Title: Validated principles, techniques and methodologies for specifying end-to-end quality and negotiating SLAs and for assuring end-to-end quality provision and SLA conformance (incl. proactiveness)

Authors: CITY, FBK, LERO, POLIMI, UniDue, UPC, UPM

Editor: Pierluigi Plebani (POLIMI), Andreas Metzger (UniDue), Osama Sammodi (UniDue)

Reviewers: Kyriakos Kritikos (UoC), Gabriele Tolomei (CNR)

Identifier: CD-JRA-1.3.6

Type: Contractual Deliverable (CD)

Version: 1.0

Date: February 29th, 2012

Status: Final

Class: External

Management summary

The aim of this deliverable is to report the results of the fourth year of the S-Cube network on research topics related to proactive quality negotiation and assurance. This paper-based deliverable summarizes the network’s research results that have been published in books, journals, and conference proceedings.

Copyright 2012 by the S-Cube consortium – All rights reserved.

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement n 215483 (S-Cube).
Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator) – UniDue
Tilburg University – Tilburg
City University London – CITY
Consiglio Nazionale delle Ricerche – CNR
Center for Scientific and Technological Research – FBK
French Natl Institute for Research in Computer Science and Control – INRIA
The Irish Software Engineering Research Centre – Lero
Politecnico di Milano – Polimi
MTA SZTAKI – Computer and Automation Research Institute – SZTAKI
Vienna University of Technology – TUW
Universit Claude Bernard Lyon – UCBL
University of Crete – UOC
Technical University of Madrid – UPM
University of Stuttgart – USTUTT
University of Amsterdam – VUA
University of Hamburg – UniHH

Associate Members of the S-Cube consortium:

Centre of Excellence in Information and Communication Technologies – CETIC
Centre for Service Research, Manchester Business School, The University of Manchester
Dipartimento di Informatica, Università di Pisa
Faculty of Informatics, Università della Svizzera Italiana – USI
Institute of Computer Science, University of Münster
IT and Media Center & Department of Computer Science, Dortmund University of Technology
IT Innovation Centre, University of Southampton
Karlsruhe Institute of Technology – KIT
School of Software, Tsinghua University
Semantic Technology Institute Innsbruck, University of Innsbruck
Service & Information Systems Engineering Department, Universitat Politècnica de Catalunya – UPC
SINTEF
South East European Research Centre – SEERC
University of L’Aquila
Université Européenne de Bretagne / Université de Bretagne-Sud
University of Groningen
Universidad de Sevilla

Published S-Cube documents

These documents are all available from the S-Cube Web Portal at http://www.s-cube-network.eu/
Foreword

Workpackage JRA-1.3 ("End-to-End Quality Provision & SLA Conformance") of S-Cube has been designed to achieve four long-term objectives:

1. To define principles and techniques for specifying, negotiating and assuring end-to-end quality provision and SLA conformance (including proactiveness) with respect to quality characteristics across the functional layers of service infrastructure, service composition and coordination, and business process management, and, across the chain of service providers and consumers. The quality characteristics to be considered included characteristics such as performance, dependability, reliability and availability.

2. To specify clearly defined interfaces and the interrelationships with respect to end-to-end quality aspects:
   - Between functional layers service infrastructure, service composition and coordination, business process management and the SBA engineering framework, and
   - Between the SBA (service-based application) engineering framework and the SBA monitoring and adaptation framework.

3. To shape the S-Cube convergence knowledge model by providing an integrated set of definitions, principles and techniques for end-to-end quality assurance and SLA conformance.

4. To provide contributions to IA-3 ("Integration Framework for Service-based Applications"), where the results are integrated into the S-Cube Framework for Service-Based Applications.

As part of these long-term goals, this deliverable provides a consolidated report on results for run-time quality assurance, quality prediction (to enable proactive adaptation) and automated and proactive negotiation, many of which have been experimentally validated, thereby contributing to the research challenges of the WP.

Acknowledgments: The editors would like to thank the authors of the papers, technical reports, articles and book chapters described in this deliverable for allowing their work to be used in this document.
Contents

1 Deliverable Overview 3
  1.1 Introduction ................................................................. 3
  1.2 Deliverable Structure .................................................... 4

2 Research Challenges and Contribution to the Integrated Research Framework 5
  2.1 Key Research Challenges .................................................. 5
    2.1.1 Challenges In Quality Definition ..................................... 6
    2.1.2 Challenges In Quality Negotiation ..................................... 7
    2.1.3 Challenges in Quality Assurance ....................................... 7
  2.2 Contribution of Deliverable to Key Research Challenges ................. 8

3 Principles, Techniques and Methodologies for Service Quality 9
  3.1 Structured Presentation of Results ....................................... 9
  3.2 Summary of Research Results ............................................ 10
    3.2.1 Usage-Based Online Testing for Proactive Adaptation of Service-Based Applications .................................................. 10
    3.2.2 Towards Accurate Failure Prediction for the Proactive Adaptation of Service-oriented Systems ............................................. 13
    3.2.3 Accurate Service Failure Prediction through Online Testing .................................................. 15
    3.2.4 Proactive SLA Negotiation for Service Based Systems: Initial Implementation and Evaluation Experience ............................................. 17
    3.2.5 SALMonADA: A platform for Monitoring and Explaining Violations of WS Agreement-compliant Documents ................................. 19
    3.2.6 Preventing Performance Violations of Service Compositions using Assumption-based Run-time Verification ............................................. 21
    3.2.7 Future Internet Apps: The Next Wave of Adaptive Service-Oriented Systems? .................................................. 22
    3.2.8 Adaptive Future Internet Applications: Opportunities and Challenges for Adaptive Web Services Technology ............................................. 23
    3.2.9 SLAs for Cross-layer Adaptation and Monitoring of Service-Based Applications: A Case Study .................................................. 25
    3.2.10 Negotiation towards Service Level Agreements: A Life Cycle Based Approach .................................................. 27
    3.2.11 A Context-Aware Framework for Business Process Evolution .................................................. 28
    3.2.12 Constraint-Based Runtime Prediction of SLA Violations in Service Orchestrations .................................................. 32

4 Conclusions 35
  4.1 Future Research Activities on Online Quality Prediction ............... 35
Chapter 1

Deliverable Overview

1.1 Introduction

Although work in Service Oriented Computing (SOC) often focuses on the what a service is able to perform, i.e., its functional aspects, quality (aka. non-functional) aspects play an important role during the entire life-cycle of a service-based application (SBA). As an example, when services are designed and deployed, their performance needs to be taken into account to ensure their usefulness. In this way, during the service discovery and matchmaking phase, service integrators (who build and deploy the SBA), when given a set of functionally equivalent services, are able to select which is the best amongst them. Employing suitable negotiation frameworks, such service selection can be personalized in a sense that the service provider and the service integrator (consumer) can “make a deal” concerning the behaviour of the services when they will be invoked. Ultimately, this leads to contracts (aka. Service-Level Agreements – SLAs) being required in the setting of SBAs. Finally, when the invocation takes place, both parties need to be sure that the execution satisfies what was established in the contract, i.e., they need to perform quality assurance of the services.

