected SLA values for the composition were defined for cost as 2800 pence, and for response time as 3.5 seconds. Moreover, since the cost associated with operations do not change over time, the weights for the cost and response times were defined as 0.9 and 0.1, respectively (see Section 3.3.1.1 for details about how weights are used to specify priorities in QoS values).

Table 4.2: Experiments parameters for Prototype I

M. Size	Cost	Time	C. Weight	T. Weight	EWMA T.	EWMA W.
60 Bytes	2800 p	3.5 s	0.9	0.1	1.5	0.6

These two QoS aspects were considered because of their importance for service-based systems (see Section 1.3), the strong correlation that exists between response time and cost aspects (there is normally an increase in the cost when providing faster services), and the possibility of demonstrating the prediction of response time values for the composition and their deployed operations.

For the EWMA function parameters (see Section 3.3.1.1), we used a threshold of 1.5 and the weight for past expected response time values of 0.6. These values were identified after executing the operations in the composition in Figure 4.1 several times, and verifying that with these values the expected response times of the operations were below their observed times for 95% of the cases.

4.1.1 Evaluation

The first prototype was used to evaluate the framework from three different perspectives, as described below:

Study (I.1) : To demonstrate that the framework improves the composition performance in terms of the time to execute and complete the whole composition when the availability of operations is proactively identified;

Study (I.2) : To demonstrate that in the event of faults or performance degradation of service operations, the framework manages to adapt the composition instances, ensuring the SLA values;

Study (I.3) : To analyse the performance of the framework;

The evaluations performed with the three studies for the first prototype are describe below.

Study I.1

For the first study of *Prototype I*, the evaluation analyses the performance gain obtained by using the spatial correlation technique used in the approach. More specifically, the study analyses the benefits of predicting the availability of operations used in the composition (see Section 3.3.1.2) by comparing the average time to finish the execution of the whole composition when there is no need for adaptation with the different times to finish the execution of the whole composition when availability is proactively detected in the three different levels, namely (a) operations, (b) services, and (c) providers.

In the situations above, the processing times of the operations in providers P2 and P3 were set to the same as P1. This is necessary to create a homogeneous environment and avoid using a replacement operation with faster processing time and, therefore, diminishing the impact of the time wasted when trying to invoke an unavailable operation.

In order to get an overall picture of the performance gain caused by the three different levels of spatial correlation among operations, services, and providers, provider P1 is set to be unavailable. However, depending on the levels of spatial correlation used, the approach adapts the service compositions in different ways, with direct impact in the time to complete each execution instance. Figure 4.3 presents the results of this experiment.

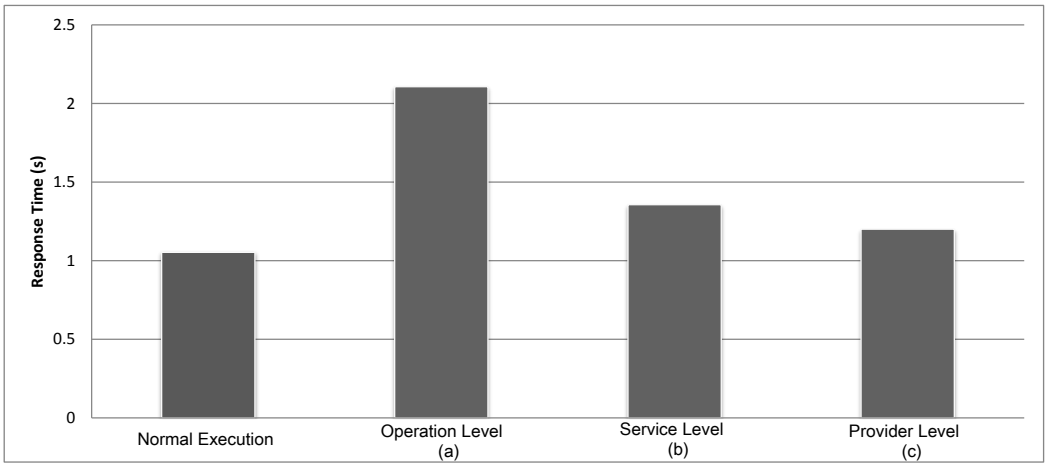

Figure 4.3: Impact of spatial correlation on composition response times

The first bar in Figure 4.3 is plot for comparison and represents the average normal execution time to finish the service composition.

As shown in the figure, for case (a) the time to execute the composition and change all operations without considering any spatial correlation is two times

greater than the time to execute the composition without any need for adaptation (normal execution). In other words, for the 12 operations, the adaptation approach only identifies the fault after trying to execute each of them.

When the spatial correlation is performed at the service level (b), unsuccessful invocations can be avoided for each operation from the service identified as unavailable. This approach resulted in a improvement of the time to execute the composition of about 36% when compared to case (a) and an increase in time of close to 28% when considering the normal execution.

Finally, in study (c) spatial correlation is considered at all levels, which means that all operations from the same service or provider of the one identified as unavailable will be marked for replacement. In this case an execution instance will observe only one fault, observed when the first activity that attempts to invoke the operation O_{00} through the unreachable provider P1.

As shown in Figure 4.3, the adoption of the spatial correlation at provider level was able to improve the execution time of the whole service composition by 43% when compared to case (a) with a penalty of only 14% when considering the normal execution. Since we can interpret case (a) as basically an optimised reactive adaptation approach, we can conclude that the results testify an improvement of using the proactive adaptation approach using spatial correlation when compared to the situation in which a non-proactive approach regarding availability is used.

Study I.2

For the second study of *Prototype I*, the client machine was configured to support an incremental increase in the number of users invoking the composition.

More specifically, the study analysed *Prototype I* in 30 intervals of ten seconds each (total of 300 seconds) with a rate of one user per second in the first interval of ten seconds, two users per second in the second interval, and an increment of one extra user every ten seconds reaching 30 users per second in the last interval.

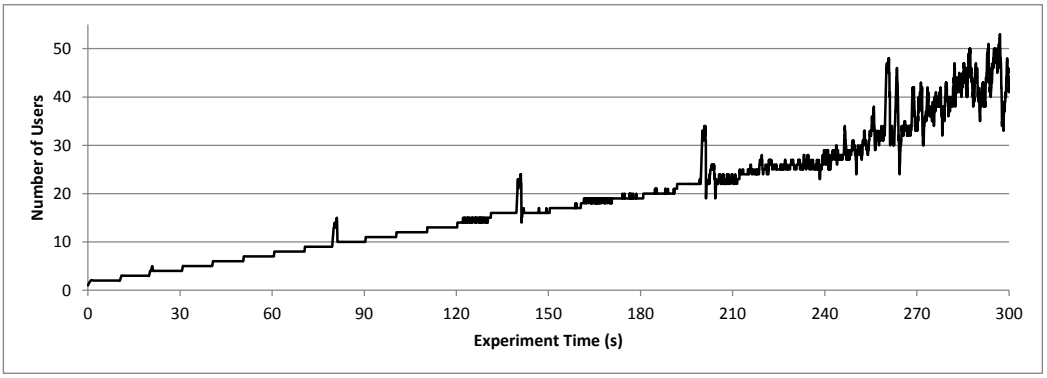

Figure 4.4: Number of simultaneous users consuming resources.

It is worth noting that for each user request performed by the client machine, 12 operations need to be invoked, therefore, the number of simultaneous operations in execution for the experiment is greater than the number of user requests in the various intervals. Moreover, at a certain time t in the experiment, the number of users invoking the composition and consuming resources is greater than the number of new users at t, since the time to execute the composition without any problem is more than 1.5 seconds (see first column in Figure 4.3) and, therefore, requests for the compositions are accumulated over time intervals.

The above simulation was used in order to provide an environment that could on its own create faults to the composition in terms of response time and cost values due to the number of users and network resources being consumed and, therefore, allow the verification of the behaviour of ProAdapt in cases of faults.

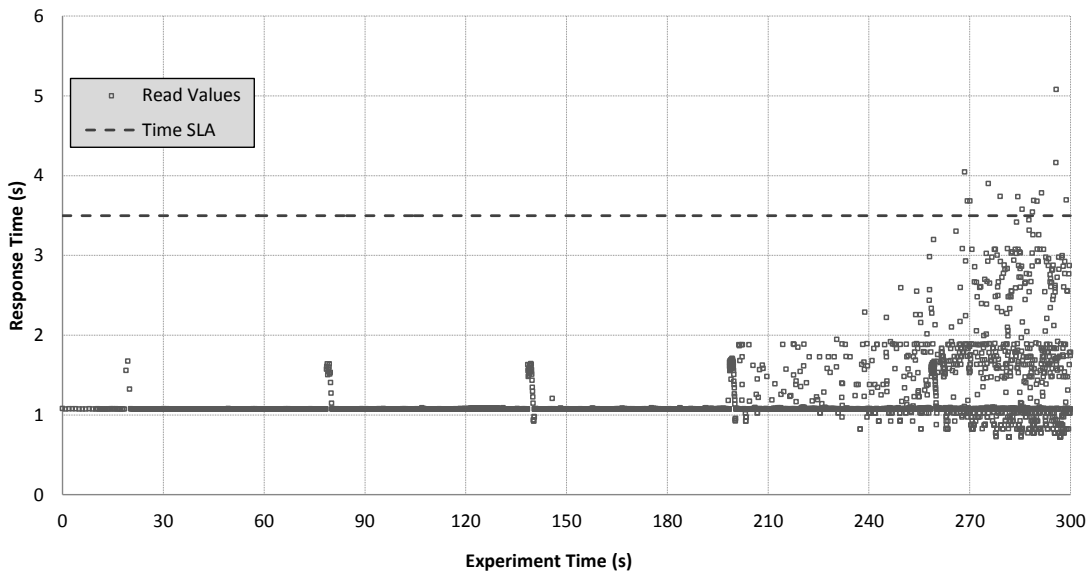

Figure 4.5: Variation of composition response times during the experiment.

Figure 4.4 shows the number of concurrent users consuming resources during the different times of the experiment. As expected, the accumulated number of users is greater than the rate of new users invoking the composition. The graph in Figure 4.4 shows a larger number of accumulated users towards the end of the experiment. This is due to the fact that during this time there were a larger number of invocations for the operations in the execution instances causing degradation in the response times of the operations, delaying the execution of the whole compositions, and accumulating even more requests.

Figure 4.5 shows the time to execute each execution instance of the composition (represented as squares) during the whole experiment, and the execution instances that were able to adapt themselves and finish within the SLA response time values (dotted line in the graph). As shown in the figure, the majority of the execution instances managed to adapt themselves and finish within the

SLA response time value. The graph also shows a stable response time for the execution instances in the beginning of the experiment and oscillations in the response times starting at 200 seconds after the experiment has started. This is because between 20 and 25 service composition requests per second, provider P1 reaches its full capacity, causing degradation in the operations response times, and eventually, the need to adapt the composition execution instances so that their respective SLA values are maintained.

The cases in which the execution instances did not finish before reaching the SLA values for response time (cases above dotted line in Figure 4.5), were due to bottlenecks in the providers and lack of available operations that could be executed faster and with the cost values specified in the experiment.

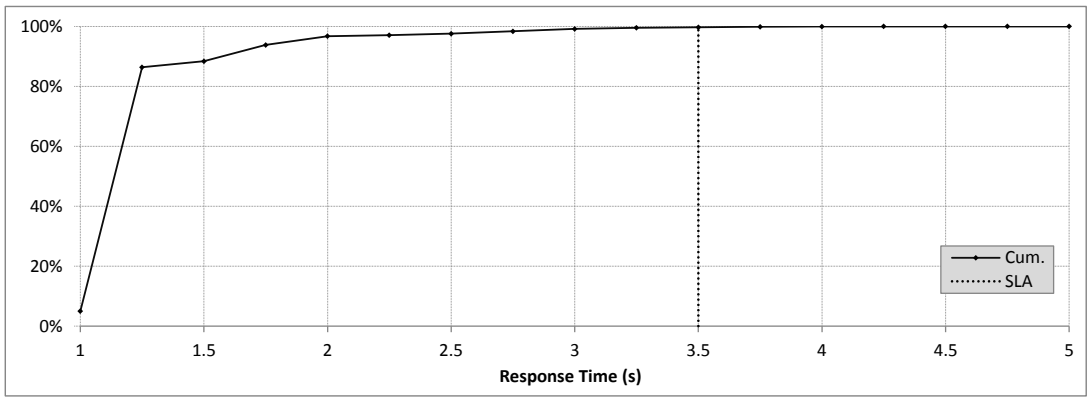

Figure 4.6: Cumulative frequency distribution of response times.

In order to verify the impact of these cases, Figure 4.6 presents the cumulative frequency distribution for the response times of the executions. As shown, the response times of the executions where below 1.5 seconds for 80.65% of the cases and in 99.69% of the cases the execution instances respected the SLA response time values. Therefore, from a total of 4650 user requests performed during 300

seconds of running the experiments, only 14 user requests could not be completed for the given SLA values.

Similarly, Figure 4.7 presents the cumulative frequency distribution for the costs of the execution instances. As shown, the cheapest executions (1200 pennies) occurred in 85% of the cases. The graph also shows that the SLA cost value was respected in all 4650 requests, which was expected since the framework identifies only operations that respect the cost values when adapting the composition, and cost values of the operations does not change over time in our experiments.

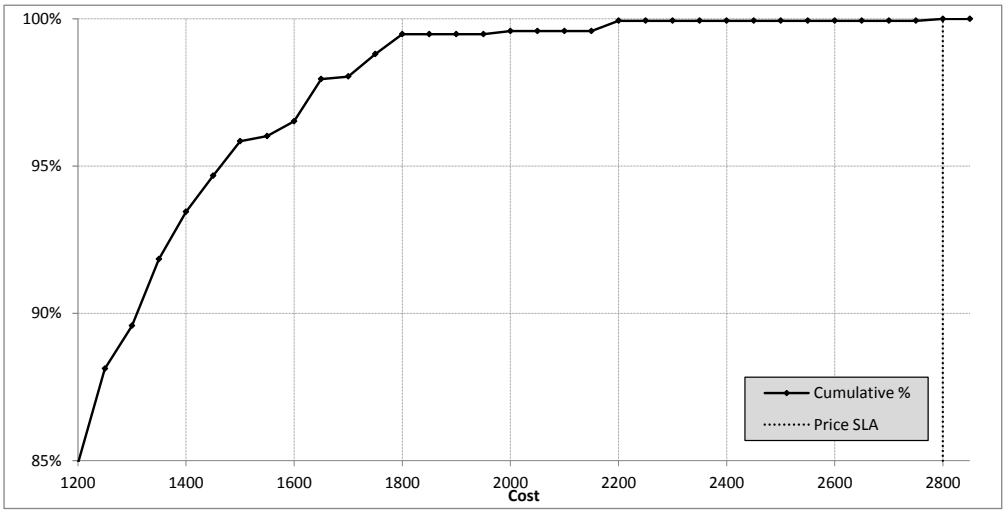

Figure 4.7: Cumulative frequency distribution of cost.

Study I.3

As described above, the third case for *Prototype I* analyses the time spent to adapt the execution instances, in order to verify the impact of such time in the overall composition execution time.

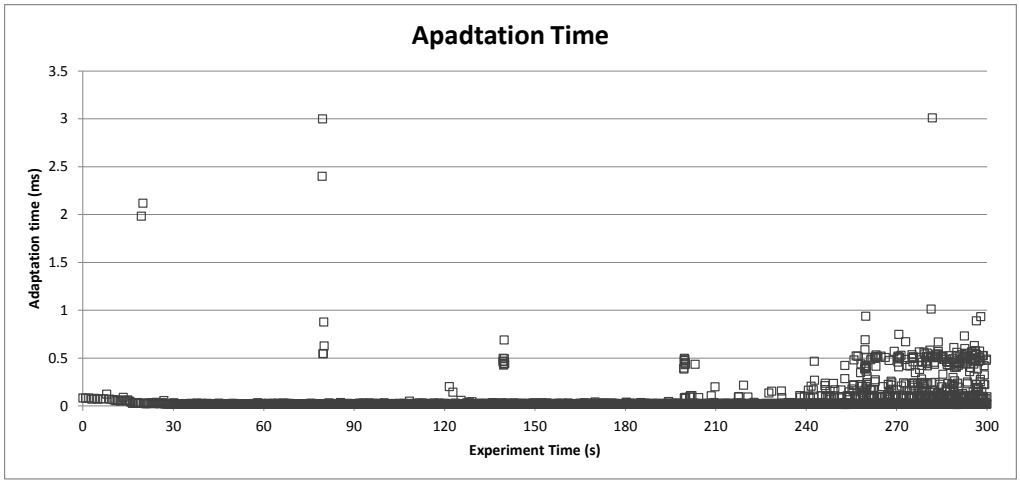

Figure 4.8: Adaptation time for each composition execution over the experiment.