The work performed in workpackage WP-JRA-1.3 during the whole duration of the S-Cube network focused on all these aspects that, according to [37, 50], can be summarized as follows:

- **Quality definition**: This concerns the definition of a model or language for the specification of contract terms, which is understood and shared by the (contracting) parties. This model or language then is used to instantiate an actual contract (e.g., an SLA) that reflects the domain dependent interests of providers and consumers, or to state the end-to-end quality requirements towards an SBA.

- **Quality negotiation**: The establishment of an electronic contract concerns the set of tasks that is required for defining actual contracts. This may involve the selection of service providers (the contract partners) among a set of potential providers, the negotiation of the contract terms between the selected providers and the service consumer, and the agreement to the contract terms.

- **Quality assurance**: This concerns tasks for assuring the satisfaction of the contracts and the fulfilment of the expected end-to-end requirements. In the case of quality contracts, this implies assuring that the quality levels negotiated and agreed between the service provider and the service requestor are met.

This deliverable summarizes the results achieved during the fourth year of the S-Cube network, especially in terms of quality negotiation and assurance considering the end-to-end quality of SBAs. This deliverable documents principles and techniques for automated and proactive quality assurance and negotiation. The work concerns (1) handling quality of SBAs that are composed by services that may be provided by different third-party providers, (2) run-time prediction of the quality provided by the whole...
application as well as its services, (3) techniques to support the proactive adaptation of the application in case imminent failures may affect the quality of the overall application.

1.2 Deliverable Structure

As this deliverable is paper-based, it contains two parts: (a) This document, which provides the overall motivation and summary of the key research outcomes of the workpackage. (b) The actual research publications that describe the workpackage outcomes in detail and which are summarized in this document.

The document is structured as follows: in Chapter 2, this deliverable provides an overview of the research challenges being pursued within WP-JRA-1.3. In Chapter 3, the document – based on research publications of S-Cube members and associate members – reports on a set of principles and techniques for quality definition, negotiation and assurance developed in year 4 of the network, thereby contributing to the research challenges of the workpackage. Chapter 4 concludes the deliverable and provides an outlook to future work in the workpackage.
Chapter 2

Research Challenges and Contribution to the Integrated Research Framework

As described in S-Cube’s Description of Work (DoW), the general research goal of workpackage WP-JRA-1.3 is to devise novel principles, techniques and methods for defining, negotiating and assuring end-to-end quality across the functional layers as well as across networks of service providers and consumers.

This chapter provides an update of the research vision for this workpackage by defining and refining the research challenges addressed in WP-JRA-1.3. Those challenges aimed at addressing key research gaps identified in [32]. Thereby, this chapter allows for relating the results presented in this deliverable to the research objectives of the workpackage (see Sections 2.2 and Chapter 3). Specifically for year 4 of the project, focus of research activities was on challenges related to “proactiveness”, i.e., on challenges introduced in Sections 2.1.2.2 and 2.1.3.2.

2.1 Key Research Challenges

Figure 2.1 provides and overview of the WP’s research challenges in the context of the quality activities (as introduced in Section 1.1). Those research challenges are detailed below.

Figure 2.1: Activities relevant for quality of SBAs and key research objectives of workpackage
2.1.1 Challenges In Quality Definition

Note: Work on challenges related to quality definition constituted the foundations for the work on other challenges of the workpackage. Those activities have been finalized – as planned – by year 2 of S-Cube. The research results related to these research challenges can be found in the S-Cube deliverables CD-JRA-1.3.2 [9] and CD-JRA-1.3.3 [19].

2.1.1.1 End-to-End Quality Reference Model

Motivation: Different kinds of quality attributes are important in an SBA. There is thus a strong need for methods that address quality attributes in a comprehensive and cross-cutting fashion across all layers of an SBA. Due to the dynamism of the world in which SBAs operate, techniques are needed to aggregate individual quality levels of the services involved in a service composition in order to determine and thus check the end-to-end quality during run-time of the application. This aggregation will typically span different layers of an SBA and thus a common understanding of what the different quality attributes mean within and across these layers is needed.

Challenge: To support end-to-end quality provision, S-Cube has aimed at making the dependencies between different kinds of quality attributes explicit. For instance, the interrelation between the fulfillment of different QoS attributes across the various layers has been modeled. In addition, S-Cube has aimed at understanding the dependencies between quality of information (QoI) attributes on the infrastructure layer, the satisfaction of quality of experience (QoE) on the service composition layer and the achievement of quality of business (QoBiz; business value or business KPIs). One key means to achieve the above objective has been to achieve a shared understanding of quality attributes between the S-Cube layers and disciplines by defining the S-Cube Quality Reference Model. Based on the S-Cube Quality Reference Model and the quality definition language (see Challenge “Rich and Extensible Quality Definition Language” in Section 2.1.1.2 below), foundations for techniques will be devised, which allow aggregating individual quality levels of the services involved in a service composition in order to determine and thus ultimately check end-to-end quality.

2.1.1.2 Rich and Extensible Quality Definition Language

Motivation: Concerning quality modeling and definition, this project has observed that there is a lack of an established, standardized, rich, extensible and semantically well-defined quality definition language [42]. As a result, quality capabilities and requirements, as well as service SLAs are described by many different formalisms and languages, such as the WSLA language [23], WSML [43], SLAng [20] and RBSLA [34] (amongst many others). Due to this fragmentation, there is still a requirement for a standardized and definition language — necessary for interoperable services.