Figure 4.8 shows the adaptation time measured in milliseconds for all executions in the experiment, while Figure 4.9 shows the cumulative frequency distribution.

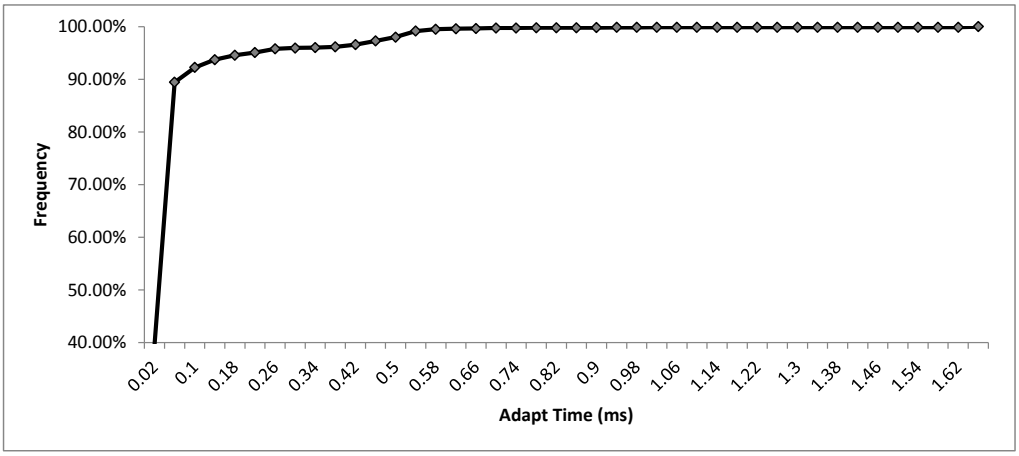

Figure 4.9: Cumulative frequency distribution of the adaptation time.

As shown, in the majority of the cases the overall adaptation time is very small when comparing with the time to execute the composition itself. Most of

the time, when required, the adaptation time comprised less than 0,0005% of the time required ti finish the execution instances. Even for the longest adaptation time observed, such ration was kept below 0,003%.

This results are possible because the adaptation can be performed while the execution engine is waiting for the reply of invoked service operation. In other words, the adaptation process can be performed in parallel. Moreover, the adaptation process is implemented in a way such that compositions are analysed based on execution logics and without looking for the optimal combination of operations, but for combinations that meet given SLA values.

Overall, the results for the first prototype prove to be very positive and demonstrate that the framework can support proactive adaptation of service composition during execution time, due to different QoS characteristics, more specifically, due to response time and cost values. The experiments also show that the performance of the adaptation process is good and that the process does not cause significant penalties when changes in the composition are necessary.

4.2 Prototype II

The second prototype was implemented with the main goal of verifying the ability to perform structural changes in the workflow and evaluate the performance of each task required by the adaptation framework.

More specifically, *Prototype II* was implemented to analyse the duration of each sub activity of the adaptation process, namely (a) time to identify if a replacement operation causes violation of the SLA values for the whole composition, (b) time to resolve SLA violations, (c) time to identify signature dependency prob-

lems, (d) time to solve or attempt to solve signature dependency problems, (e) time to adapt by changing groups of operations in a composition, and (f) time to identify spatial correlations due to operation, service, and provider unavailability.

Because it was necessary to have full control of the web service infrastructure (i.e. network links, request entity, provider entity, and discover entity) in order to measure and understand the impact of each task and algorithm within the adaptation framework, a decision was made to implement *Prototype II* in a complete simulated environment based on the Network Simulator 3 (NS3) [62].

NS-3 is a discrete-event network simulator that was proposed as a replacement of the widely used NS-2 simulator [102]. It was developed with a new core from scratch, using good software engineering strategies, what results in a well define architecture, entirely written in C++ programming language, and with support of open source community. One of the key aspect of the NS-3 simulator is the attention to realism. The various core modules are designed to be a faithful representation for real-world network devices and protocols.

The use of the NS-3 simulator makes some things a little easier for the programmer. For instance, one does not need to give much special attention for handling race conditions and similar multithreading issues due to NS-3 be a discrete simulator. This helped in reducing the complexity of the code for the execution engine and execution instances. Moreover, the event-oriented nature of the framework is much easily accomplished in the NS-3 environment due to the simulator itself be presented in such a way. There is no need for special communication code since passing events between components is a feature already presented.

The list of benefits of using the NS-3 simulator in our experiments were not

short, however, it is important to note that all components of the framework had to be implemented in C++ as components of the NS-3 stack. Some of these components required additional modifications to suit the new requirements.

The service discovery, for instance, does not work with an external database and was implemented just to make visible a set of services. The composer could not rely on the same library previously used to interpret WS-BPEL files and a new approach had to be employed.

Moreover, the foundation of NS-3 is very strong, providing support for both IP and non-IP based networks, and many different communication protocols, but it does not provide built-in support for services. In other words, there was no set of components or libraries to create and use service operations. Therefore, a new set of modules was implemented to simulate the required interaction between a service requester and the required service operation. This included the implementation and addition to the NS-3 stack of protocols, such as the HTTP and SOAP. Service operations, web services, and service providers were also implemented as abstract entities with attached QoS parameters.

When a requester entity attempts to invoke a service operation, the bind information is used to send the request to the correct provider, which will check if the required service is register that in turn checks for the required operation. If the service is not present, or a operations is not present, the relative error message is produce, otherwise, a standard reply is produced. We do not implement the service agent, but this standard reply is a valid answer for the operation with processing time established by the QoS parameters of the provider.

The implemented components are loosely coupled and can potentially be reused by other approaches and researchers, specially since we were able to pro-

vide with such implementation a simulated web service infrastructure environment. As stated in Chapter 1, such Simulated Web Service Infrastructure is one of the main contributions of the work described in this thesis.

It is important to note that using a simulated infrastructure does not invalidate the experiment. In fact, because this work is concerned with the adaptation of service compositions due to events collected or predicted in the execution context, the simulated environment actually allows testing of the approach in situations that would be difficult to be generated or manipulated on a real environment.

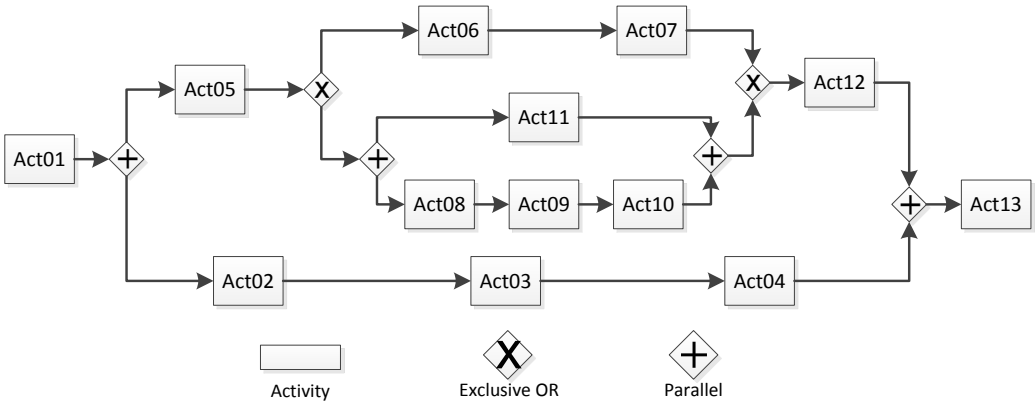

Figure 4.10: Compositon workflow logic with invoke activites for evaluation of Prototype II

The evaluation was executed in a service composition with 13 activities and different types of execution logics, as shown in Figure 4.10. The 13 activities in the composition are executed by 13 operations from ten distinct operation endpoints. There were four candidate replacement operations for each service operation Op_i in the composition, which gives a total of 50 different operation endpoints. The operations used in the experiments have different numbers and

types of input and output parameters. Moreover, each operation has at least a signature dependency with another operation.

Prototype II was evaluated in terms of the performance of the adaptation process using a Pentium 4 3GHz with 3GB RAM. More specifically, two studies were analysed: (a) the performance of the main activities in the adaptation process (study II.1); and (b) the performance of the whole adaptation process for problems in different parts of the composition (study II.2).

In the same way as in *Prototype I*, both cost and response time were considered as valid QoS aspects for the candidates service operations and the SLA values of the whole composition. The combinations of the response time and cost values for the operations used in the experiments were (5ms, 25p), (10ms, 20p), (15ms, 15p), (20ms, 10p), (25ms, 5p). Since both cost and response time were important for the experiment, the same weights were used to order the candidate services. The 50 endpoint operations in the experiment were associated with 25 different services and five providers.

4.2.1 Evaluation

Study II.1

The first study case for Prototype II focused in the analyse of the duration of each sub activity for the adaptation process (sub-activities (a) to (f) described above).

Table 4.3 shows the results of these times in nanoseconds for 100,000 executions with mean, standard deviation (SD), median, minimum, and maximum

values. In activity (a) it is measured the time to compute the aggregation of the observed and expected QoS values of deployed operations in a composition and to verify if there is a violation of the SLA values of the whole composition. It is important to note that this time is independent of where a problem occurs in the composition since all the operations need to be considered.

Table 4.3: Performance per subactivity of the adaptation process.

		Results				
Act	**Cases**	***Mean (ns)***	***SD (ms)***	***Med. (ns)***	***Min (ns)***	***Max (ns)***
(a)	-	593	2	593	589	606
(b)	-	84704	866	84802	82527	87018
(c)	1 dep	1119	11	1117	1110	1214
	3 dep	3418	18	3419	3410	3498
	5 dep	5891	25	5881	5865	5993
(d)	1 dep	8902	255	8775	8736	9646
	5 dep	25406	333	25259	24965	26215
(e)	G 1	37346	439	37098	36669	38854
	G 2	48653	788	48294	47561	51078
	G 3	56394	514	56528	55575	57987
	G 4	65278	541	65381	64372	67069
(f)	Oper.	885	13	881	871	979
	Serv.	7031	95	1026	6765	7447
	Prov.	18033	333	17960	17368	19276

In order to analyse the time for activity (b), candidate replacement operations were introduced which did not create signature dependency problems, since in this case the interest is in the time elapsed to resolve only SLA violations. The result shown in Table 4.3 is for the worst case scenario in which the combination of replacement operations that satisfies the composition uses all the last candidates in the normalised list for each operation (see Equation 3.14). As shown in

Table 4.3, the time achieved by the approach is almost negligible, which can be explained by the use of the optimisation technique to select candidates and to the fact that we look for a possible combination, not necessarily the optimal one.

The analysis of activity (c) considered the situation in which an operation O needs to be replaced and O has a dependent operation O'. The study included situations in which O' depends only on O, O' depends on three operations including O, and O' depends on five operations including O (see signature dependencies in Section 3.3). As the number of dependencies increases, so the required adaptation time, since more verifications need to be performed. The results in Table 4.3 show the time to verify if O' has dependency problems with one, three, and five operations. The results show that the time to execute this situation increases linearly with the number of dependencies.

In activity (d), the analyses considered similar scenarios as in (c), but now the time to find a solution is also measured. In any of the considered cases, the solution was identified for the last candidate replacement operation. For the case in which there were five dependencies, the study considered a scenario in which the current operation can only satisfy one dependency; the first candidate can satisfy two dependencies; the third candidate can satisfy three dependencies, and so on until reaching the fifth candidate that satisfies all five dependencies, which creates a total of 19 verifications.

In the case of one dependency, one would expect a ratio of one to five between the time to find a solution (d) and the time to identify the problem (c) in a dependent operation, given that five candidates were used in the experiment. The additional time shown in the results for the case of one dependency is due to extra activities executed by the approach (e.g., selection of candidate operations

and execution of changes in the model). A similar situation is found for the case of five dependencies given that 19 verifications where performed.

The measurements for activity (e) considered a problem in operation Op_8 that can only be resolved by trying to replace a group of operations in the composition. A analyse was made for the time to resolve the situation in four different groups of operations. More specifically, the study considered the following groups: G_1 with operations Op_8, Op_9; G_2 with operations Op_8, Op_9, Op_{10}, Op_{11}; G_3 with operations Op_5, Op_6, Op_7, Op_8, Op_9, Op_{10}, Op_{11}, Op_{12}; and G_4 with all operations in the composition (Op_1-Op_{13}).

The results in Table 4.3 show a linear increase with the number of execution logics used in the evaluation, and not with the number of operations being grouped. This is due to the incremental increase of the execution logics by the approach when trying to find a solution.

Activity (f) shows the times to identify spatial correlations when an operation, service, and provider become unavailable. The study considered these cases as initial steps for resolving problems in class C1, which are followed by the steps in cases (a) to (e), explained above. The time spent in case of provider unavailability is higher than the time for service unavailability, which is higher than the time for operation unavailability, due to the number of operations associated with a service and a provider.

The results in Table 4.3 demonstrate that activity (b) has the higher time of execution due to the large number of combinations of replacement operations that this case needs to consider. It is also observed from the results, that the activity to group operations requires more time to be executed when compared to the other activities.

Study II.2

In order to measure the performance of the whole adaptation process, the second case for Prototype II considered extra signature dependencies between operations in the composition, namely Op_3 depends on Op_1; Op_9 depends on Op_5; Op_{12} depends on Op_5; and Op_{13} depends on Op_{12} and Op_4. In other words, the output of some operations are directly required as input for other operations.

This case was analysed for problems in different parts of the composition. More specifically, the study considered problems in operations (i) Op_1, (ii) Op_8, and (iii) Op_{11}. For situation (ii), the problem in Op_8 occurred after operations Op_1, Op_5, and Op_2 have been executed. For case (iii), the problem in Op_{11} occurred after operation Op_1, Op_5, Op_2, Op_6, Op_7, and Op_8 have been executed.

Differently from activity (b) in study II.1, where the only combination that satisfies the SLA values for the whole composition is the one that uses all the last candidates for each operation, in this case there are other combinations that satisfy the SLA values. However, the only combination that satisfies both SLA values of the composition and signature dependencies of operations is the one that uses the last candidate operations.

Table 4.4 shows the results of these experiments in milliseconds for 100,000 executions with their mean, standard deviation (SD), median, minimum, and maximum values. These results involve all the activities discussed in Table 4.3.

As shown in Table 4.4, the adptation time for problems in an operation in the beginning of the execution of the composition (Op_1) is higher than the adaptation time for an operation in the middle of the execution (Op_8), which is higher than the adaptation time for an operation in the end of the execution (Op_{11}).

Table 4.4: Performance gor the whole adaptation process.

	Problem Location		
Metric	***Op1***	***Op8***	***Op11***
Mean (ms)	1.015005	0.954445	0.206295
SD (ms)	0.004789	0.003574	0.017664
Min (ms)	1.0106	0.950112	0.202703
Median (ms)	1.01336	0.953046	0.203881
Max (ms)	1.04865	0.967082	0.698439

This is because, there is a need to verify more steps in the adaptation process and consider more combinations of replacement operations when a problem occurs in the beginning of the composition execution instead of towards the end. The probability of finding a solution when a problem occurs in the beginning of the execution is also higher than when the problem occurs towards the end due to the number of available replacement candidates.

In other words, as soon as an issue is identified for an execution instance, less operations deployed for the execution instance would have been executed. Thus, more reconfiguration options would have been available and it would be more likely to find a solution for the issue under the required SLA.

Overall, the results of the experiments performed for Prototype II show that the time for the adaptation process is small when considered the time that it takes to execute an operation, to send the operation request and receive its response over the network, along with other times involved in the service composition execution. Thus, the proposed adaptation process can be used without causing

considerable performance penalties to the execution of the composition.

4.3 Prototype III

The idea of the *Prototype III* was to study the parallel adaptation as described in Section 3.3.1.3 and load balancing technique as desctibed in Section 3.3.1.4. Since concurrent issues are strongly related to physical resources available and how the operating systems manage them, the third prototype did not use the virtual environment created for *Prototype II* and extended *Prototype I* instead.

More precisely, in order to demonstrate and evaluate the work we have implemented a prototype tool of the framework in Java. The tool assumes service compositions in BPEL4WS [73] exposed as Web Services using SOAP protocol, and participating operations and user requests emulated using soapUI. The service discovery tool was also implemented in Java and is exposed as a web service.

Prototype III aimed at demonstrating that the adaptation process may be improved, resulting in better performance and reliability of service compositions, when the analysis of events and detection of the need for adaptation is shared and triggered among multiple instances of the same or different compositions executing in parallel.

The approach has been evaluated in order to demonstrate if there are improvements in the adaptation process in terms of the number of service compositions that can adapt successfully. The evaluation considers the situation in which the analysis of events and detection of the need for adaptation is shared and triggered among multiple instances of the same service composition, or different service compositions executed in parallel. Given that the process is the same for

these two cases, in the evaluation we considered multiple instances of the same service composition.

In the experiments we assume that each of the various execution instances starts its execution in different time-steps. We also consider that the number of running execution instances at different points of their execution flow is approximately the same. More specifically, the number of running instances executing the initial part of their flows is similar to the number of those in the middle or in the end of their execution flows.