Challenge: S-Cube has developed the concepts for a quality definition language (i.e., the S-Cube Quality Meta Model), which allow describing every relevant aspect of quality for services and SBAs, including metrics, units, measurement functions and directives, constraints, value types, etc. In addition, this quality definition language encompasses a rich set of domain-dependent and global quality attributes and is extensible so as to allow the addition of new quality dimensions when it is needed (e.g., for an application domain which has currently not been considered). As a starting point, the set of quality attributes as defined in the S-Cube Quality Reference Model (see Challenge “End-to-End Quality Reference Model” above) has been exploited. Further, this standard quality definition language is semantically enriched - where feasible - to be machine-processable or machine-interpretable. This quality definition language is created to be applicable in complex SBAs, in which services can be invoked and composed with variable quality profiles. The quality definition language is capable of expressing quality capabilities and SLAs by using functions, operators and comparison predicates on quality metrics. It also allows the description of composition rules for possible combinations of composition constructs and quality metrics.
2.1.2 Challenges In Quality Negotiation

2.1.2.1 Automatic quality contract establishment

*Motivation:* Service negotiation and agreement involves selecting one out of many service providers based on his quality offer so as to agree on and thus establish the contracts for the delivered service. To address dynamic adaptations of SBAs, a growing need for automating the negotiation and agreement of quality attributes (e.g., as stipulated by SLAs) can be observed. However, this issue requires considering user interaction and experience (e.g., QoE) issues that may impact on the negotiation itself. This aspect requires a multi-disciplinary effort in which technology researchers will have to interact with researchers addressing user interaction issues.

*Challenge:* One key research objective regarding quality contract establishment is to exploit user and task models, which codify user preferences and characteristics (see JRA-1.1), in order to devise advanced automated negotiation techniques and protocols. Those advanced techniques could lead to service negotiators (e.g., autonomous components provided as core services) that perform the negotiation process on behalf of the service consumers (requestors) and providers.

2.1.2.2 Proactive SLA negotiation and agreement

*Motivation:* Similar to proactive (and possibly automated) adaptation (see Challenge “Quality Prediction Techniques to Support Proactive Adaptation” in Section 2.1.3.2), proactive SLA negotiation and agreement is a key prerequisite for effective run-time SLA negotiation since negotiation may have a significant computational cost and, therefore, undertaking it when there is an immediate need to use a new service can be unlikely or unfeasible at run-time.

*Challenge:* The challenge for quality contract negotiation and agreement is how to negotiate the terms and conditions under which a service can be offered before the need for deploying or invoking these services arises. Many of these challenges lie in the definition of negotiation models, and to make the envisioned advances in automated negotiation. We aim to address the limitations introduced above by starting negotiation when there is evidence that the need for deploying a new service and/or change the conditions of deploying a current service is likely to arise but has not arisen yet. Thus, our proactive negotiation approach is based on forecasting at run-time a number of factors related to the deployment of services. Those include, for example, the expected demand for a service, the expected levels of service provision, and the expected service terms and conditions that a service negotiator is likely to agree. The availability of accurate forecasts can lead to effective proactive run-time negotiation strategies for service clients. Prediction also plays a role in quality prediction for proactive adaptation (see Challenge “Quality Prediction Techniques to Support Proactive Adaptation” in Section 2.1.3.2). Although the factors which are relevant differ in both situations, we expect to be able to exploit synergies between the principles and techniques that are developed.

2.1.3 Challenges in Quality Assurance

2.1.3.1 Run-time Quality Assurance Techniques

*Motivation:* Given the need for adapting SBAs at run-time, quality assurance techniques that can be applied at run-time are essential. The major type of run-time quality assurance techniques used today is monitoring, which is often classified as passive (when monitoring relies on actual inbound service consumer traffic to take measurements, so problems can only be discovered after they have occurred) or active (e.g., during run-time testing where the consumer traffic is generated by the testing agent). Monitoring observes the SBA (or its constituent services) during their current execution, i.e., during their actual use or operation. However, monitoring only allows the assessment of the quality of ‘representative’ applications (in fact the application in operation) and thus key problems might only be discovered by coincidence. In contrast, standard and consolidated software quality assurance techniques employed
during design time, can uncover problems that might only occur after many invocations of the SBA. As an example, model analysis can examine classes of executions, thereby leading to more universal statements about the properties of the artifacts.

**Challenge:** S-Cube investigates in how standard and consolidated offline software quality assurance techniques can be extended to be applicable while the application operates. For instance, we investigate into run-time model analysis techniques and other online techniques such as online testing. In addition to extending the quality assurance techniques to the operation phase, synergies between the different classes of quality assurance techniques are exploited. As an example, we investigate how testing can be combined with monitoring in such a way that when a deviation is observed during monitoring, dedicated test cases are executed in order to determine - with high confidence - the cause for the deviation. In order to achieve feasible results from run-time quality assurance, it is essential that the artifacts exploited for run-time analysis or testing are a consistent and up-to-date representation (abstraction) of the running SBA. For example, this leads to the challenge on how to 'synchronize' the model with the SBA in order to achieve valid analysis results. Existing quality assurance techniques appear to be not yet fully incorporated into a comprehensive life-cycle. These aspects are particularly critical as the designers find it quite difficult to understand what will happen as a result of some self-adaptation design choice. Research, jointly with WP-JRA-1.1, thus addresses the consistent and comprehensive integration of quality assurance into the service life-cycle (see JRA-1.1).

### 2.1.3.2 Quality Prediction Techniques to Support Proactive Adaptation

**Motivation:** To respond in a timely fashion to changes implied by the highly dynamic and flexible contexts of future SBAs and to promptly compensate for deviations in functionality or quality, SBAs have to be able to self-adapt. In current solutions, self-adaptation often happens after a change or a deviation has occurred, i.e., in a reactive fashion. Yet, such reactive adaptations have several drawbacks, such as:

1. Executing faulty services can lead to unsatisfied users and typically requires the execution of additional activities (e.g., compensation or roll-back);
2. Execution of adaptation activities takes time and thereby can reduce the system performance;
3. It can take time before problems in the system lead to monitoring events (e.g., time needed for the propagation of events from the infrastructure to the business process level), thus events might arrive so late that an adaptation of the system is not possible anymore (e.g., because the system is in a deadlock situation).

Proactive adaptation presents a solution to address these drawbacks, because – ideally – the system will detect the need for adaptation and will self-adapt before a deviation will be observed by the users of the SBA. Key to proactive adaptation is to predict the future quality (and functionality) of a SBA and to proactively plan mitigation and repair activities [44] if the prediction uncovers deviations from expected quality (or functionality).

**Challenge:** To support the vision of proactive adaptation, S-Cube works on devising novel quality prediction techniques. These build on concepts such as online testing, run-time model analysis, model-checking at run-time, static analysis, simulation and machine learning.

### 2.2 Contribution of Deliverable to Key Research Challenges

Summarizing the above challenges, WP-JRA-1.3 pursues integrative and innovative research to devise novel principles and techniques for defining, negotiating and assuring end-to-end quality for SBAs.

This deliverable provides an overview of the research results of the WP-JRA-1.3 members for year 4 of the project. More specifically, this document summarizes the key research outcomes in addressing challenges in Quality Negotiation (see Section 2.1.2) and Quality Assurance (see Section 2.1.3). In the remainder of this document, those research outcomes will be presented and related to the WP challenges as well as to the other S-Cube WPs in more detail.
Chapter 3

Principles, Techniques and Methodologies for Service Quality

The main research results in the context of this deliverable have been published or submitted as research papers, articles and book chapters. Thus, in this section we will present (in compact form) the contributions of those papers and how these contributions relate to the WP research challenges described earlier in Section 2.1.

It should be noted that due to S-Cube being a Network of Excellence, and not an Integrated Project, the papers that constitute this deliverable present solutions to the WP challenges from different angles rather than different “views” on the very same technical solution. More important for S-Cube is the fact that those papers document a significant step towards integration of the different research communities that participate in S-Cube. As an example, the techniques exploit software engineering solutions (such as testing, verification or model-driven development) and techniques from SOC (service composition) and service infrastructures to address problems specific to SBAs.

3.1 Structured Presentation of Results

The following 8-part structure, inspired in parts by “How to Get Your Paper Accepted at OOPSLA” [31], is used to describe each of the reports, papers and articles that form the results of this deliverable.

- **Context and Background:** Initially, the context and background of the problem being addressed in the paper is provided.
- **Problem Statement:** Based on the background, the problem that is addressed (i.e., the research question which is answered) is motivated and explained.
- **Relevance of the Problem and Progress from State of the Art:** The explanation on why the problem is relevant is important to understand why the problem (i.e., research question) is worth pursuing. In addition, the relation of the work to the state of the art helps understanding the novelty of the contribution and its progress from existing work.
- **Relation to WP Challenges:** The contribution to the WP research challenges is described to understand the contribution of the paper to the overall aims of the deliverable (cf. Section 2.2) and the WP.
- **Solution / Research Method:** Either the (innovative) solution (idea) to the problem is stated or the employed research method (e.g., empirical study) is described.
- **Benefits and Evaluation:** The benefits and utility of the solution when applied to the problem is stated, and, if applicable, it is described how those benefits have been demonstrated by means of an evaluation (method of evaluation and results).
- **Relation to Research Framework:** The solution of the paper is related to the elements of the S-Cube Research Framework and thus to S-Cube JRA work packages, thereby describing the integration
achieved across JRA-1 and JRA-2: Monitoring and Adaptation (JRA-1.2), Engineering and Design (JRA-1.1), BPM (JRA-2.1), Service Composition and Coordination (JRA-2.2), and Service Infrastructure (JRA-2.3).

- Discussion and Future Work: Critical discussion on what are the current gaps and shortcomings of the solution and which future research activities are planned. This will allow shaping the future research roadmap for the WP.

3.2 Summary of Research Results

Table 3.2 summarizes the research results achieved by the S-Cube project during the fourth year in the areas of Quality Negotiation, and Assurance. The research results (i.e., papers) are categorized in relation to their contribution to the research challenges of this workpackage, described earlier in Sections 2.1.2–2.1.3. In accordance to the objective of the deliverable, the research results mainly focus on the proactive quality negotiation and assurance. Giving to the SBA the capability of an automatic negotiation and to estimate possible future failures is an important enabling factors for the SBA adaptation. For this reason many results presented here are also tightly related to research challenges pursued in WP-JRA-1.2 (Monitoring and Adaptation).

Sections 3.2.1–3.2.12 now provide the descriptions of the research results according to the standard structure described above.

3.2.1 “Usage-Based Online Testing for Proactive Adaptation of Service-Based Applications” [45]

Context and Background: Service-orientation is increasingly adopted as a paradigm for building highly dynamic, distributed and (self-)adaptive software systems, called service-based (or service-oriented) applications (SBAs). An SBA is realized by composing individual software services. In contrast to a software component, for the service composer (aka. service consumer) a software service is not an individual piece of software. Instead, the service consumer can only access the functionality and quality provided by that piece of software via the services’ interface. There is a clear trend that in the future SBAs will increasingly be composed from third-party services that are accessible over the Internet. SBAs based on third-party services allow taking the concept of ownership to the extreme: not only is the development, quality assurance, and maintenance of the software under the control of third parties, but the software itself is also operated and managed by them. This scenario implies a fundamental change to how software is developed, deployed, and maintained. An SBA cannot be specified and realized completely in advance (i.e, during design-time) due to the incomplete knowledge about the third-party services as well as the system’s context and communication infrastructure. Thus, compared to traditional software systems, much more decisions need to be taken during the operation of the SBA (i.e., after it has been deployed), once the missing knowledge is available. One specific problem that needs to be faced in that setting is that third-party services can change or evolve in ways not anticipated by the service consumer. For instance, a service can become unavailable, or can reply too slow due to network latencies or overload at the provider’s side. This means that SBAs need to dynamically adapt to such failures during run-time to ensure that they maintain their expected functionality and quality. Ideally, the need for an adaptation is proactively identified, i.e., failures are predicted before they can lead to consequences such as costly compensation and roll-back activities.