4.3.1 Evaluation

Similarly to Prototype I, three perspectives were used to evaluate this prototype, as described below:

Study (III.1) : To compare the performance of the use of the parallel and proactive approach with the proactive approach only for a simple service composition;

Study (III.2) : To extend the comparison between the parallel and proactive approach with the proactive approach only for a complex service composition;

Study (III.3) : To analyse the performance of the load balancing technique;

In studies III.1 and III.2, we assume that one or more operations in the composition become unavailable. However, the approach supports a similar process for the other types of problems (e.g.,violation of QoS values of an operation). The

evaluations performed with the three studies for the third prototype are describe below.

Study III.1

For the first study, the idea was to show the performance improvement of the parallel approach under a simple composition where it would be straight forward to understand and interpret the results, without losing representativeness. We decided to start using a service composition with a sequential workflow formed by ten *invoke activities*, as shown in Figure 4.11.

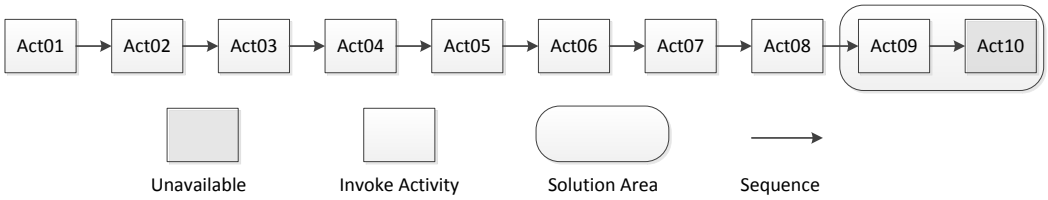

Figure 4.11: Service composition workflow for evaluation of Scenario 1

Moreover, three different scenarios were created to simulate different circumstances as described below.

Study III.1 - Scenario 1: For this scenario we assume that at a certain time in the experiment the service operation assigned to the last invoke activity (Act10) of the worflow presented in Figure 4.11 becomes unavailable. Consider the existence of a set of candidate service operations for each invoke activity (Act1 - Act10), and that the use of any of the available candidate service operations for Act10, along with the current assigned operations for (Act1 - Act09) would cause a

violation of the SLA value of the whole composition. Consider the existence of a valid configuration for the service composition when replacing both the operations assigned for activities Act9 and Act10.

We compared the number of execution instances that (a) were able to adapt successfully (successful), (b) were not able to adapt (unsuccessful), and (c) did not require adaptation because they were not affected by the problem (not required), for the case in which we used the parallel and proactive approach with the case in which we used only the proactive approach. We considered 50, 100, 150, and 200 execution instances of the composition shown in Figure 4.11.

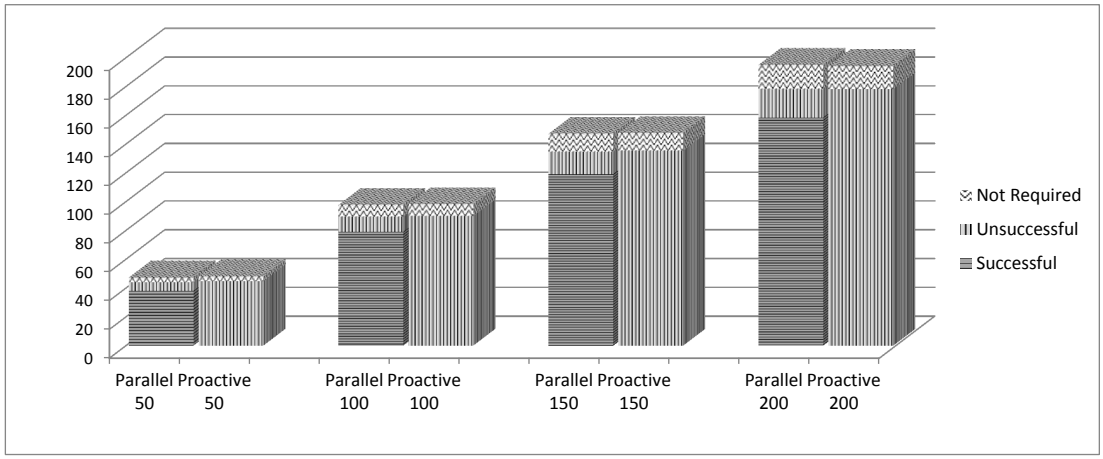

Figure 4.12: Comparison of the adaptation process for Case 1 - Scenario 1.

Figure 4.12 presents the results of this experiment. For each different number of execution instances considered in the experiment, the first column represents the results when using the parallel and proactive approaches (specified as *parallel* for simplicity), while the second column represents the results when using only the proactive approach (specified as *proactive* for simplicity).

As shown in Figure 4.12, when using the combined parallel and proactive approaches there are many more instances that are adapted and finished success-

fully. This is because in the parallel approach, several execution instances that are still in operation are notified about the unavailability of the operation associated with Act10, and have not yet executed the operation associated with Act09. Contrary, in the case when only the proactive approach is used, the adaptation is only attempted when the execution process tries to invoke the operation associated with Act10 and realises that this operation is unavailable. In this scenario, the process requires the replacement of the operation associated with Act09 as well. However, when attempting to invoke the operation associated with Act10, the operation for Act09 has already been executed and cannot be replaced.

Figure 4.12 also shows that even when using the proactive approach only some instances are able to finish successfully for all the different numbers of execution instances used in the experiment. These instances are the ones that managed to invoke the operation associated with Act10 before this operation became unavailable and, therefore, were able to finish their execution successfully.

Study III.1 - Scenario 2: In this scenario, we use the same service composition of Scenario 1, but we consider different positions in the composition where the operation associated with an activity becomes unavailable. We consider the situations in which the operations associated with activities Act04, Act07, and Act10 become unavailable. In all three cases, we assume that a valid configuration exists when replacing both the operation that becomes unavailable and the one associated with the previous activity; i.e., operations associated with (i) Act03 and Act04; (ii) Act06 and Act07; and (iii) Act09 and Act10. We assume 200 instances of the service composition executed at the same time.

Figure 4.13 shows the results of the experiments for situations (i) to (iii)

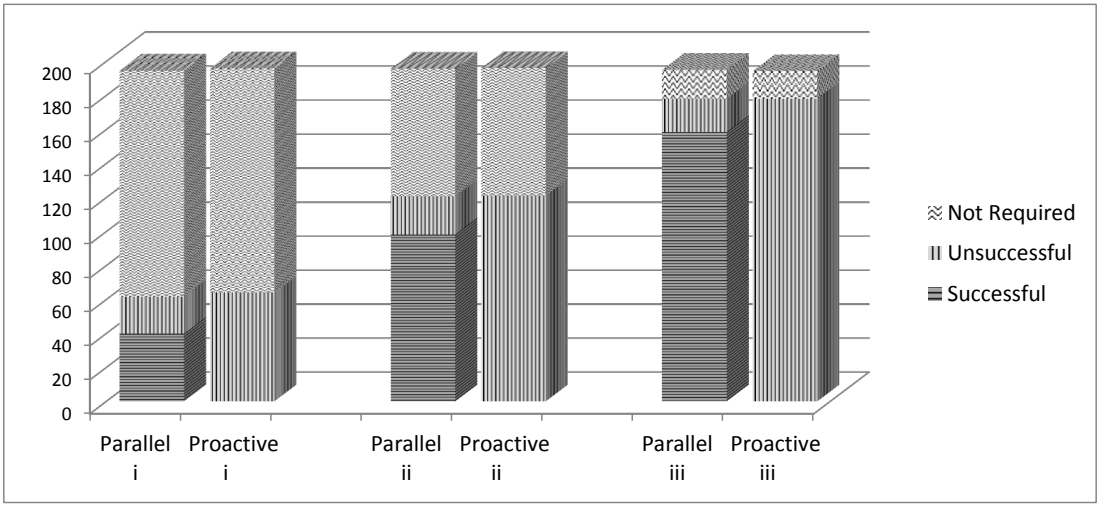

Figure 4.13: Comparison of the adaptation process for Case 1 - Scenario 2

above. As shown in the figure, when using only the proactive approach, in any of situations (i) to (iii), none of the execution instances could be successfully adapted. This is because the execution instances have already invoked the operations associated with the activities that occur before the activities that become unavailable (activities Act03, Act06, and Act09).

The results also show that the number of execution instances that do not require adaptation decreases when the problem occurs at a position closer to the end of the composition; while the number of unsuccessful adaptation instances increases. This is due to the number of running execution instances that are at a point *before*, *the same*, or *after* the point in which the problem is identified, during their execution. This also explains the reason for having similar numbers of execution instances that do not require adaptation, for the parallel and proactive approach and the proactive approach only, in situations (i) to (iii).

From Figure 4.13 we observe that when using the parallel and proactive approaches, the number of successful adaptation instances increases, as the problem

occurs at a position closer to the end of the composition. This is because the number of execution instances that can be adapted increases, since there are more instances at execution points before the operation becomes unavailable.

Study III.1 - Scenario 3: In this scenario, we compare the approaches when using service compositions with a sequential structure, as in Scenarios 1 and 2, but of different sizes. We considered compositions with (i') five, (ii') ten, and (iii') fifteen activities. Similar to the above scenarios, we assume that in each of the three compositions the operation associated with the last activity becomes unavailable and that a valid configuration exists when replacing both the operation that becomes unavailable and the one associated with the previous activity. We assume 200 instances of the service composition executed at the same time.

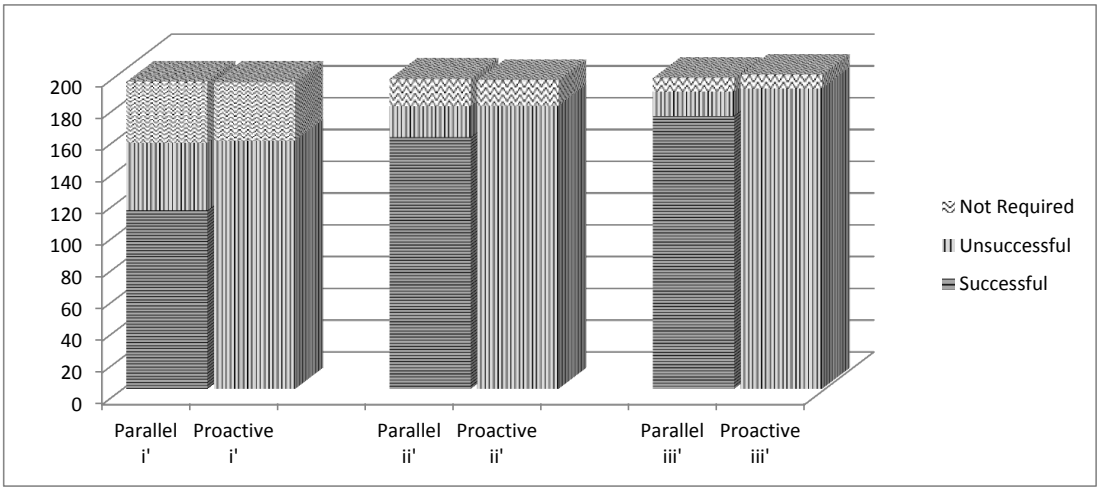

Figure 4.14: Comparison of the adaptation process for Case 1 - Scenario 3

Figure 4.14 shows the results of the experiments for compositions (i') to (iii'). The results in the figure show an increase in the number of execution instances

that required adaptation as the size of the compositions increase. As in the case of Scenario 2, this is due to the number of running execution instances that are at a point before, the same, or after the point in which the problem is identified, during their execution. Similarly, the results show an increase in the number of successfull adaptations for bigger compositions. Similar to Scenarios 1 and 2, the results show that when using only the proactive approach, in any of situations (i') to (iii'), none of the execution instances could be successfully adapted.

Study III.2

In this study we use the service composition shown in Figure 4.15. We consider that the operations associated with activities Act04, Act15, and Act19 becomes unavailable, and that, for each unavailable operation, the solution of a valid configuration exists when replacing the operations associated with the previous and next activities and the one that becomes unavailable. We assume 100 instances of the service composition executed at the same time.

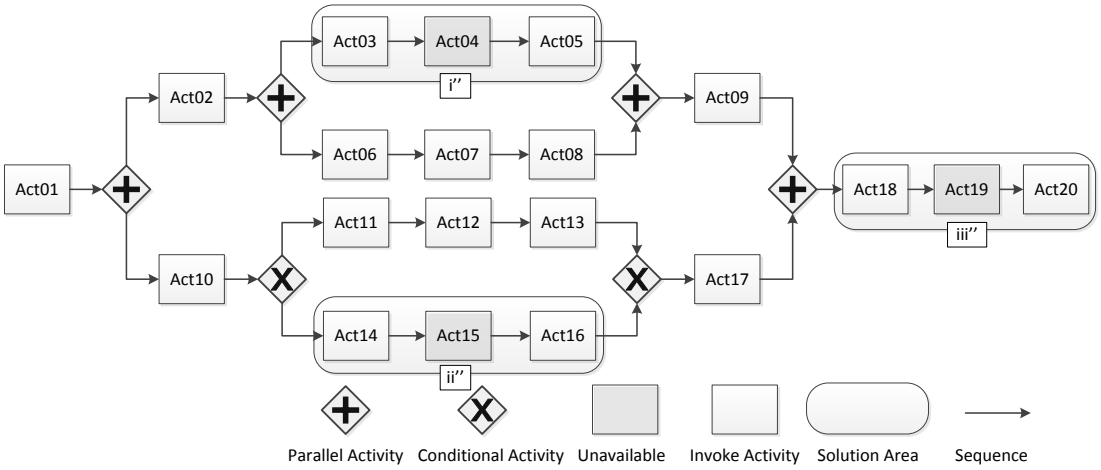

Figure 4.15: Complex service composition workflow

The example in Case 2 differs from the scenarios in Case 1 since: (a) the service composition is more complex with more activities organised in different execution logics (conditional and parallel); (b) a valid configuration for the composition includes the replacement of operations associated with activities before and after the operation that becomes unavailable; and (c) the operations that become unavailable are associated with activities in different execution logics: sequential, parallel, and conditional execution logics.

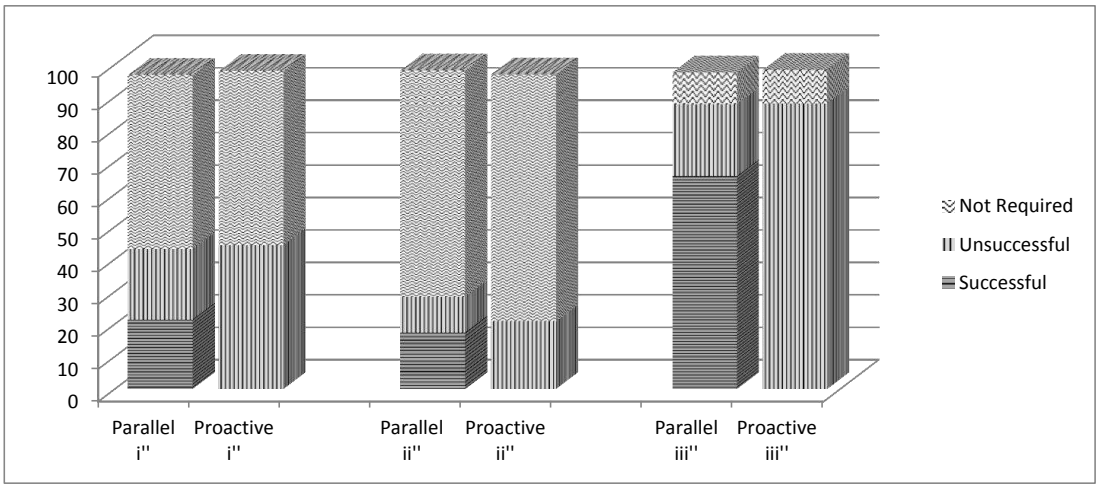

Figure 4.16: Comparison of the adaptation process for Case 2

The results of this experiment are shown in Figure 4.16 for the unavailability of (i") Act04, (ii") Act15, and (iii") Act19. As it was expected, when the operation that becomes unavailable is at the end of the composition (situation (iii")), a larger number of execution instances require adaptation since there are more running instances at execution points before the operation becomes unavailable (as in the previous scenarios). The results show that for situation (i"), half of the execution instances did not require adaptation. For those execution instances that required adaptation, half of them were successfully adapted.

We also observe that situation (i") required more instances to be adapted than situation (ii"). This is due to the fact that situation (ii") is a conditional execution logic and, therefore, not necessarily all the execution instances will execute this path in the composition. This is not the case in situation (i") in which all the instances need to execute the respective path in the composition.

Study III.3

In our approach, the efficiency of the proposed technique to dynamically distribute the load of service operations requests among different service providers, and in parallel with the execution of service composition instances, depends on the number of requests for a particular service operation and the capacity of the service operation to fulfil its requests. The size, complexity, and logic of a service composition does not cause impact to the load balancing technique. Therefore, in order to evaluate the load balancing technique, we used a simple service composition. More specifically, the evaluation was executed in a scenario with a single invoke activity (IA) deployed in two operations given by two different providers P_1 as OP_1, and P_2 as OP_2. We assumed both OP_1 and OP_2 configured with a processing time of one second. Moreover, in the experiment we used a maximum of 20 concurrent service composition instances and configured OP_1 and OP_2 to be able to handle up to ten concurrent requests.

In order to introduce some random behaviour in the income rate of operation requests, we simulated the compositions requests and assumed that each request respects a uniform distribution with minimum zero and maximum one. In other words, each of the 20 parallel processes generating concurrent requests sleeps for a specific amount of time and generates a new request. After that, the process starts

again if the experiment is not over. This behaviour is depicted in Figure 4.17

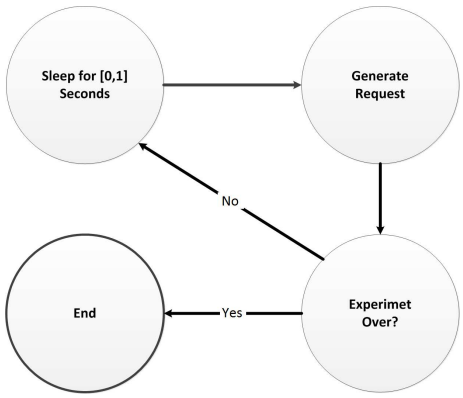

Figure 4.17: State Machine of the Concurrent Requests Generator Process

In the above described experiment we expected to observe an improvement in the overall performance of the execution engine in terms of the number of successfully concluded compositions requests. The basic idea is that if no distribution of the load is in place, the best thing that an approach can do is to jump from one operation to another as soon as it is detected as unavailable (e.g., an operation that is not responding due to high traffic). Moreover, considering that no single operation is suitable to answer all concurrent instances, it is almost mandatory to employ some form of online testing to discover if the operation becomes available again. Without a way to assess the availability of an operation, the composition instances would just fail to be created since the system would indicate that no operations is available to perform the required tasks. We implemented a basic online testing procedure that periodically checks if the previously failed operation is available and marks it in the local repository as available again.

In our experiments using the parallel adaptation with load balancing techniques, the average time to finish an execution instances was about one second. This was expected since the processing time of both OP_1 and OP_2 were config-

ured as one second. Moreover, there was only at most 20 concurrent requests, and the combined throughput for OP_1 and OP_2 was 20. Therefore, no extra issues were introduced. We noted that in the case in which the ability to distribute the load between different operations was turned off, the average time to conclude an execution instance rose to about 3 seconds. This was due to the fact that now the Adaptor component had to constantly face an error due to the high load of requests in either OP_1 or OP_2. Figure 4.18 presents a snapshot of the distribution of the requests made to OP_1 in both experiments. As we can see, when the load balancing technique is in place, the load distribution is much more homogeneous. The black line at 10 concurrent requests indicates the threshold for the serving capability of OP_1. Given that the approach with load balancing respects the threshold for individual operations by identifying its maximum capability, no issues are introduced. However, when such awareness is removed, and there is no way to alleviate the load, the threshold is not respected. This causes errors, requiring that adaptations are executed.

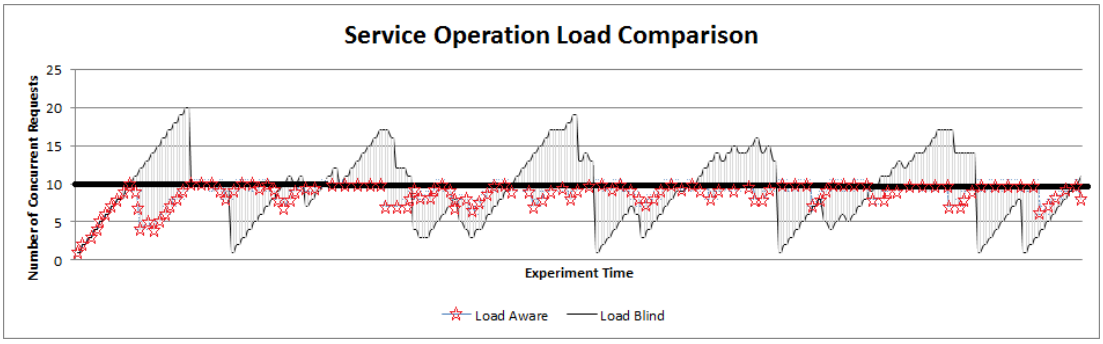

Figure 4.18: Comparisons of the distribution of operation request for a single provider between the approach with and without load balancing.

4.4 Summary and Discussions

This chapter presented the prototypes of the ProAdapt framework proposed that were developed to demonstrate and evaluate the ProAdapt framework. More specifically, the chapter describes the process undertaken to verify the concepts, techniques, and methods of the adaptation framework through three prototypes of the framework that were implemented in an iterative way.

For the first prototype, we were mainly concerned with the ability to predict the need for changes in service compositions and adapt this composition accordingly, without stopping the business process. We have show in this chapter that it was possible to predict the availability of service operations by using a spatial correlation technique. Moreover, the QoS analysis (see Section 3.3.1.1) could be successfully used to enforce the SLA values for the whole composition. The benefits came with little overhead, since the additional time required to monitor and adapt the execution instances was show to be very small, with little impact on the overall response time for the composition as a whole.

With the results of the first prototype, we knew that the proactive adaptation could be used in practice, that is, we could predict the need for changes and enact these changes in parallel of the execution of the service composition.

This chapter also describes the second prototype (*Prototype II*), which expanded our first prototype with the ability to perform structural changes in the workflow, and supported the four different classes of situations that may trigger the need for adaptation (see Chapter 3). We have provided for Prototype II a more deep discussion of the performance of the framework in terms of the duration of each sub activity of the adaptation process. After carefully analysing the

results, we came to the conclusion that ProAdapt could be used without causing considerable performance penalties to the execution of the composition. This was due to the fact that the required time for the adaptation process is small when considering all other times involved in the service composition execution.

Finally, this chapter also presents the description and results concerning the third prototype (*Prototype III*), which extended *Prototype I* and *Prototype II* to support parallel adaptation of service composition (see Section 3.3.1.3), and the load balancing technique (see Section 3.3.1.4). We have shown that by using these techniques the adaptation process could be improved, in terms of the number of service compositions that can adapt successfully.

The next chapter presents a different branch of the ProAdapt framework. An extension that targets the ability to compensate the expected behaviour of service compositions by looking at the local repository of service operations and iteratively querying a service discovery tool in order to find candidate replacement services.

Chapter 5

Behavioural Compensation Extension

Existing approaches for service composition assume that deployed services can be replaced by other candidates that fulfil the structural and behaviour aspects of the original service. However, it is not always possible to identify a service, or even a group of services, that can fully satisfy the behaviour characteristics of another service and, therefore, it is necessary to replace a service with the "best" possible candidate and try to compensate the "missing" behaviour by other candidates in the composition, or by identifying new services.

This section describes an extension of ProAdapt to support the above situation, which we we call *behavioural compensation*. In this extension, queries representing the characteristics of the services to be identified are generated based on the expected behaviour of current deployed services and the service composition workflow.

Services are identified based on matching of the structural and behavioural

aspects of the query and service descriptions. The behavioural compensation is executed by decomposing the queries to identify services that together can fulfil the requested behaviour.

The generated queries are used to identify services in service repositories that can fulfil the queries and be used in the composition. The identification of services is executed by an extension of the service discovery tool presented in [147].

More specifically, the service discovery tool matches structural (expected interfaces of a service) and behavioural (expected functionality of a service) aspects of queries with structural and behavioural service descriptions based on the computation of *distance measurements*. The distance measurements indicate how similar are the descriptions of a service with respect to a query, based on the parts of the queries that are matched with the service descriptions.

The version of the service discovery tool in [147] identifies a single service that fulfils the query, when such service exists. We have extended the work to support the identification of more than one service that *together* can fulfil the query. The new extension supports the situation in which a service that fulfils part of the query is identified and the rest of the query not fulfilled by the identified service is used to identify other services that match the rest of the query. These other services can either be external services from the composition (i.e., services that are not yet deployed in the composition), or services that are already used in the composition.

As mentioned in the previous chapters, ProAdapt was designed to support stateless services. This extension adds supports for additional constraints specified for the service composition, such as stateful services. The behavioural compensation extension can be incorporated in the ProAdapt architecture as an extra

attempt of adaptation to be performed inside the adaptor component. If the standard adaptation approach fails, the behavioural compensation can be activated.

Assumptions and Decisions

- **Service Specification:** In the framework, a service is assumed to be specified by a set of XML facets representing different service aspects, including structural facets describing the operations of services with their signatures in WSDL, and behavioural facets describing behavioural models of services in WS-BPEL.

- **No Instance Adaptation:** Due to the complexity of the behavioural compensation approach, we have decide to not consider its use for instance level adaptation. More specifically, after considering the computational cost of the approach we decided that it was better to use the solution to adapt service compositions and their templates, but not single execution instances.

- **Service Discovering Time:** For this extension we do not analyse the time that it takes for the service discovery tool to perform the matching since this depends on the number of services in a repository and such analysis has already been published in [147].

5.1 Running Example

In other to illustrate our approach, we use a running example of a *Car Rental Service (CRS)* application. This example is an extension of the CRS application described in [17]. Figure 5.1 shows an overview of the CRS application in

BPMN notation [30]. As shown in the figure, the example consists of a service composition that is executed at a car rental office branch. The service composition interacts with six different types of services with their descriptions and operations below, without their parameters for simplicity.

S_1 Car Broker Service (CBS). Operations: startRental, stopRental.

S_2 User Interaction Service (UIS). Operations: findCar, findcarCB.

S_3 Car Information Service (CIS). Operations: lookupCar, markAvailable, markUnavailable.

S_4 Car Parking Sensor Service (CPSS). Operations: carEnter, carExit.

S_5 Check Car Service (CCS). Operations: checkforDamage, checkforPetrol, sendtoRepair.

S_6 Payment Service (PS) Operations: calcpetrolBill, calcrepairBill, calctotalBill, chargecreditCard.

The *Car Broker Service* (CBS) controls the operation of the branch. The *User Interaction Service* (UIS) allows customers to request for a car. The *Car Information Service* (CIS) maintains a database of available cars and allocates cars to customers. The *Car Parking Sensor Service* (CPSS) senses when a car enters or exits the parking lot of the branch. The *Check Car Service* (CCS) verifies the status of the car in terms of any damage with the car and the level of petrol in the tank of a car when the car in returned. The *Payment Service* (PS) calculates the different types of extra expenses for the car and charges the credit card of the user.

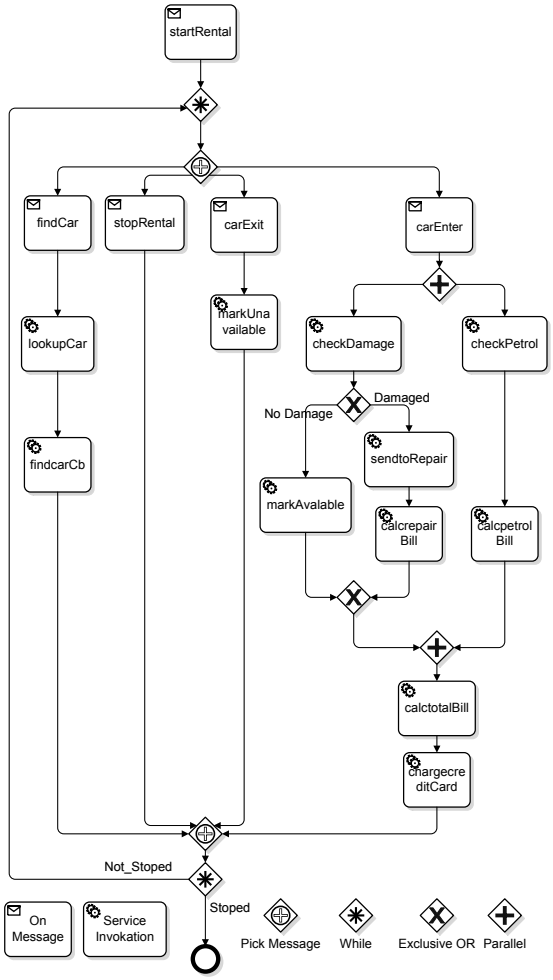

Figure 5.1: Example of Car Rental Service

As shown in Figure 5.1, the CRS composition starts when it receives a *startRental* message. After this the composition enters an infinite loop until it receives a *stopRental* message. In the loop, the composition can receive a *findcar*, *stopRental*, *carEnter* or *carExit* message. For each of these messages the composition takes a different flow of execution and invoke different messages.

For example, in the case of a *findcar* message, the composition invokes the *lookupCar* operation which returns either a negative value or an identifier for

an available car. This information is passed to the *findcarCB* operation to be informed to the user. In the case of a *carExit* message, the composition invokes *markUnavailable* operation in order to update the database about available cars.

In the case of a *carEnter* message, the composition performs a parallel execution flow and invokes the *checkforDamage* and the *checkforPetrol* operations. In the situation in which the car is not damaged, the *markAvailable* operation is invoked and the database of cars is updated. In the situation in which the car is damaged, the operation *sendtoRepair* is invoked and the operation to calculate the bill for repairing the car (*calrepairBill*) is invoked.

After operation *checktoPetrol* is invoked, operation *calcpetrolBill* is also invoked to calculate how much the user needs to pay for petrol in case the fuel tank of the car is not fulled when the car is returned. After executing the parallel flow above, the composition invokes the *calctotalBill* operation to calculate how much in total the user still needs to pay and invokes the *chargecreditCard* operation.

5.2 Overview

Figure 5.2 shows an overview of the behavioural compensation extension. As shown in the figure, the framework receives as inputs: a *workflow* (WF) representing the service composition or an application to be developed based on the composition of services (e.g. the CRS application), and a set of service operations that need to be replaced.

The first step is to generate a query to represent the operations that need to be replaced in a service composition as well as the order in which they need to be invoked. The generated query is used by the *query-based service selection*

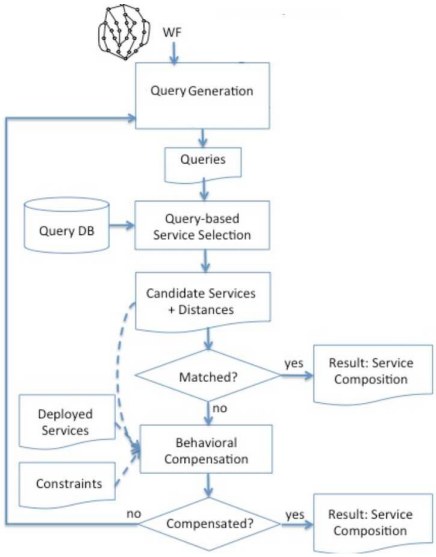

Figure 5.2: Service Selection with Behavioural Compensation

component, which interacts with a service repository to identify services that together can fulfil the queries, based on the match of query specifications and structural and behavioural service descriptions.

The structural descriptions are interface descriptions of the operations of services with their data types using WSDL [32]; while the behavioural descriptions represent behavioural models of services describing the functionality of the services using BPEL4WS [73].

The result of the matching process is a set of candidate services that "best" match the queries; i.e., the services that match larger portions of the characteristics of the queries. These characteristics include different aspects of the workflow and the services that can be used in the workflow. They include structural (aka syntactical interface) and behavioural flows representing expected functionality from the services.

For each query, when there is an identified candidate service S that fully

matches the query, S is bound to the service composition. In the case in which no service in the repository fully matches the query, the *behavioural compensator* component analyses if the identified candidate services, or any of the already deployed services in the composition, can be used to fulfil the "missing" behaviour of the query (i.e., the behaviour parts of the query that is not matched by a service), together with the associated structural parts of the query. The behavioural compensator component also uses extra pre-defined constraints during the analysis. Examples of these constraints are concerned with: (i) operations that need to be used from a particular service (due to previous agreement or contracts between the service providers and consumers), (ii) a group of operations that need to be provided by a certain service, or even (iii) contextual or quality constraints concerned with the operations (the time or cost to execute an operation).

In the case in which the "missing" behaviour of the query can be matched by other services that were already discovered, and are already present in the workflow, these services are used and bound to the service composition. Otherwise, new queries need to be generated for service selection with the repository. Note that we do not need to generate new queries based on the workflow, but we only need to change previously generated queries to take into account the operations that are still missing.

5.3 Query-based Service Selection

For each query Q in the set of queries, the framework tries to identify a service, or a set of services, that together can fulfil the structural and behavioural characteristics of the query. As explained above, we have extended the service

discovery tool [147] to support not only structural and behavioural matching between queries and services, but to also support *behaviour compensation*.