Problem Statement: Key to proactive adaptation is the ability to predict the future quality of the SBA and its constituent services. Typically, monitoring is used to assess the quality of the SBA and its constituent services during their operation. Based on monitoring data, failures are predicted and thus the
### Quality Negotiation

#### 3.2.1 Usage-Based Online Testing for Proactive Adaptation of Service-Based Applications

- Paper Title: [45]
- Algorithm: ✓
- Quality Assurance: ✓
- Also Contributed to: JRA-1.2, JRA-2.2

#### 3.2.2 Towards Accurate Failure Prediction for the Proactive Adaptation of Service-oriented Systems

- Paper Title: [27]
- Algorithm: ✓
- Quality Assurance: ✓
- Also Contributed to: JRA-1.2, JRA-2.2

#### 3.2.3 Accurate Service Failure Prediction through Online Testing

- Paper Title: [46]
- Algorithm: ✓
- Quality Assurance: ✓
- Also Contributed to: JRA-1.2, JRA-2.2

#### 3.2.4 Proactive SLA Negotiation for Service Based Systems: Initial Implementation and Evaluation Experience

- Paper Title: [25]
- Algorithm: ✓
- Also Contributed to: JRA-1.2

#### 3.2.5 SALMonADA: A platform for Monitoring and Explaining Violations of WSAAgreementcompliant Documents

- Paper Title: [24]
- Algorithm: ✓
- Quality Assurance: ✓
- Also Contributed to: JRA-1.2

#### 3.2.6 Preventing Performance Violations of Service Compositions using Assumption-based Run-time Verification

- Paper Title: [47]
- Algorithm: ✓
- Quality Assurance: ✓
- Also Contributed to: JRA-1.1

#### 3.2.7 Future Internet Apps: The Next Wave of Adaptive Service-Oriented Systems?

- Paper Title: [29]
- Algorithm: ✓
- Quality Assurance: ✓
- Also Contributed to: JRA-1, JRA-2

#### 3.2.8 Adaptive Future Internet Applications: Opportunities and Challenges for Adaptive Web Services Technology

- Paper Title: [26]
- Algorithm: ✓
- Quality Assurance: ✓
- Also Contributed to: JRA-1.2, JRA-2.2

#### 3.2.9 SLAs for Cross-layer Adaptation and Monitoring of Service-Based Applications: A Case Study

- Paper Title: [5]
- Algorithm: ✓
- Quality Assurance: ✓
- Also Contributed to: JRA-1.1, JRA-1.2, JRA-2.1, JRA-2.2, JRA-2.3

#### 3.2.10 Negotiation towards Service Level Agreements: A Life Cycle Based Approach

- Paper Title: [13]
- Algorithm: ✓
- Also Contributed to: JRA-1.1, JRA-1.2

#### 3.2.11 A Context-Aware Framework for Business Process Evolution

- Paper Title: [6]
- Algorithm: ✓
- Also Contributed to: JRA-1.1, JRA-1.2, JRA-2.1

#### 3.2.12 Prediction of SLA Violations in Service Orchestrations

- Paper Title: [16]
- Algorithm: ✓
- Also Contributed to: JRA-2.1, JRA-2.2

<table>
<thead>
<tr>
<th>Section</th>
<th>Paper Title [reference]</th>
<th>Quality Negotiation</th>
<th>Quality Assurance</th>
<th>Also Contributes to Workpackage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>Usage-Based Online Testing for Proactive Adaptation of Service-Based Applications</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>JRA-1.2, JRA-2.2</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Towards Accurate Failure Prediction for the Proactive Adaptation of Service-oriented Systems</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>JRA-1.2, JRA-2.2</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Accurate Service Failure Prediction through Online Testing</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>JRA-1.2, JRA-2.2</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Proactive SLA Negotiation for Service Based Systems: Initial Implementation and Evaluation Experience</td>
<td>✓ ✓</td>
<td></td>
<td>JRA-1.2</td>
</tr>
<tr>
<td>3.2.5</td>
<td>SALMonADA: A platform for Monitoring and Explaining Violations of WSAAgreementcompliant Documents</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>JRA-1.2</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Preventing Performance Violations of Service Compositions using Assumption-based Run-time Verification</td>
<td>✓ ✓</td>
<td></td>
<td>JRA-1.1</td>
</tr>
<tr>
<td>3.2.7</td>
<td>Future Internet Apps: The Next Wave of Adaptive Service-Oriented Systems?</td>
<td>✓ ✓</td>
<td></td>
<td>JRA-1, JRA-2</td>
</tr>
<tr>
<td>3.2.8</td>
<td>Adaptive Future Internet Applications: Opportunities and Challenges for Adaptive Web Services Technology</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>JRA-1.2, JRA-2.2</td>
</tr>
<tr>
<td>3.2.9</td>
<td>SLAs for Cross-layer Adaptation and Monitoring of Service-Based Applications: A Case Study</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>JRA-1.1, JRA-1.2, JRA-2.1, JRA-2.2, JRA-2.3</td>
</tr>
<tr>
<td>3.2.10</td>
<td>Negotiation towards Service Level Agreements: A Life Cycle Based Approach</td>
<td>✓ ✓</td>
<td></td>
<td>JRA-1.1, JRA-1.2</td>
</tr>
<tr>
<td>3.2.11</td>
<td>A Context-Aware Framework for Business Process Evolution</td>
<td>✓ ✓</td>
<td></td>
<td>JRA-1.1, JRA-1.2, JRA-2.1</td>
</tr>
<tr>
<td>3.2.12</td>
<td>Prediction of SLA Violations in Service Orchestrations</td>
<td>✓ ✓</td>
<td></td>
<td>JRA-2.1, JRA-2.2</td>
</tr>
</tbody>
</table>

Table 3.1: Coverage of Research Challenges by Research Results ((*) = submitted, (+) = accepted)
need for adaptation is identified. However, monitoring only observes services or SBAs during their actual use in the field. Due to its “observational” or “passive” nature, monitoring does not guarantee a comprehensive coverage of the “test object”, i.e., monitoring might not cover all relevant service executions. This can diminish the precision of failure prediction, i.e., the ability to correctly predict deviations in expected functionality or quality. To address the shortcomings imposed by the “passive” nature of monitoring, researchers have suggested performing test activities during the operation of the SBA. Such an online testing means that the constituent services of an SBA are systematically tested in parallel to the normal use and operation of the SBA. Thus, “online testing” is sometimes called “active monitoring” in the literature. In online testing, like in traditional testing, we face the problem of determining when, how and how much to test. However, in answering those questions we need to address the following two requirements for online testing, which are imposed by the key differences of SBAs from traditional software systems:

- Services need to be (re-)tested periodically in order to determine failures, because third-party services can change without notice. This need for periodically retesting is significantly different from traditional software systems. A tester of a traditional software system is aware of the changes of software components and only needs to run regression tests after such changes. Furthermore, a tester of a traditional software system has control over the environmental conditions (test environment) during testing and thus, if a test has passed/failed, it will pass/fail again for a later invocation of the same version of the component/system.

- The number of online tests needs to be limited due to economic and technical considerations. This limitation in the number of tests is different from traditional software systems. The number of online tests can affect the provisioning of the service and thus could impact performance for example. Furthermore, testing costs can become a limiting factor as the software components that provide the service are not owned by the “testing” organization and the service provider might charge “per use” of the service. Finally, the number of times that a user is allowed to invoke a service can be limited by service contracts.

**Relevance of Problem and Progress from State of the Art:** Recent trends in SOC research show emphasis on proactive adaptation for SBAs, and quality prediction of SBAs has become a vivid research area. Our approach progresses from the state of the art by devising and evaluating a novel test case selection technique that exploits synergies between monitoring and usage-based testing in order to increase the precision of failure prediction and thus proactive adaptation.

**Relation to WP Challenges:** This work addresses the research challenges of “Quality Prediction Techniques to Support Proactive Adaptation” and “Run-time Quality Assurance Techniques”, as the prediction occurs during run-time.

**Solution / Research Method:** In this work we address the above problems and introduce a novel online testing technique that provides enhanced proactive adaptation capabilities to SBAs. More specifically, the work provides the following major contributions:

- A framework for proactive adaptation, which exploits synergies between monitoring, online testing and quality prediction. The framework’s core element is a test selection activity that utilizes information about the usage of the SBA’s services in order to select test cases that lead to better coverage of service executions, while utilizing a limited number of online test executions.

- Prototypical implementation of the proposed framework based on an existing monitoring framework with components to collect and report monitoring information, and to execute usage-based online tests.
Simulation and experimental assessment of the proposed techniques and framework, in order to evaluate and analyze the improvements of our approach on the precision of failure predictions for the case of performance testing, i.e., focusing on response time as a QoS attribute.

Benefits and Evaluation: The complementary use of online testing & monitoring (in our approach) improves the precision of failure prediction (i.e., the ability to correctly predict violations in the expected service quality) when compared to monitoring in isolation. Ultimately, this means that an SBA furnished with the complementary approach will have better proactive adaptation capabilities.

To assess the above benefit of our approach, we ran an experiment which is based on (1) the simulation of an example SBA and its associated services, together with (2) the prototypical implementation of the online test case selection, execution and prediction components of our proactive adaptation framework.

Relation to Research Framework: The approach, from a mechanisms point of view, focuses on the Service Composition and Coordination layer (JRA-2.2), as individual services are monitored and tested to determine failures and deviations. In addition, the approach is relevant to Monitoring and Adaptation (JRA-1.2) as the combined monitoring and testing data is used for making precise failure prediction to support proactive adaptation.

Discussion and Future Work: A quality prediction framework for proactive adaptation of SBAs that exploits usage-based online testing has been presented. The framework exploits synergies between monitoring, online testing and quality prediction to assure better coverage of service executions, and thus, enables more precise prediction of adaptation triggers. The framework relies on a test case selection component which exploits information about the usage of an SBA's services. This includes techniques to determine the number of test cases to be executed. Furthermore, we introduced our prototypical implementation of the framework with components to collect and analyze monitoring information (including usage frequencies), and to determine and execute usage-based online tests. Finally, we presented the results of an experiment we conducted which showed that the complementary use of testing and monitoring, as advocated by our framework, improves the precision of failure prediction (i.e., the ability to correctly predict violations of expected service quality) when compared to using monitoring in isolation.

Several issues remain open for future work. For example, we plan to use more advanced prediction techniques in order to enhance the prediction and to further reduce the number of required tests that are needed. We will also consider other different metrics and cost models that would, for example, relate the cost of testing to the cost of compensation activities of wrong adaptations. Furthermore, we plan to run live experiments with executions of SBAs and test cases in order to more realistically assess our approach.

3.2.2 “Towards Accurate Failure Prediction for the Proactive Adaptation of Service-oriented Systems” [27]

Context and Background: Service-orientation is increasingly adopted as a paradigm for building highly dynamic, distributed service-oriented systems. A service-oriented system is realized by composing individual software services. In contrast to a software component, not only the development, quality assurance, and maintenance of the software can be under the control of third-parties, but the software can also be executed and managed by third-parties.

There is a clear trend that in the emerging “Future Internet”, service-oriented applications will be increasingly composed of third-party services accessible over the network. As a consequence, the capabilities and quality of service-oriented systems more and more will depend on the quality of their third-party services.
In particular, this means that service-oriented systems will need to become resilient against failures of their third-party services. As a simple example, a service might become unavailable due to overloads on the service provider’s side. Furnishing service-oriented systems with self-adaptation capabilities is considered a key solution to address this challenge.

**Problem Statement:** To trigger the proactive adaptation of a service-oriented system, pending failures need to be predicted. It is important that such a failure prediction is accurate, such as to avoid the execution of unnecessary proactive adaptations, as well as not to miss proactive adaptation opportunities.

Unnecessary adaptations can have the following severe shortcomings: Firstly, unnecessary adaptations can be costly. For instance, additional activities such as Service Level Agreement (SLA) negotiation for the alternative services might have to be performed, or the adaptation can lead to a more costly operation of the service-oriented system, e.g., if a seemingly unreliable but cheap service is replaced by a more costly one. Secondly, unnecessary adaptations could be faulty (e.g., if the new service has bugs), leading to severe problems as a consequence. Thirdly, as executing the adaptation takes time, this means that in the worst case, an unnecessary adaptation will leave less time to address actual failures.

In case an adaptation opportunity is missed due to inaccurate failure predictions, this obviously can lead to the same shortcomings as faced in the setting of reactive adaptations, i.e., it can require compensation or costly repair activities. This means, inaccurate predictions would diminish the benefits of proactive adaptation.