Matching between queries and services is executed based on ranking of services due to computation of structural and behavioural distances between queries and services. The distances capture the similarity of the services with respect to the queries. The structural distance captures the linguistic similarities between the names of the operations and the names of their respective parameter data types, and the similarities between the structure of the data types of the parameters.

For our case study, this distance is said to be zero, and therefore, the service found can be used as a candidate, when the functionality expressed in the queries can be verified against the service aspects. If the query violates any of the defined aspects and its constraints, the service is not considered as a candidate.

During the matching process, the structural and behavioural parts of a query are evaluated against the structural and behavioural specifications of the services and a distance between each of these services and the query is computed. More specifically, the structural and behavioural evaluation of a query against a service is executed by comparing operations in the structural part of a query with operations in the WSDL specifications of the services, and by comparing behavioural part of a query with WS-BPEL specifications of the services to which the structural parts are matched.

Matching: A match between a service and a query is found if for each operation (Opi) in the query, the service has an operation which has the same name as Opi, input parameters with data types that are supertypes of the types of the input parameters of Opi, and output parameters whose types are subtypes of the output of Opi. This is because when the above conditions hold, the input

information necessary to invoke *Opi* covers the input information needed by the service operation, and the output information produced by the service operation covers the output information expected by *Opi*. Furthermore, the service must have a behavioural model satisfying at least part of the behavioural conditions of a query.

Selection: When the above conditions are satisfied by a service S, a distance between the query and the service is computed. This distance is used to rank the service with respect to other services that satisfy the above conditions, and the framework selects the service with the smaller distance for the query.

Structural distance: The structural distance between query and service operations is calculated by matching the signatures of service and query operations, based on the comparison of the names of the operations, graphs representing the data types of the input and output parameters of the operations, and names of operations and parameters. The comparison of the input and output parameter data type graphs is based on an algorithm that we have developed to detect morphisms between service graphs, and to verify if a graph is a subgraph of another. A detailed description of the algorithm and distances can be found in [147].

The structural distances are computed for each possible pair of an operation in a query Q and an operation in a service S. After computing the structural distance for each pair of query and service operations, the matching process identifies all possible mappings between the operations in Q and operations in S, in which each operation in Q is mapped onto a single operation in S. For each identified mapping, the matching process computes the behavioural distance between the mapped service and query operations.

Behavioural distance: The behavioural distance is calculated based on compar-

isons of paths representing the behavioural in a query (or conjunction of queries) and behavioural service specification. More specifically, the behaviour matching is executed by transforming BPEL4WS [73] behavioural specifications of each service into a state machine and the behavioural part of the query into another state machine, and verifying if each path in the state machine of the query can be matched with a path in the state machine of the service. The path verification analyses if the query path can be executed by the state machine of the service. Transformation of BPEL4WS specifications into state machines is described in [147].

As described above, it is possible to identify services that cannot fulfil the whole structural and behavioural characteristics of a query. In this case, it is necessary to compensate the "missing" parts of the query by identifying other services that can fulfil these missing parts. The next section describes an approach to deal with this situation.

5.4 Behavioural Compensation

In set theory, a set A is said to be a subset of B if all elements of A are also elements of B. In this context, a partition of a set B is a division of B as a union of non-overlapping and non-empty subsets of B. In other words, a partition of B is a set X of subsets of B such that there is no intersection between the elements of X and the union of all elements in X equals to B. The definition above is the core to our solution for behavioural compensation. If a query Q is not fully matched by a service, we decompose it into all its possible sub-queries, that is. Considering Q a set o elements, the sub-queries can be defined as partitions of Q.

We then systematically explore each decomposition to find services that match the respective sub-queries. We first check the services that are already bound to the workflow (WF) and if a match is not found we check the repository as explained in the section above.

Each query Q is partitioned into a list of all possible *sub-queries* of Q that could satisfy structural and behavioural constraints represented in Q.

This can be represented in the following notation:

1. $\emptyset \notin D$
2. $\bigcup D = Q$
3. $(A \in D \wedge B \in D \wedge A \neq B) \Rightarrow A \cap B = \emptyset$

Where D is a partition of Q, and $\bigcup D$ is the union of all elements in D.

For example, assume $Q = \{S_1.OP_1 \rightarrow S_1.OP_2 \rightarrow S_1.OP_3\}$. The approach creates the following D_1 to D_5 sets of sub-queries:

$$D_1 = \{\{Op1 \rightarrow Op2 \rightarrow Op3\}\}$$

$$D_2 = \{\{Op1\}, \{Op2 \rightarrow Op3\}\}$$

$$D_3 = \{\{Op1 \rightarrow OP2\}, \{Op3\}\}$$

$$D_4 = \{\{Op1 \rightarrow Op3\}, \{Op2\}\}$$

$$D_5 = \{\{Op1\}, \{Op2\}, \{Op3\}\}$$

The total number of partitions of an n-element set (i.e. the number of sub-queries of a query) is the Bell number B_n, which satisfies the recursion defined in

Equation 5.1. The first Bell number are $B_0 = 1$, $B_1 = 1$, $B_2 = 2$, $B_3 = 5$, $B_4 = 15$, $B_5 = 52$, $B_6 = 203$, $B_7 = 877$, $B_8 = 4140$, $B_9 = 21147$, and $B_{10} = 115975$.

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \tag{5.1}$$

For each of these sub-queries, the approach tries to identify services that can match a set of elements in the partitions that together include all operations in Q and do not have common operations (i.e. the same ActivityName:OpName in WF).

For sub-queries that do have common operations, we need to consider them together, using logical *and* operation, that is supported by our tool. The approach tries to identify services that can match distinct subsets of the operations in a query, starting with the full query, that contains all operations. For the example above, suppose that a candidate service S' that matches $\{Op1\}$ is identified. In this case, the other possible candidate services to fulfil Q can match either element $\{Op2 \rightarrow Op3\}$, or elements $\{Op2\}$ and $\{Op3\}$.

5.5 Matching and Compensation Cases

To follow the different possible matching between queries and services, and how these matches can be compensated, assume that a service S with the operations OPi ($1 \leq$ i ≤ 6) becomes unavailable. Consider query Q, with operations from service S, and service S_1 as a candidate service for S.

Q : `OP1->OP2->OP3->OP5->OP6`

C1: S_1 fully matches Q.

Action: Replace S by S_1 in the service composition.

C2: S_1 matches part of the signature of S, and for the part of the signature that is matched, S_1 fully matches the behaviour described in Q.

Actions:

1. Calculate the part of the query that is not matched (Q');
2. Check if there is another service S_2 that is already bound in the WF that could match Q'. If such S_2 exists, replace S by S_1 and S_2 in the composition. Otherwise, invoke the service selection component with a new query with Q'.

C3: S_1 matches part of the signature of S, and S_1 matches part of the behaviour in Q for the part of the signature that is matched. In this case, the behaviour of S_1 can fall in one of the following examples, with respect to query Q:

a. `S.OP1->S.Op3`

b. `S.Op1->S.Op3->S.OP4`

c. `S.Op1->S.Op4->S.Op3` (S has extra operations in the middle of the logical flow)

d. `S.Op4->S.Op1->S.Op3` (S has extra operations in the beginning of the logical flow)

e. `S.Op1->S.Op6->S.Op3`

Actions:

1. If the behaviour of the operations in S appear in the correct order when not considering the missig or additional parts of the signature in the query (examples a, b, c, d), proceed to step 2; otherwise this is not a valid candidate (example e).
2. If there are no other additional operations in the behaviour of S (example a), proceed as in case C2; otherwise proceed to step 3 (examples b,c,d).
3. If the additional operations in the bahaviour of S, which are not in the query, can be found in the WF (i.e., the operations are supported in WF by other services), and if these additional operations in S satisfy the properties and constraints of WF (example b), proceed as in case C2; otherwise, this is not a valid candidate (examples c, d, e).

C4: S_1 fully matches the signature of S and S_1 matches only part of the behaviour for that signature.

Action: This case will be treated as cases C2 or C3, depending on the behaviour match. Note that the fact that S fully matches the signature of S_1, does not change on how to deal with the situation.

C5: There are no candidate services.

Action: In this case the execution of the service composition cannot be continued, unless a service provider creates a service with the necessary functionality.

The above matching and compensation cases are supported by the notion of decomposition of a query into a list of partition described in Subsection 5.3.

Example: Consider query $Q = \{findCar \rightarrow lookupCar \rightarrow findcarCB\}$ of the Car Rental Service (CRS) and the query presented below. The decompositions of query Q into sub-queries are:

$$D_1 = \{\{findCar \rightarrow lookupCar \rightarrow findcarCB\}\}$$

$$D_2 = \{\{findCar\}, \{lookupCar \rightarrow findcarCB\}\}$$

$$D_3 = \{\{findCar \rightarrow lookupCar\}, \{findcarCB\}\}$$

$$D_4 = \{\{findCar \rightarrow findcarCB\}, \{lookupCar\}\}$$

$$D_5 = \{\{findCar\}, \{lookupCar\}, \{findcarCB\}\}$$

The approach tries to identify a service that can be matched with D_1. If such service does not exist, the approach will try to identify services that can be matched with the two elements in D_2. If these services are not found, the approach carries on trying to identify services that can be matched with the elements of the other decompositions. In the case in which a service that matches certain elements of a decomposition is identified, the approach tries to identify services

that match elements in other decompositions that together include all operations in the query.

For the repository above, service S_1 will be matched with set D_1. However, suppose that we have a repository with services S_2, S_3, S_4, and S_5 above, but not with service S_1. In this case, the elements of set D_2 will be matched with services S_3 and S_5, respectively. Another possibility is the match of elements in decomposition D_4 with services S_2 and S_4, respectively.

For another example, suppose that we have a repository with services S_3, S_4, and S_7 above. In this case, the first element of D_2 {findCar} will be matched with service S_3, the second element of D_4 {lookupCar} will be matched with service S_4, and the third element of D_5 {findcarCB} will be matched with service S_7. If we consider the situation in which we have service S_5 and S_6 in the repository, a possible match will be a match between the first element of P_2 {findCar} with service S_6; and the match of the second element of P_2 with either service S_5 or service S_6.

Feasability

The decomposition and matching processes can be very expensive, given that the number of sub-queries grows fast in response to the size of the query. The two factors that impact its complexity and computational cost are generation of sub-queries and the interactions with the service discovering tool.

Decompositions for the composition as a whole, logic regions, and services, can be computed in parallel of the execution of the compositions but require constant consumption of system memory. Thus, the generation of sub-queries does not affect much the feasability of the approach.

The actual maximum number of interactions with the discovering tool how-

ever, depends on the number of distinct sub-sets of a query. Formally speaking, the maximum number of attempts to compensate through decomposition is given by the number of subsets of the power set of query minus one. In other words, given a query with n elements, the maximum number of attempts would be $2^n - 1$.

In our experiments, the actual number of interactions is roughly two thirds of this maximum, which is related to the order in which sub-queries are attempted and the available repository of candidate services. Considering that the communication with the service discovering tool is expensive, this is the major concern regarding deploying out approach in a real setting.

5.6 Proof of Concept

We have implemented a prototype tool of the framework. The query generation uses SPIN tool for model checking. The service selection and behavioural compensation components have been implemented in Java as an extension of the service discovery tool described in [147]. The framework has been evaluated for the Car Rental Service application in terms of its use in two cases. We describe below these cases and the results of the evaluation.

Case 1: The use of the framework to identify services to be initially bound to service compositions based on a service workflow.

For this case we evaluated the use of the framework to identify services based on four different queries described below. The queries were chosen in such a way so that they represent different query sizes, they cover different flows of execution

in the model, including cases in which the flow of executions involve loops and do not involve loops.

Moreover, in the evaluation we consider two cases as described below:

Case 1.1: only the abstract operations are defined for the service composition, without information regarding the actual binding, that is, we do not know what are the services or provides to be used. This simulates the case where a designer knows what are the operations required but requires a discovering tool in order to found the actual services.

Case 1.2: the deployed services are know, and thus, we make separate queries per service.

In this evaluation we used a service repository with a mixture of services from the domain of the Car Rental Service application, including the services listed in Section 5.1 (services S_1 to S_6), and services from other domains.

$$Q_1$$

$$< startRental, findCar, lookupCar, findcarCB, stopRental >$$

$$Q_2$$

$$< startRental, findCar, lookupCar, findcarCB,$$
$$stopRental, carExit, markUnavailable, stopRental >$$

$$Q_3$$

$$< startRental, carEnter, checkforDamage,$$
$$markAvailable, checkforPetrol, calcpetrolBill,$$
$$calctotalBill, chargecreditCard, stopRental >$$

$$Q_4$$

$$< startRental, carEnter, checkforDamage,$$
$$sendtoRepair, calrepairBill, checkforPetrol,$$
$$calcpetrolBill, calctotalBill, chargecreditCard,$$
$$stopRental >$$

The results of the experiments for Case 1.1 are shown in Table 5.1. In the table, for each query Q_i described above we show: (i) the total number of decompositions of the query that were used to identify a solution, and the total number of decompositions generated for that query (column *Decompositions*); (ii) the total number of interactions with the service discovery component necessary to identify a solution, and the total number of distinct elements generated for all the decompositions (column *SD Interactions*); and (iii) the identified solution for

the query (column *Solution*). The total number of distinct elements generated for all the decompositions represent the maximum number of interactions with the service discovery component.

The number of interactions with the service discovery component is independent of the time that it takes for the service discovery component to execute the matching between a query (or decompositions of a query) and service specifications. As previously stated, we are not considering the time that it takes for the service discovery tool to perform the matching. We are interested in the number of interactions with the service discovery component given the focus of the work on behavioural compensation, and the concept of decompositions used to support this focus.

Table 5.1: Results of the experiments for Case 1.1

Query	Decompositions	SD Interactions	Solution
Q_1	35 / 52	19 / 31	Sl_1
Q_2	645 / 877	79 / 127	Sl_2
Q_3	16889 / 21147	352 / 511	Sl_3
Q_4	94026 / 115975	664 / 1023	Sl_4

Sl_1: $\{\{S_1.startRental, S_1.stopRental\},$
$\{S_2.findCar, S_2.findcarCB\},$
$\{S_3.lookupCar\}\}$

Sl_2: $\{S_1.startRental, S_1.stopRental\},$
$\{S_2.findCar, S_2.findcarCB\},$
$\{S_3.lookupCar, S_3.markUnavailable\},$
$\{S_4.carExit\}\}$

Sl_3: $\{S_1.startRental, S_1.stopRental\},$

$\{S_4.carEnter\}$,

$\{S_5.checkforDamage, S_5.checkforPetrol\}$,

$\{S_3.markAvailable\}$,

$\{S_6.calcpetrolBill, S_6.calctotalBill,$

$S_6.chargecreditCard\}\}$

Sl_4: $\{S_1.startRental, S_1.stopRental\}$,

$\{S_4.carEnter\}$,

$\{S_5.checkforDamage, S_5.sendtoRepair,$

$S_5.checkforPetrol\}$,

$\{S_6.calcrepairBill, S_6.calcpetrolBill,$

$S_6.calctotalBill, S_6.chargecreditCard\}\}$

In the example, services S_1 to S_6 identified for the queries in Case 1.1 are the ones described in Section III.

As shown in Table 5.1, although the number of decompositions generated for the bigger traces is large, the real number of interactions with the service discovery component is small. This is because the process only tries to identify each different element in the list of decompositions once.

Moreover, the size of the service repositories does not contribute to the number of interactions with the service discovery components. Furthermore, the order in which the decompositions and their elements are used during the identification of services influences the number of interactions with the service discovery component and possible solutions identified for a query.

Another aspect that influences the solutions identified for the queries and the number of interactions with the service discovery tool is the structure of the

state machines representing the behaviour of the services. For example, suppose services S_a and S_b with the same sets of operations Op_1, Op_2, Op_3.

Consider the sate machine for S_a with a path representing the operations in sequence, while the sate machine for S_b with each of the operations as transitions starting in the initial state. The behaviour represented in the state machine for S_a is more restrictive in terms of the queries that can be matched to it, when compared to the behaviour of the state machine for S_b.

However, in the case of a query representing the above three operations in sequence and a repository with services S_a and S_b, the process will require a single interaction with the service discovery tool, given that the process starts by trying the decompositions with a single element composed by all the three operations.

The results of the experiments for Case 1.2 are shown in Table 5.2. In the table, for each query Q_i we show: the queries generated by the projection of the queries on the respective services (column Sub-Queries); and, as in Case 1.1, the numbers of decompositions and interactions with the service discovery tool, and the identified solutions. In this case, the solutions Sl_1 to Sl_4 are the same as in Case 1.1, due to the use of the same repository.