Providing accurate failure predictions is extremely challenging in the setting of service-oriented systems, if third-party services are present. The observed quality and functionality of those third-party services can significantly vary between different service invocations. For instance, the performance of a third-party service might depend on the load of the infrastructure at the provider’s side or the network latency, if services are offered over the Internet. As an example, a failure observed at one point in time (e.g., unavailability of a service because of an overload at the service provider side) can disappear at a later point in time (e.g., the same service is now executed because of a lower load at the service provider side).

**Relevance of Problem and Progress from State of the Art:** Several quality prediction approaches exist in the literature. Each approach works differently, in different settings and with different assumptions; e.g., machine learning works well once a significant amount of training data has been collected. Still, all of those techniques share the same concern: they need to accurately predict the future quality, resp. future failures, of a service-oriented system.

**Relation to WP Challenges:** This work addresses the research challenges of “Quality Prediction Techniques to Support Proactive Adaptation” and “Run-time Quality Assurance Techniques”, as the prediction occurs during run-time.

**Solution / Research Method:** This work introduces two directions along which accurate failure predictions for proactive adaptation could be established; firstly, by improving the prediction techniques themselves; secondly, by dynamically estimating the accuracy during run-time. Based on selected prediction techniques from the literature and metrics to assess the accuracy of predictions, those two directions for achieving accuracy are critically discussed.

**Benefits and Evaluation:** The discussions in this work are backed by results from experiments which are based on: (1) simulation of an example service-oriented system and its associated third-party services, together with (2) a prototypical implementation of selected quality prediction techniques.
Relation to Research Framework: The approach, from a mechanisms point of view, focuses on the Service Composition and Coordination layer (JRA-2.2), as individual services are monitored and tested to determine failures and deviations. In addition, the approach is relevant to Monitoring and Adaptation (JRA-1.2), as the combined monitoring and testing data is used for making precise failure prediction to support proactive adaptation.

Discussion and Future Work: Failure prediction (or quality prediction) techniques are key to engineer service-oriented systems with proactive adaptation capabilities. However, as discussed previously, those predictions have to be accurate, as – for instance – false predictions can lead to additional operational costs and severe failures.

Research in the field has produced a diverse range of prediction techniques that are applicable in different settings and have various benefits and shortcomings. Of course, one could strive to design techniques that provide high accuracy for a known setting.

However, if we consider the highly dynamic nature of service-oriented systems in the “Future Internet”, even a failure prediction technique which provided good accuracy for known settings can quickly become “obsolete”, as it will be highly probable that new, unforeseen settings will dynamically arise.

Ultimately, this suggests to further investigate quality prediction techniques that also provide capabilities to dynamically assess their accuracy during runtime.

3.2.3 “Accurate Service Failure Prediction through Online Testing” [46]

Context and Background: Web-based services provide unprecedented opportunities to build highly flexible systems by integrating service offerings from third parties. However, service consumers and integrators have limited control over third-party web-based services. Those services thus may behave in ways not anticipated during design time, leading to failures during run-time. For example, those services may exhibit a degradation of quality of service (QoS), such as reduced performance or low reliability.

Online failure prediction of third-party services allows anticipating degradations in expected QoS (e.g., as stipulated in SLAs). Online failure prediction thus allows, for instance, to plan and implement proactive repair or compensation activities. Several online failure prediction techniques have been presented in literature. All these techniques all rely on monitoring of QoS data to predict failures.

Monitoring only passively observes services during their actual use. The amount and timeliness of QoS data collected by monitoring thus may be limited; e.g., when a service is only seldom invoked by users. Sparse monitoring data may undermine the accuracy of failure predictions. On the one hand, this may lead to false positive predictions, which imply unnecessarily switching to an expensive service, delays due to unnecessary repair activities, or replacing a working service with another one that has severe defects. On the other hand, this may lead to false negative predictions, which mean that the opportunity for proactive repair and compensation may be missed altogether.

Problem Statement: Online testing has been proposed as an active quality assurance technique to complement passive monitoring. Online testing means that constituent services of a service-oriented system are systematically tested in parallel to its normal use. QoS data of tested services can thus be used to augment QoS data from monitoring. Our previous work indicates that such augmented data improves accuracy of failure predictions of web-based services.

However, online testing implies additional costs for operating software systems; e.g., when online testing a pay-per-use service. To become applicable in practice, one has to understand when online testing pays off with respect to improvements in accuracy gains.

Different factors can impact accuracy gains achieved by online testing. For example, gains in accuracy diminish as more frequent monitoring data is available. After a certain point, online testing thus may not pay off.
Relevance of Problem and Progress from State of the Art: In general, failure prediction approaches rely on QoS data from monitoring. However, due its “passive” nature, the amount and timeliness of the collected data may be limited, which may undermine the accuracy of failure predictions. Our proposed framework, which is called PROSA, complements such approaches by collecting timely QoS data using online testing.

In the literature, several techniques and platforms exist that advocate to perform online testing periodically or event-driven, but only for runtime quality assurance. However, exploiting online testing for online failure prediction has not been addressed.

For combining online testing with monitoring, two main directions can be observed in the literature. Firstly, using monitoring data to build the usage profiles or usage models. However, the direction was proposed for test case definition or offline testing and not for triggering online tests.

Secondly, monitoring data was exploited to reduce the number of service invocations when executing a test suite (i.e., reducing the cost of testing). Approaches used this direction to mimic the service responses. However, they do not complement monitoring data with online testing for failure prediction.

Relation to WP Challenges: This work addresses the research challenges of “Quality Prediction Techniques to Support Proactive Adaptation” and “Run-time Quality Assurance Techniques”, as the prediction occurs during run-time.

Solution / Research Method: This work presents the setup and results of extensive experiments to empirically assess how accuracy gains achieved by online testing depend on different factors, including online test rates, usage frequencies, failure prediction models, and failure rates. The experiments are based on QoS data of real-world web-based services.

Based on the experimental findings, this work introduces an extension of an existing Web Services monitoring framework. The extended framework can be parameterized such that online tests are only triggered when the costs of online testing may pay off with respect to accuracy gains. The conceptual design of the framework and the technical implementation of its extended monitoring engine, are described in this work.

Benefits and Evaluation: As already mentioned, the influential factors are analyzed based on extensive experiments using real Web services. The results clearly indicate that service usage frequency, online test rates, prediction model, and service failure rates have clear influence on the accuracy gains achieved by online testing.