Table 5.2: Results of the experiments for Case 1.2

Query	Sub-Queries	Decomposition	SD Interaction	Solution
Q_1	$Q_5Q_6Q_7$	3 / 5	3 / 7	Sl_1
Q_2	$Q_5Q_6Q_8Q_9$	4 / 7	4 / 10	Sl_2
Q_3	$Q_5Q_{10}Q_{11}Q_{12}Q_{13}$	5 / 11	5 / 15	Sl_3
Q_4	$Q_5Q_{10}Q_{14}Q_{15}$	4 / 23	4 / 26	Sl_4

The use of the operations of the various services described in Section 5.1 to split the query based on particular services generates a number of different

sub-queries for each original query. As show in Table 5.2, after the projections, three sub-queries were generated for Q_1, four sub-queries were generated for Q_2, five sub-queries were generated for Q_3, and four sub-queries were generated for Q_4. The projection of the four queries generated eleven distinct sub-queries in total, as shown in the list below. For each sub-query in the list, we present the maximum number of decompositions (D_m), and the maximum number of possible interactions with the service discovery (I_m).

$Q_5 = \{startRental \rightarrow stopRental\}\{D_m = 2, I_m = 3\}$

$Q_6 = \{findCar \rightarrow findcarCB\}\{D_m = 2, I_m = 3\}$

$Q_7 = \{lookupCar\}\{D_m = 1, I_m = 1\}$

$Q_8 = \{lookupCar \rightarrow markUnavailable\}\{D_m = 2, I_m = 3\}$

$Q_9 = \{carExit\}\{D_m = 1, I_m = 1\}$

$Q_{10} = \{carEnter\}\{D_m = 1, I_m = 1\}$

$Q_{11} = \{checkforDamage \rightarrow checkforPetrol\}\{D_m = 2, I_m = 3\}$

$Q_{12} = \{markAvailable\}\{D_m = 1 I_m = 1\}$

$Q_{13} = \{calpetrolBill \rightarrow calctotalBill \rightarrow chargecreditCard\}\{D_m = 5, I_m = 7\}$

$Q_{14} = \{checkforDamage \rightarrow sendtoRepair \rightarrow checkforPetrol\}\{D_m = 5, I_m = 7\}$

$Q_{15} = \{calctotalBill \rightarrow calcpetrolBill \rightarrow caltotalBill$
$\rightarrow chargecreditCard\}$
$\{D_m = 15, I_m = 15\}$

The difference in the number of distinct sub-queries with respect to the number of total sub-queries generated is due to the overlap in the services used to project the queries. On a real environment, if the queries are executed sequentially, the identical parts do not need to be checked again.

In Table 5.2, the total number of decompositions for each query is given by the sum of the number of possible decompositions for all sub-queries related to that query. Similarly, the total number of possible interactions with the service discovery component is given by the sum of the number of interactions for the associated sub-queries.

As shown in the table, for each query, the number of decompositions of the query that are used to identify a solution and the number of interactions with the service discovery component necessary to identify a solution are the same as the number of sub-queries for that trace. This is due to the fact that, for each query, a solution was identified for the first decomposition of that query that was checked against the repository. In all cases, the first decomposition has one element composed by a sequence of all the operations in the query. Moreover, the state machines of the services that were matched with the query accept a sequence of the respective operations.

Overall, the above experiments have shown that the number of interactions with the service discovery component in order to identify a match for queries depends on (i) the order in which the query decompositions and their elements are traversed, (ii) the structure of the state machines of the services representing the behaviour of these services, and (iii) the existence of a solution for the queries in the service repository.

Case 2: The identification of services to replace already deployed services in the composition that become unavailable or need to be changed.

For this case we assume that Payment Service (PS) (service S_6) in the Car Rental

Service application becomes unavailable and evaluate the use of the framework to identify services that can replace the Payment Service (PS). For this evaluation we used four different types of service repositories with different sets of candidate services for Payment Service. The different set of candidate services in the four repositories used in the experiment are shown below:

A: S_1; S_2; S_3; S_4; S_5; S'_6;

B: S_1; S_2; S_3; S_4; S_5; S_7; S_8;

C: S_1; S_2; S_3; S_4; S_5; S_7; S_9; S_{10};

D: S_1; S_2; S_3; S_4; S_5; S_9; S_{10}; S_{11}; S_{12};

Where services S_1 to S_5 are the same as the services described in Section III, and the other services are as follows: S'_6: a copy of S_6 in Section III; S_7: a service with operations *calcpetrolBill* and *calcrepairBill*; S_8: a service with operations *calctotalBill* and *chargecreditCard*; S_9: a service with operation *calctotalBill*; S_{10}: a service with operation *chargecreditCard*; S_{11}: a service with operation *calcpetrolBill*; and S_{12}: a service with operation *calcrepairBill*.

In these experiments, we consider the projection of the operations of service S_6 in the workflow of the service composition of the CRS application, which produces query $Q = \{calcrepairBill \rightarrow calcpetrolBill \rightarrow calctotalBill \rightarrow chargecreditCard\}$. For the set of repositories considered in this case, each set of candidate services present a single and distinct solution for the problem.

Table 5.3 presents the results of these experiments with the identified solutions described below. As shown in the table, for repository A, the solution was found in one interaction with the service discovery tool, while for the other repositories the

solutions were found in five, seven, and 15 interactions with the service discovery tool. This is due to the different types of services that were matched to the queries in the various repositories used.

Table 5.3: Results for Payment Service Compensation Experiment

Repository	Decompositions	SD Iteractions	Solution
A	1/15	1/15	Sl_1
B	4/15	5/15	Sl_2
C	5/15	7/15	Sl_3
D	15/15	15/15	Sl_4

Sl_1: $\{\{S'_6.calcrepairBill, S'_6.calcpetrolBill,$
$S'_6.calctotalBill, S'_6.chargecreditCard\}\}$

Sl_2: $\{\{S_7.calcrepairBill, S_7.calcpetrolBill\},$
$\{S_8.calctotalBill, S_8.chargecreditCard\}\}$

Sl_3: $\{\{S_7.calcrepairBill, S_7.calcpetrolBill\},$
$\{S_9.calctotalBill\}, \{S_{10}.chargecreditCard\}\}$

Sl_4: $\{\{S_{11}.calcrepairBill\}, \{S_{12}.calcpetrolBill\},$
$\{S_9.calctotalBill\}, \{S_{10}.chargecreditCard\}\}$

5.7 Summary and Discussions

This chapter presents an extension for ProAdapt that is based on the automated support for behavioural compensation in the cases when services cannot be found that can fully satisfy the required specifications of the original services. The chapter start with a brief explanation of the proposed extension and includes an example of a car rental service that we use to illustrate the work.

We then present an overview of the approach, describing the various steps and and expected flow of operation of the behavioural compensation approach. Since for this extension the service discovery tool and its internal working is so important, we describe the specific query-based service selection mechanism. More specifically, how queries are created and candidate services identified based on these queries.

Following the discussion regarding query-based service selection, we present the strategy used for the behavioural compensation, including a illustrative example of how it works. Then we present and discuss the five possible cases that may happen when attempting to find candidate services using the proposed approach. Finally, we show some experiments that we have conducted to evaluate the work.

As mentioned before, it is not always possible to identify a service, or even a group of services, that can fully satisfy the behaviour characteristics of another service. Some service require a specific order of messages and operations, and these constraints must be follow if one expect to interact with the service.

The results obtained show the feasibility of our approach in providing ways to identify if other services participating in the composition or available as candidates can be used to fulfil the expected behaviour of the service composition while respecting the constraints of the deployed services.

However, there are a few drawbacks in our approach. For instance, it terminates when the first possible solution is found. The solutions are attempted based on the order in which decompositions of queries are traversed. In the case of two or more possible solutions for a query there is no difference in the distance returned by the solutions and, therefore, it is not possible to say if one solution

is better than the other. However, if we expand the work to consider other constraints such as quality or contextual aspects of the services, it could be possible to have a solution that is better than another, depending on how these extra constraints are matched.

Moreover, the behavioural compensation logic does not scale well, since the number of partitions can increase really fast. If such logic is implemented at the service discovering tool side, however, the process becomes more smooth, since the knowledge of all available services can reduce the search space, saving resources and giving a faster answer.

Despite these restrictions, we believe that our proposed behavioural compensation approach is successful in its attempt to provide a solution for the identification of services based on matching of the structural and behavioural aspects of queries and service descriptions. In addition, the solution was implemented in a real testbed, which means less effort to adapt the solution for a final product.

The following chapter concludes the work presented in this thesis. We revisite the objectives and hypotheses and presentes what we expect for now on.

Chapter 6

Conclusions and Future Work

In this report we presented the ProAdapt framework as the result of the work performed in the area of adaptation of service compositions. ProAdapt is a QoS-aware, dynamic, and proactive adaptation framework for service composition that uses monitoring and prediction techniques to prevent possible execution failures and to improve service composition continually. Moreover, the proposed solution is able to identify individually the need for adaptation for each parallel running instance of a service composition, while avoiding unnecessary changes, and distributing load request among different service operations when necessary.

ProAdapt can also be seen as a hybrid adaptation approach, since it combines the ability to both react to events and predict the need for adaptation. Reactive adaptation, however, can also be seen as a special case of proactive adaptation where the impact of a monitored event is eminent, and thus require prompt reaction.

In what follows, we revisit the working hypotheses and objectives defined for this thesis report in Chapter 1, and provide a discussion of the obtained remarks

and how we manage to address each of the objectives.

General hypothesis

Proactive adaptation of service compositions can improve the performance, reliability, and general conformance with system requirements of service compositions when compared to traditional static and reactive adaptation approaches.

Sub-hypotheses

H.1 It is possible to proactively identify the need for adaptation in service compositions by constantly monitoring the execution environment, systems requirements, and the status of the service composition itself.

H.2 It is possible to use a local service repository and proactively replace candidate operations in service compositions.

H.3 It is possible to adapt service compositions in parallel to their execution without stopping the business process.

H.4 It is possible to avoid unnecessary changes in service compositions when identified problems can be compensated by parts of the service composition yet to be executed.

General Objective

To support a dynamic and proactive adaptation of service compositions through the use of monitors, candidate service replacements, and techniques for prediction of problems in order to improve the reliability, performance, and conformance of business process.

Objectives

O.1: Literature Review

To provide a literature review and analysis of works in relevant areas of the research topic. The review needs to include topics such as service composition, monitoring, discovery, adaptation, and failure prediction techniques.

O.2: Prediction of Problems

To design and implement mechanisms to support the detection and proactive prediction of events that may require adaptation of service compositions.

O.3: Events Analysis

To deliver techniques to help understanding potential faults and to identify the parts in a service composition that may be affected by detected or predicted faults.

O.4: Adaptation Approaches

To analyse and specify relationships between the different ways of adapting service compositions and the circumstances that may trigger adaptation.

O.5: Adaptation Framework

To specify an adaptation framework including techniques for monitoring, detection and prediction of events that may require adaptation, and enforcement of changes in a dynamic and proactive way.

O.6: Evaluation

To develop scenarios and evaluate proof-of-concept tools that will be created to support the techniques and mechanisms created for the Adaptation Framework considering medium to large scale case studies.

Hypothesis and Objectives Discussion

H.1 Since the first developed prototype we have been able to predict the need for adaptation using QoS analysis (Section 3.3.1.1) and spatial correlation (Section 3.3.1.2). These techniques were able to predict the need for adaptation before reaching the point of the execution instance where the prediction would turn into a fault. Moreover, the parallel analysis (Section 3.3.1.3) goes even further in identifying the need for adaptation as soon as possible, and thus augmenting the adaptation success rate.

H.2 Most of our research was focused on using a cacheable base of services, or the *Bind Information Repository* (see Figure 3.3), which could be externally updated in any way by different tools, such as the one used during the experiments [147]. This approach creates less of a problem to proactively adapt compositions due to the fact that service replacements are not found only when problems arise, but in parallel of the execution of service compositions. During the behavioural compensation extension (Chapter 5), however, the discovering tool is used more iteratively and on demand.

H.3 ProAdapt is able to go beyond the simple adaptation of service composition by being able to adapt single execution instances independently in parallel of other execution instances of the same or different service compositions (see Figure 3.2). The two major challenges to accomplish this behaviour was to extend the model representation of a running instance and modify the execution engine to support such new model.

H.4 The compensation performed within the execution instance without adaptation is possible for the third class of events covered by ProAdapt, that is, events that allow the composition to continue to be executed, but not necessarily in its best way. As discussed in Section 3.3.1.1, these events may not even cause a disruption at all if other parts of the composition are to *compensate* for the observed deviation. In fact, that is true not only for the parts of the service composition yet to be executed, as pointed out by fourth sub-hypotheses, but also by operations within the execution instance that outperformed their expected quality of service parameters, mainly the response time.

O.1 A literature review was made in order to identify the current knowledge related to the topic of the research. As a result a theoretical background was created to serve as conceptual base for the proposed approach, and a review of related works presented.

O.2 While studying the context of composite services, and more specifically the mechanics or infrastructure that supports the execution of service compositions, it becomes clear that predicting problems in service-based system is not an easy task. This is mainly because service-based systems are actually a system of systems which have no direct control of its internal components (other systems). In this context, this work identifies that the detection of problems in service compositions is concerned with deviations of the expected interaction model between the composition as a whole an its internal components. This reports includes two prediction techniques related to the core concept of service compositions, the interaction with service opera-

tions. The first technique focuses on the aggregation of QoS parameters of individual components of the composition, and the modelling of expected response time of operations to predict the probability of failure to execute the service composition in accordance with the SLA (see Section 3.3.1.1). The second technique, named spatial correlation, exploits the strong correlation between the availability of operations of the same service or provider to predict potential unreachable operations deployed in a composition.

O.3 As a result of the expected study to cover this objective, the current work is able to analyse specific observed or predicted events and assess the impact of these readings in the service composition. This work extends previous approaches by enabling the analyses to be centred in a specific execution instance, while at the same time creates the required means to expand the analyses across multiple and parallel execution instances.

O.4 While different strategies for adaptation of service composition can be identified, such as renegotiation with provider entities, or deployment of additional service agents, this work is so far centred at the rearrangement of service compositions, in terms of the internal logic and service components. In this context, the idealised approach is able to adapt execution instances individually, without affecting other instances at the same time that it is possible to reverberate the need for adaptation across multiple executing instances or even the ones yet to be created. The adaptation approach is backed up by a backtrack selection algorithm, which is in charge of computing a new set of deployed service operations for a specific execution instance.

O.5 As the main result of the research conducted, a new proactive adaptation framework (ProAdapt) was introduced. The framework supports techniques for locating and selecting service operations, monitoring, detection, and prediction of events that may required adaptation and enforcement of changes in a dynamic and proactive way.

O.6 We have considered a range of scenarios, ranging for simple service compositions for illustration purposes, real case scenarios, and complex compositions created for our evaluations. As observed in Chapter 4 three prototypes were implemented to support the proposed adaptation framework. These prototypes include tools for locating service candidates, monitoring, executing service compositions, and change the execution when necessary. Different evaluations were conducted including the use of simulated and real platforms.

6.1 Future Work

The experiments conducted during this work were useful to support our hypotheses and incrementally improve our framework. ProAdapt supports the dynamic adaptation of service compositions triggered by different classes of situations and is able to predict faults and failures caused by QoS deviation and unavailability of operations.

The results shows that ProAdapt, as well as its extension, can improve the overall quality aspects of running service compositions, however, we had to make certain assumptions to verify our solution and there are a few limitations. This section presents some ways in which ProAdapt could be further improved.

Support for User Interaction. Through the research conducted in this work it was noticed that existing approaches for service-based system adaptation do not necessarily consider interactions of the users with the services. However, several of the service-based applications require the user in the loop when service compositions are executed.

An extension of the ProAdapt framework is planned to support user interaction. More specifically, it is necessary to consider the time required by users to interact with the system and provide response to the system and the impact of this new activity in the process of QoS aggregation, prediction, and verification.

A first solution to consider the case of user interaction would be to ensure the SLA constraints for specific logic regions, and lessen such constraints for the composition as a whole. In such case, the SLA constraints would be ensured for regions before and after a defined user interaction.

It is important to note that the user interaction is not supported by default in implementations of BPEL4WS [73] execution engines. There are, however, some existing extensions, such as the activity *HumanTask* defined for IBM WebSphere [77].

Load Balancing Model. The use of the loading balancing technique in ProAdapt is successful in providing alternatives paths to the execution of service compositions under high loads. In order to avoid overloading services and receiving a degraded response, however, the strategy employed a solution that can limit the maximum throughput of service operations. In order to improve the maximum throughput of the composition as a whole while avoiding overloading the deployed operations, we are investigating the use of the congestion control algorithm used in the TCP protocol to dynamically adjust the expected throughput

of service operations.

In the TCP protocol the amount of data that can be sent is dynamically adjusted according to the sender and receiver capabilities. The main advantage of such approach is that it is able to couple with variations in both the network conditions as well as the current status of available internal resources, such as memory and processing power, of both sender and receiver.

Service Interface Adaptors. As discussed before, the matching process using in the service selection process takes into account both the syntax and semantic description of candidate services. Even with the great number of functionally equivalent services available in public providers nowadays, the interface of such services is not standardized. They may require different input parameters and produce different output values.

Things such as the names of the parameters, or their order, can usually be relaxed in an experimental setup, but for a final product even these small discrepancies on the interface of services can create a problem. An extension for this work is to include adaptors that would amend the difference in the interfaces of services that are semantically equivalent, in order to use them seamlessly in the ProAdapt approach.

Additional Constraints. The prototypes implemented for the ProAdapt framework consider basically the response time, cost, and throughput parameters of service operations. We can extend such considerations with other different types of constraints when attempting to adapt a service composition. By designing decision, some services or group of services may have to be fixed in a composition, which is not supported yet, but would require little effort to implement. Another example of constraints is the level of security or the quality of

the services as stated by some refutation system centred in users or consumers opinions. The problem of service selection with behavioural compensation, for instance, could be expressed as a constraint satisfaction problem.

Stateful Services. In order to handle more complex interactions with service based systems, our proposed framework would have to couple with stateful services. More precisely, stateful services keep some sort of state or information between the service calls, and replacing them within an execution instance may require additional steps, such as the rollback of operations already performed. Our behavioural compensation extension is able to couple with the restriction imposed by stateful services regarding the order that operations need to be invoked, but we consider a future improvement of our work the total support for stateful services, mainly regarding the roll back of operations and the implications in restabilising the correct execution of service operations.

6.2 Final Remarks

Our work support a dynamic, parallel, and proactive adaptation of service compositions through the use of monitors, candidate service replacements, and techniques for prediction of faults and failure in order to continually improve the reliability, performance, and conformance of business processes.

The research have identified various need for adaptation in service composition and ways to monitor or even predict events that may require adaptation. Techniques were presented to support the dynamic selection of candidate services and updating of compositions instances during run time, in parallel with the execution of the service based system.

One of the greater advantages of the framework presented in this work is the ability to adapt execution instances independently, which means that exceptional circumstances can be treated locally and isolated. Execution instances are created and maintained in such a way that it is virtually possible to have a different set of deployed service candidates for each execution instance.

The results collected through experiments performed with the various implemented prototypes are promising and show the advantage of the designed proactive approach. There are, however, a number of limitations or threats to validity, which may have an impacted our work. One of them is concerned with the fact that the services operations in the execution instances of the compositions are selected and replaced in our experiments based on a set of syntactically and semantically equivalent services.

A particular limitation is the assumption of stateless services when selecting and replacing candidate operations. Even though it is safe to assume that stateless services constitutes the majority of the available web services on public databases, due to its simplicity, complex interactions are usually implemented with stateful services. Considering stateful services, however, can be partially achieved with our behavioural compensation extension, as described previously.

Another possible threat to validity is related to the scenarios used during our experiments. Even though we used an illustrative real scenario to explain the approach, the evaluations were performed in artificial service compositions. This could be seen as a possible limitation to applicability; however, these scenarios were created to either study the solution in simply cases or stress it with complex situations in order to assess its efficiency. We believe that our choice for artificial compositions proves the validity of the approach for general cases, rather than

been specific to some examples.

One of the advantages of ProAdapt when comparing to other adaptation frameworks is the easiness in which the framework can be deployed in real setting using the developed prototypes. Many similar approaches provide sound theoretical frameworks but require a costly process to bring the ideas to the real world.

In our prototypes, we have implemented real components such as proxies and modified the ODE execution engine to work with real WS-BPEL specifications. This is not usually the case for the surveyed approaches. As our prototypes and evaluation scenarios were becoming more complex, however, we had to rely on our simulated infrastructure.

There are some challenges to be overcame in order to bring the whole framework into a real setting. This challenges are basically mostly to how the execution engine process the service compositions. For ProAdapt to work, service operation invocation must me accompanied by the information regarding the related execution engine and the point of execution inside such execution instance. Moreover, the execution engine must allow dynamic change of the executable service composition.

The growing number of service databases indicates that finding semantically equivalent operations services are not a major concern, however, these operations are usually presented with different interfaces. These interfaces are taken into account in our ProAdapt framework, but in order to improve its applicability further, some adaptor, as described in Chapter 2, could be considered.

The applicability of ProAdapt is not limited to standard web service compositions. Many concepts of the service-oriented computing can be used in fields

such as cloud computing and Internet of Things (IoT. The service composition for cloud services has recently gain attention an propose an alternative to improve cloud services collaboration and integration. In this scenario, adaptation approaches previously target at service compositions can be harnessed to devise smarter adaptation strategies to could services.

The same is valid for the IoT, which considers a pervasive presence of object, such as TVs, mobile phones, and sensors, which are able to cooperate among themselves to reach a common goal [56]. The concepts of service composition adaptation could be used to further improve the reliability of the applications in the context of the Internet of Things.

References

[1] Aggarwal, R., Verma, K., Miller, J., and Milnor, W. Constraint Driven Web Service Composition in METEOR-S. In *Proceedings of the 2004 IEEE International Conference on Services Computing* (Washington, DC, USA, 2004), SCC '04, IEEE Computer Society, pp. 23–30. 41

[2] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. *Web Services: Concepts, Architectures and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2010. 2, 6, 12

[3] Anselmi, J., Ardagna, D., and Cremonesi, P. A qos-based selection approach of autonomic grid services. In *Proceedings of the 2007 workshop on Service-oriented computing performance: aspects, issues, and approaches* (2007), ACM, pp. 1–8. 58

[4] Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., and Plebani, P. Paws: A framework for executing adaptive web-service processes. *IEEE Softw. 24*, 6 (Nov. 2007), 39–46. 42

[5] Ardagna, D., and Pernici, B. Adaptive service composition in flexible processes. *Software Engineering, IEEE Transactions on 33*, 6 (2007), 369–384. 59, 60

[6] Arshad, N., Heimbigner, D., and Wolf, A. L. A planning based approach to failure recovery in distributed systems. In *Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed Systems* (New York, NY, USA, 2004), WOSS '04, ACM, pp. 8–12. 43

[7] Åström, K. J., Hägglund, T., Hang, C. C., and Ho, W. K. Automatic tuning and adaptation for pid controllers-a survey. *Control Engineering Practice 1*, 4 (1993), 699–714. 30

[8] Autili, M., Benedetto, P., and Inverardi, P. Context-aware adaptive services: The plastic approach. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009* (Berlin, Heidelberg, 2009), FASE '09, Springer-Verlag, pp. 124–139. 40

[9] Bairavasundaram, L. N., Goodson, G. R., Pasupathy, S., and Schindler, J. An analysis of latent sector errors in disk drives. In *ACM SIGMETRICS Performance Evaluation Review* (2007), vol. 35, ACM, pp. 289–300. 36

[10] Baresi, L., Di Nitto, E., Ghezzi, C., and Guinea, S. A framework for the deployment of adaptable web service compositions. *Service Oriented Computing and Applications 1* (2007), 75–91. 10.1007/s11761-007-0004-1. 41

[11] Baresi, L., Ghezzi, C., and Guinea, S. Towards self-healing service compositions. In *PriSE04, First Conference on the Principles of Software Engineering* (2004). 42

[12] Bellwood, T., Capell, S., Clement, L., Colgrave, J., Dovey, M., Feygin, D., Kochman, A., Macias, P., Novotny, M., Paolucci, M., et al. Universal description, discovery and integration specification (uddi) 3.0. *Online: http://uddi. org/pubs/uddi-v3. 00-published-20020719. htm* (2002). 10

[13] Benbernou, S., Cavallaro, L., Hacid, M. S., Kazhamiakin, R., Kecskemeti, G., Poizat, J.-L., Silvestri, F., Uhlig, M., and Wetzstein, B. State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs, 2008. 37

[14] Benedetto, D. P. A framework for context aware adaptable software applications and services, July 2010. 40

[15] Berbner, R., Spahn, M., Repp, N., Heckmann, O., and Steinmetz, R. Heuristics for qos-aware web service composition. In *Proceedings of the IEEE International Conference on Web Services* (Washington, DC, USA, 2006), ICWS '06, IEEE Computer Society, pp. 72–82. 41

[16] Bhaskaran, K., and Schmidt, M.-T. Websphere business integration: An architectural overview. *IBM Systems Journal 43*, 2 (2004), 238–254. 40

[17] Bianculli, D., Giannakopoulou, D., and Pasareanu, C. S. Interface decomposition for service compositions. In *ICSE* (2011), pp. 501–510. 160

[18] Bodenstaff, L., Wombacher, A., Reichert, M., and Jaeger, M. C. Analyzing impact factors on composite services. In *Proceedings of the 2009 IEEE International Conference on Services Computing* (Washington, DC, USA, 2009), SCC '09, IEEE Computer Society, pp. 218–226. 51

[19] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. Web Services Architecture, W3C Working Group Note. Tech. rep., W3C Working Group, February 2004. 2, 6, 7, 9

[20] Bordeaux, L., Salaün, G., Berardi, D., and Mecella, M. When are two web services compatible? In *Proceedings of the 5th international conference on Technologies for E-Services* (Berlin, Heidelberg, 2005), TES'04, Springer-Verlag, pp. 15–28. 47

[21] Bucchiarone, A., Lafuente, A., Marconi, A., and Pistore, M. A formalisation of adaptable pervasive flows. In *Web Services and Formal Methods*, C. Laneve and J. Su, Eds., vol. 6194 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 61–75. 42

[22] Bucchiarone, A., Pistore, M., Raik, H., and Kazhamiakin, R. Adaptation of service-based business processes by context-aware replanning. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on* (Dec 2011), pp. 1–8. 42

[23] Burstein, M. H., Hobbs, J. R., Lassila, O., Martin, D., McDermott, D. V., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T. R., and Sycara, K. P. Daml-s: Web service description for the semantic web. In *Proceedings of the First International Semantic Web Conference on The Semantic Web* (London, UK, UK, 2002), ISWC '02, Springer-Verlag, pp. 348–363. 11

[24] Canfora, G., Di Penta, M., Esposito, R., and Villani, M. L. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (2005), ACM, pp. 1069–1075. 59, 60

[25] Canfora, G., Penta, M. D., Esposito, R., and Villani, M. L. Qos-aware replanning of composite web services. In *ICWS* (2005), IEEE Computer Society, pp. 121–129. 41

[26] Casati, F., Ceri, S., Pernici, B., and Pozzi, G. Workflow evolution. *Data & Knowledge Engineering 24*, 3 (1998), 211–238. 70

[27] Casati, F., Ilnicki, S., Jin, L.-j., Krishnamoorthy, V., and Shan, M.-C. Adaptive and dynamic service composition in eflow. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering* (London, UK, UK, 2000), CAiSE '00, Springer-Verlag, pp. 13–31. 41

[28] Chen, M., Tan, T., Sun, J., Liu, Y., Pang, J., and Li, X. Verification of functional and non-functional requirements of web service composition. In *Formal Methods and Software Engineering* (2013), L. Groves

and J. Sun, Eds., vol. 8144 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 313–328. 15

[29] Cheng, B. H., De Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., et al. Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems*. Springer, 2009, pp. 1–26. 30

[30] Chinosi, M., and Trombetta, A. Bpmn: An introduction to the standard. *Comput. Stand. Interfaces 34*, 1 (Jan. 2012), 124–134. 13, 161

[31] Chiu, D., Deshpande, S., Agrawal, G., and Li, R. A dynamic approach toward qos-aware service workflow composition. In *Proceedings of the 2009 IEEE International Conference on Web Services* (Washington, DC, USA, 2009), ICWS '09, IEEE Computer Society, pp. 655–662. 85

[32] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web Service Definition Language (WSDL). Tech. rep., Mar. 2001. 10, 12, 164

[33] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web services description language (wsdl) 1.1, w3c working group note. Tech. rep., W3C Working Group, March 2001. 6

[34] Colombo, M., Nitto, E. D., and Mauri, M. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In *ICSOC* (2006), A. Dan and W. Lamersdorf, Eds., vol. 4294 of *Lecture Notes in Computer Science*, Springer, pp. 191–202. 41

[35] Computing, A., et al. An architectural blueprint for autonomic computing. *IBM White Paper* (2006). 31

[36] Console, L., Ardagna, D., Ardissono, L., Bocconi, S., Odile, M., Drira, K., Eder, J., Friedrich, G., Fugini, M., Furnari, R., Goy, A., Guennoun, K., Hess, A., Ivanchenko, V., Guillou, X. L., Pencol, Y., Petrone, G., Pernici, B., Picardi, C., Pucel, X., Robin, S., Roz, L., Segnan, M., Tahamtan, A., Tejie, A. T., Dupr, D. T., Harmelen, F. V., Vidal, T., and Subias, A. 9 ws-diamond: Web servicesdiagnosability, monitoring, and diagnosis, 2008. 42

[37] Csenki, A. Bayes predictive analysis of a fundamental software reliability model. *Reliability, IEEE Transactions on 39*, 2 (1990), 177–183. 34

[38] da Silva, I., and Zisman, A. A framework for trusted services. In *Service-Oriented Computing*. Springer, 2012, pp. 328–343. 57

[39] da Silva, I., and Zisman, A. Decomposing ratings in service compositions. In *Service-Oriented Computing*. Springer, 2013, pp. 474–482. 58

[40] Dai, Y., Yang, L., and Zhang, B. Qos-driven self-healing web service composition based on performance prediction. *J. Comput. Sci. Technol. 24*, 2 (2009), 250–261. 47, 94

[41] Davey, B., Boland, N., and Stuckey, P. J. Efficient intelligent backtracking using linear programming. *INFORMS Journal on Computing 14*, 4 (2002), 373–386. 112

[42] Dawson, C. *Practical Research Methods: A User-friendly Guide to Mastering Research Techniques and Projects*. UBS Publishers' Distributors Pvt. Limited, 2003. 25

[43] Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., and Pohl, K. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Eng. 15*, 3-4 (Dec. 2008), 313–341. 3, 37

[44] Dumont, G. A., and Huzmezan, M. Concepts, methods and techniques in adaptive control. In *American Control Conference, 2002. Proceedings of the 2002* (2002), vol. 2, IEEE, pp. 1137–1150. 30

[45] Dustdar, S., and Papazoglou, M. P. Services and service composition - an introduction (services und service komposition - eine einführung). *it - Information Technology 50*, 2 (2008), 86–92. 3, 37

[46] Dustdar, S., and Schreiner, W. A survey on web services composition. *Int. J. Web Grid Serv. 1*, 1 (Aug. 2005), 1–30. 2, 12

[47] Erl, T. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005. 30

[48] Erl, T. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007. 2

[49] Farrell, J., and Lausen, H. Specification: Semantic annotations for wsdl and xml schema (sa-wsdl), 2007. 11

[50] Feng, T.-y. A survey of interconnection networks. *Computer 14*, 12 (1981), 12–27. 104

[51] Fu, S., and Xu, C.-Z. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing* (New York, NY, USA, 2007), SC '07, ACM, pp. 41:1–41:12. 72

[52] Fu, S., and Xu, C.-Z. Quantifying temporal and spatial correlation of failure events for proactive management. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on* (2007), IEEE, pp. 175–184. 34

[53] Gao, C., Cai, M., and Chen, H. Qos-aware service composition based on tree-coded genetic algorithm. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International* (2007), vol. 1, IEEE, pp. 361–367. 59, 60

[54] Ghalimi, I. Why bpel. Why BPEL Matters. Retrieved from http://itredux.com/2008/09/28/why-bpel. 14

[55] Ghandehari, M., and Stroulia, E. A lightweight coordination approach for resource-centric collaborations. In *REST: Advanced Research Topics and Practical Applications*, C. Pautasso, E. Wilde, and R. Alarcon, Eds. Springer New York, 2014, pp. 147–165. 15

[56] Giusto, D., Lera, A., Morabito, G., and Atzori, L. *The Internet of Things*. Springer, 2010. 199

[57] Guinea, S., Kecskemeti, G., Marconi, A., and Wetzstein, B. Multi-layered monitoring and adaptation. In *Proceedings of the 9th international conference on Service-Oriented Computing* (Berlin, Heidelberg, 2011), ICSOC'11, Springer-Verlag, pp. 359–373. 48

[58] Gunasinghe, T., and Kelly, T. Establishing a standard business process execution architecture for integrating web services. In *Proceedings of the IEEE International Conference on Web Services* (Washington, DC, USA, 2005), ICWS '05, IEEE Computer Society, pp. 365–372. 15

[59] Hackmann, G., Haitjema, M., Gill, C., and Roman, G.-C. Sliver: A bpel workflow process execution engine for mobile devices. In *Lecture Notes in Computer Science* (2006), vol. 4294, pp. 503–508. 16

[60] Hay, D. *Requirements Analysis: From Business Views to Architecture.* Prentice Hall, 2011. 13

[61] Heimerdinger, W. L., Heimerdinger, W. L., Weinstock, C. B., Weinstock, C. B., Miller, T. R., and Col, L. Tech. rep., Pittsburgh, Pennsylvania, USA. 33

[62] Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C., and Kopena, J. Network simulations with the ns-3 simulator. *SIGCOMM demonstration* (2008). 24, 135

[63] Hielscher, J., Kazhamiakin, R., Metzger, A., and Pistore, M. A framework for proactive self-adaptation of service-based applications based on online testing. In *Proceedings of the 1st European Conference on To-*

wards a Service-Based Internet (Berlin, Heidelberg, 2008), ServiceWave '08, Springer-Verlag, pp. 122–133. 37

[64] Hielscher, J., Metzger, A., and Kazhamiakin, R. Taxonomy of adaptation principles and mechanisms. Tech. rep., 2009. 3

[65] Hollingsworth, D. Workflow Management Coalition: The Workflow Reference Model, 1995. 15

[66] Huang, A. F., Lan, C.-W., and Yang, S. J. An optimal qos-based web service selection scheme. *Information Sciences 179*, 19 (2009), 3309–3322. 59

[67] Hunter, J. S. The exponentially weighted moving average. *J. QUALITY TECHNOL. 18*, 4 (1986), 203–210. 22

[68] Ivanovic, D., Kaowichakorn, P., and Carro, M. Towards qos prediction based on composition structure analysis and probabilistic environment models. In *Principles of Engineering Service-Oriented Systems (PESOS), 2013 ICSE Workshop on* (2013), IEEE, pp. 11–20. 58

[69] Jaeger, M. C., Mühl, G., and Golze, S. Qos-aware composition of web services: An evaluation of selection algorithms. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*. Springer, 2005, pp. 646–661. 58

[70] Ji, X. Research on web service discovery based on domain ontology. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on* (Aug 2009), pp. 65–68. 56

[71] Jiang-Hong, J. Z.-Y. H., and Zhao, W. An optimization model for dynamic qos-aware web services selection and composition. *Chinese Journal of Computers 5* (2009), 019. 56

[72] Jun, N., Bin, Z., Xiangyu, Z., Zhiliang, Z., and Dancheng, L. Two-stage adaptation for dependable service-oriented system. In *Service Sciences (ICSS), 2010 International Conference on* (2010), IEEE, pp. 143–147. iii, 53, 54

[73] Juric, M. B. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing, 2006. 14, 144, 164, 168, 194

[74] Kankanamge, C. *Web Services Testing With Soapui*. Packt, 2012. 121

[75] Kay, R. Introduction to network latency engineering. 19

[76] Kazhamiakin, R., Wetzstein, B., Karastoyanova, D., Pistore, M., and Leymann, F. Adaptation of service-based applications based on process quality factor analysis. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops - International Workshops, ICSOC/ServiceWave 2009, Stockholm, Sweden, November 23-27, 2009, Revised Selected Papers* (2009), A. Dan, F. Gittler, and F. Toumani, Eds., vol. 6275 of *Lecture Notes in Computer Science*, pp. 395–404. 42

[77] Keen, M., Balani, N., Natraj, A., Chaubal, A., Nadgir, D., Sharma, M., Steele, M., and Tost, A. *Getting started with ibm websphere business services fabric v6.1*. IBM Redbooks, 2008. 16, 194

[78] Kephart, J. O., and Chess, D. M. The vision of autonomic computing. *Computer 36*, 1 (2003), 41–50. 30

[79] Kongdenfha, W., Motahari-Nezhad, H. R., Benatallah, B., Casati, F., and Saint-Paul, R. Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters. *IEEE Trans. Serv. Comput. 2*, 2 (Apr. 2009), 94–107. 47

[80] Koren, I., and Krishna, C. M. *Fault-Tolerant Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. 32

[81] Kothari, C. *Research Methodology: Methods and Techniques*. New Age International (P) Limited, 2004. 25

[82] Krafzig, D., Banke, K., and Slama, D. *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall Professional, 2005. 30

[83] Kumar, R. *Research Methodology: A Step-by-Step Guide for Beginners*. SAGE Publications, 2014. 25

[84] Leangsuksun, C., Liu, T., Rao, T., Scott, S., and Libby, R. A failure predictive and policy-based high availability strategy for linux high performance computing cluster. In *The 5th LCI International Conference on Linux Clusters: The HPC Revolution* (2004), Citeseer, pp. 18–20. 35

[85] Lécué, F. *Optimizing qos-aware semantic web service composition*. Springer, 2009. 59, 60

[86] Leite, L. A. F., Oliva, G. A., Nogueira, G. M., Gerosa, M. A., Kon, F., and Milojicic, D. S. A systematic literature review of service

choreography adaptation. *Service Oriented Computing and Applications 7*, 3 (2013), 199–216. 15

[87] Leitner, P., Michlmayr, A., Rosenberg, F., and Dustdar, S. Monitoring, prediction and prevention of sla violations in composite services. In *IEEE International Conference on Web Services, ICWS 2010, Miami, Florida, USA,July 5-10, 2010* (2010), IEEE Computer Society, pp. 369–376. 47, 48

[88] Li, L., Vaidyanathan, K., and Trivedi, K. S. An approach for estimation of software aging in a web server. In *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n* (2002), IEEE, pp. 91–100. 34

[89] Li, X., Fan, Y., Sheng, Q. Z., Maamar, Z., and Zhu, H. A petri net approach to analyzing behavioral compatibility and similarity of web services. *Trans. Sys. Man Cyber. Part A 41*, 3 (May 2011), 510–521. 47

[90] Li, Y., Zhou, M.-h., Li, R.-c., Cao, D.-G., and Mei, H. Service selection approach considering the trustworthiness of qos data. *Journal of Software 19*, 10 (2008), 2620–2627. 57

[91] Lin, K.-J., Zhang, J., Zhai, Y., and Xu, B. The design and implementation of service process reconfiguration with end-to-end qos constraints in soa. *Serv. Oriented Comput. Appl. 4*, 3 (Sept. 2010), 157–168. iii, 45, 46

[92] Linden, G. Make your data useful. Presentation of Amazon results. Retrieved from http://www.scribd.com/doc/4970486. 19

[93] Lins, F. A. A., dos Santos Júnior, J. C., and Rosa, N. S. Adaptive web service composition. *SIGSOFT Softw. Eng. Notes 32*, 4 (July 2007). 43, 44, 45

[94] Liu, Y., Ngu, A. H., and Zeng, L. Z. Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (2004), ACM, pp. 66–73. 56, 57

[95] Lowe, D., Ed. *BizTalk server*. Osborne/McGraw-Hill, New York [u.a.], 2002. 40

[96] Lyu, M. R., Ed. *Handbook of software reliability engineering*. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996. 3

[97] M, X. *Software Reliability Modelling*. [Series on quality, reliability & engineering statistics. World Scientific, 1991. 34

[98] Mabrouk, N. B., Georgantas, N., and Issarny, V. A semantic end-to-end qos model for dynamic service oriented environments. In *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems* (2009), IEEE Computer Society, pp. 34–41. 58

[99] Martin, D., et al. Daml-s (and owl-s) 0.9 draft release (may 2003). *Online http://www. daml. org/services/daml-s/0.9*. 10

[100] Maximilien, E. M., and Singh, M. P. Toward autonomic web services trust and selection. In *Proceedings of the 2nd international conference on Service oriented computing* (2004), ACM, pp. 212–221. 56

[101] Mayer, M. What google knows. 19

[102] McCanne, S., Floyd, S., and Fall, K. The lbnl network simulator. *Software on-line: http://www. isi. edu/nsnam* (1997). 135

[103] Menascé, D. A., Casalicchio, E., and Dubey, V. A heuristic approach to optimal service selection in service oriented architectures. In *Proceedings of the 7th international workshop on Software and performance* (2008), ACM, pp. 13–24. 58, 60

[104] Meng, H.-N., Qi, Y., Hou, D., and Chen, Y. A rough wavelet network model with genetic algorithm and its application to aging forecasting of application server. In *Machine Learning and Cybernetics, 2007 International Conference on* (2007), vol. 5, IEEE, pp. 3034–3039. 35

[105] Metzger, A., Sammodi, O., Pohl, K., and Rzepka, M. Towards pro-active adaptation with confidence: augmenting service monitoring with online testing. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* (New York, NY, USA, 2010), SEAMS '10, ACM. 47, 49

[106] Metzger, A., Schmieders, E., Cappiello, C., Di Nitto, E., Kazhamiakin, R., Pernici, B., and Pistore, M. Towards proactive adaptation: A journey along the s-cube service life-cycle. In *MESOA: 4th International Workshop on Maintenance and Evolution of Service-Oriented Systems* (2010). iii, 51, 52

[107] Mike P. Papazoglou, Paolo Traverso, S. D., and Leymann, F. Service-oriented computing: a research roadmap. *Int. J. Cooperative Inf. Syst. 17*, 2 (2008), 223–255. 3, 37, 40

[108] Miyagi, M., Ohkubo, K., Kataoka, M., and Yoshizawa, S. Performance prediction method for web-access response time distribution using formula. In *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP* (april 2004), vol. 1, pp. 905 –906 Vol.1. 95

[109] Mokhtar, S. B., Liu, J., Georgantas, N., and Issarny, V. Qos-aware dynamic service composition in ambient intelligence environments. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (New York, NY, USA, 2005), ASE '05, ACM, pp. 317–320. 85

[110] Moser, O., Rosenberg, F., and Dustdar, S. Non-intrusive monitoring and service adaptation for ws-bpel. In *Proceedings of the 17th international conference on World Wide Web* (New York, NY, USA, 2008), WWW '08, ACM, pp. 815–824. 48

[111] Na, J., Gao, Y., Zhang, B., Huang, L.-p., and Zhu, Z.-l. Improved adaptation of web service composition based on change impact probability. In *Proceedings of the 2010 Third International Conference on Dependability* (Washington, DC, USA, 2010), DEPEND '10, IEEE Computer Society, pp. 146–153. 48, 49

[112] Nelson, V. P. Fault-tolerant computing: Fundamental concepts. *Computer 23*, 7 (1990), 19–25. 30

[113] OASIS. Organization for the advancement of structured information standards. Website available from https://www.oasis-open.org. 14

[114] ODE. Apache ode - the orchestration director engine. Website available from http://ode.apache.org/. 16

[115] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, Apr. 2007. 74

[116] Palmer, N. Tech. rep. 14

[117] Papazoglou, M. P. Service -oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering* (Washington, DC, USA, 2003), WISE '03, IEEE Computer Society, pp. 3–. 12

[118] Parazoglou, M., Traverso, P., Dustdar, S., and Leymann, F. Service oriented computing: State of the art and research challenges. *Computer 40*, 11 (2007), 38–45. 18

[119] Perera, S., Herath, C., Ekanayake, J., Chinthaka, E., Ranabahu, A., Jayasinghe, D., Weerawarana, S., and Daniels, G. Axis2, middleware for next generation web services. In *Proceedings of the IEEE International Conference on Web Services* (Washington, DC, USA, 2006), ICWS '06, IEEE Computer Society, pp. 833–840. 121

[120] Pernici, B., Ed. *Mobile Information Systems: Infrastructure and Design for Adaptivity and Flexibility.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. 41

[121] Pistore, M., Marconi, A., Bertoli, P., and Traverso, P. Automated composition of web services by planning at the knowledge level. In *Proceedings of the 19th international joint conference on Artificial intelligence* (San Francisco, CA, USA, 2005), IJCAI'05, Morgan Kaufmann Publishers Inc., pp. 1252–1259. 41

[122] Popescu, R., Staikopoulos, A., Liu, P., Brogi, A., and Clarke, S. Taxonomy-driven adaptation of multi-layer applications using templates. In *Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems* (Washington, DC, USA, 2010), SASO '10, IEEE Computer Society, pp. 213–222. 48

[123] Pradhan, D. K. Fault-tolerant computing: Theory and technique, volume i. 30

[124] Russell, J., and Cohn, R. *Process Flow Diagram.* Book on Demand, 2012. 13

[125] Salatino, M. *jBPM Developer Guide.* Packt Publishing, 2010. 16

[126] Salfner, F., Lenk, M., and Malek, M. A survey of online failure prediction methods. *ACM Comput. Surv. 42*, 3 (Mar. 2010), 10:1–10:42. 34, 39, 73

[127] Sammodi, O., Metzger, A., Franch, X., Oriol, M., Marco, J., and Pohl, K. Usage-based online testing for proactive adaptation of service-based applications. In *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual* (2011), IEEE, pp. 582–587. 51

[128] Schuller, D., Polyvyanyy, A., García-Bañuelos, L., and Schulte, S. Optimization of complex qos-aware service compositions. In *Proceedings of the 9th international conference on Service-Oriented Computing* (Berlin, Heidelberg, 2011), ICSOC'11, Springer-Verlag, pp. 452–466. 59, 60, 88

[129] Siqueira, M. G., and Alwan, A. Steady-state analysis of continuous adaptation in acoustic feedback reduction systems for hearing-aids. *Speech and Audio Processing, IEEE Transactions on 8*, 4 (2000), 443–453. 30

[130] Spanoudakis, G., and Zisman, A. Discovering services during service-based system design using uml. *IEEE Trans. Softw. Eng. 36*, 3 (May 2010), 371–389. 79, 108

[131] Stephanidis, C., Paramythis, A., Akoumianakis, D., and Sfyrakis, M. Self-adapting web-based systems: Towards universal accessibility. In *4th Workshop on User Interface For All, Stockholm, Sweden* (1998). 31

[132] Strunk, A. Qos-aware service composition: A survey. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on* (2010), IEEE, pp. 67–74. 59

[133] Talaq, J., and Al-Basri, F. Adaptive fuzzy gain scheduling for load frequency control. *Power systems, IEEE transactions on 14*, 1 (1999), 145–150. 30

[134] Tanner, J. Feedback control in living prototypes: A new vista in control engineering. *Medical electronics and biological engineering 1*, 3 (1963), 333–351. 30

[135] Tosi, D., Denaro, G., and Pezze, M. Towards autonomic service-oriented applications. *Int. J. Autonomic Comput. 1*, 1 (Apr. 2009), 58–80. 47, 49

[136] Vu, L.-H., Hauswirth, M., and Aberer, K. Qos-based service selection and ranking with trust and reputation management. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*. Springer, 2005, pp. 466–483. 57

[137] Wan, S., Wei, J., Song, J., and Guan, H. Developing a selection model for interactive web services. In *Web Services, 2006. ICWS '06. International Conference on* (Sept 2006), pp. 231–238. 56

[138] WfMC. Workflow management coalition. Website available from http://www.wfmc.org. 14

[139] Wu, Z., Deng, S., Li, Y., and Wu, J. Computing compatibility in dynamic service composition. *Knowl. Inf. Syst. 19*, 1 (Mar. 2009), 107–129. 47

[140] Xianglan, H., Yangguang, L., Bin, X., and Gang, Z. A survey on qos-aware dynamic web service selection. In *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on* (2011), IEEE, pp. 1–5. 56

[141] Xie, M. *Software Reliability Modelling*. Series on quality, reliability & engineering statistics. World Scientific, Oct 1991. 32

[142] Xu, D., and Nahrstedt, K. Finding service paths in a media service proxy network. In *Electronic Imaging 2002* (2001), International Society for Optics and Photonics, pp. 171–185. 56

[143] Zeng, L., Benatallah, B., Ngu, A. H., Dumas, M., Kalagnanam, J., and Chang, H. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on 30*, 5 (2004), 311–327. 59, 60

[144] Zengin, A., Kazhamiakin, R., and Pistore, M. Clam: Cross-layer management of adaptation decisions for service-based applications. In *Proceedings of the 2011 IEEE International Conference on Web Services* (Washington, DC, USA, 2011), ICWS '11, IEEE Computer Society, pp. 698–699. 48

[145] Zhai, Y., Zhang, J., and Lin, K. SOA Middleware Support for Service Process Reconfiguration with End-to-End QoS Constraints. In *Proceedings of the 2009 IEEE International Conference on Web Services* (Washington, DC, USA, 2009), ICWS '09, IEEE Computer Society, pp. 815–822. 42

[146] Zisman, A., Spanoudakis, G., and Dooley, J. A framework for dynamic service discovery. In *Proceedings of the 2008 23rd IEEE/ACM In-*

ternational Conference on Automated Software Engineering (Washington, DC, USA, 2008), ASE '08, IEEE Computer Society, pp. 158–167. 79, 108

[147] Zisman, A., Spanoudakis, G., Dooley, J., and Siveroni, I. Proactive and reactive runtime service discovery: A framework and its evaluation. *Transactions on Software Engineering*. To appear. 159, 160, 166, 167, 168, 175, 190

[148] ström, K. J., and Hägglund, T. Pid controllers: theory, design, and tuning. *Instrument Society of America, Research Triangle Park, NC* (1995). 30