Consequently, this work presents a framework which can be parameterized to trigger online tests if the costs of online testing may pay off with respect to accuracy gains in failure predictions.

Relation to Research Framework: The approach, from a mechanisms point of view, focuses on the Service Composition and Coordination layer (JRA-2.2), as individual services are monitored and tested to determine failures and deviations. In addition, the approach is relevant to Monitoring and Adaptation (JRA-1.2), as the combined monitoring and testing data is used for making precise failure prediction to support proactive adaptation.

Discussion and Future Work: Online testing promises to improve the accuracy of online failure prediction by complementing passive monitoring. However, online testing can incur additional costs for operating software systems. Understanding when online testing pays off with respect to accuracy gains is thus an important issue for the applicability of online testing. To this end, this work has presented the results of extensive experiments conducted to empirically assess how accuracy gains from online testing depend on different factors, including online test rates, usage frequencies, and failure prediction models.

Based on the experimental findings, an extension of an existing monitoring framework for Web Services
was introduced, which is called PROSA. PROSA can be parameterized to trigger online tests when the costs of online testing may pay off (depending on the application settings) with respect to accuracy gains in failure predictions.

Future work includes using PROSA to complement approaches for predicting failures for a service-oriented system or a composite service.

### 3.2.4 “Proactive SLA Negotiation for Service Based Systems: Initial Implementation and Evaluation Experience” [25]

**Context and Background:** Service Level Agreements (SLA) define quality of service (QoS) and functional properties, which should be guaranteed during the provision of a software service, as well as the penalties that should be applied in case the properties are not fulfilled. An SLA is set through a negotiation between the provider and the consumer of a service. SLA negotiation can be particularly complex depending on the requirements and affordances of the two parties. Furthermore, it may need to be carried out at runtime, if a constituent service of a service based system (SBS) becomes unavailable whilst SBS is in operation, or it fails to perform according to its established SLA. There are a number of possible scenarios that may lead to the violation of an SLA. More specifically, a SLA may be violated due to, i) poor QoS delivered by a participating service, ii) delayed delivery of service by a participating service, iii) unavailable service or resource, iv) change of requester’s circumstances, e.g. requester needs service at better level than the agreed level after the SLA has been agreed and v) change of provider’s circumstances, e.g. provider suffers from peak demand of services from its requesters. In such cases, an SBS should be able to discover alternative replacement services for the failed service, and negotiate SLAs with their providers at runtime.

**Problem Statement:** To minimize the runtime interruption of the SBS, the discovery of replacement services for the SBS’s constituent ones should be proactive, i.e., it should be performed before a constituent service of SBS becomes unavailable or fails to perform according to its established SLA. Proactiveness is important since service discovery is a time consuming activity and, therefore, carrying it in a reactive mode, is likely to cause significant interruption in the provision of the composite service and violations of its own SLA. SLA negotiation should also be proactive, as it will be necessary to have adequate SLAs for the potential replacement services that have been identified by proactive discovery, while attempting SLA negotiation just prior to binding to an alternative service is likely to cause significant delay.

**Relevance of Problem and Progress from State of the Art:** Existing work on service level agreements has focused on SLA specification, negotiation and monitoring. The need for runtime SLA negotiation or re-negotiation has also been addressed in the literature, where either the terms of an SLA are revised to accept a constituent service from an existing provider or a new SLA is negotiated with a new service provider and an existing SLA is terminated. All these approaches, however, are reactive as they support corrective actions only after an SLA has been violated. Thus they may fail to guarantee uninterrupted runtime provision of composite services. To address the above shortcomings, a framework was developed for integrating proactive SLA negotiation with dynamic service discovery in order to provide cohesive runtime support for both these activities. The proactive negotiation of SLAs, as part of service discovery is necessary for reducing the extent of interruptions during the operation of a SBS, when the need for replacing services in SBS arises.

**Relation to WP Challenges:** This work addresses the research challenge Proactive SLA Negotiation and Agreement and also partially addresses the challenges Adaptation and Monitoring and Discovery and Registry Infrastructure (defined in WP-JRA-2.3) as we argue about the combination of service discovery and monitoring to facilitate the proactive SLA negotiation.
**Solution Research Method:** In this work we have produced a framework called PROSDIN (PROactive Service DIscovery and Negotiation). In this framework, we have developed a proactive runtime SLA negotiation tool, and integrated it with a tool supporting proactive runtime service discovery. More specifically, in PROSDIN, SLA negotiation has been developed as an integrated part of the service discovery process, enabling the execution of both activities in a coordinated manner. Proactive SLA negotiation is performed immediately after the execution of service discovery queries to ensure that adequate SLAs are provisionally agreed for given periods of time with the providers of the discovered services, if possible. Also when a pre-agreed SLA expires, it is proactively re-negotiated.

The service discovery tool in PROSDIN is used to identify candidate services that could potentially be used by the SBS. Service discovery is based on queries that express conditions about the interface, behaviour, contextual and quality characteristics of services. To use PROSDIN, each of the constituent SBS services, which is replaceable at runtime, should be associated with a discovery query which, specifies the conditions for discovering services that could potentially replace this constituent service. These queries should be specified by the developer of the SBS during the SBS development, and passed to PROSDIN by the SBS at runtime in order to be executed when service failures occur and enable service discovery. Following the subscription of such queries for a constituent service $S_c$ of an SBS, PROSDIN executes them proactively and in parallel with the execution of the SBS, and stores the services that match them in an external registry to maintain an up-to-date set of candidate replacement services for $S_c$.

The negotiation tool in the PROSDIN manages the negotiation process on behalf of service client applications. In particular, it provides access to different negotiation engines that may be plugged into the framework by translating negotiation rules expressed in the common language of the framework, into the different negotiation specifications accepted by these engines, and realizes the interface for interacting with broker which, carry out the negotiation process on behalf of services. These brokers may be the same as the broker used by PROSDIN or other brokers that realize the same SLA negotiation interaction interface with it. The negotiation process is carried out according to a two-phase protocol that may result either in a pre-agreed but not activated SLA or fail. Pre-agreed SLAs have an expiry period within which they can become active, if the service client application decides to activate them.

**Benefits and Evaluation:** We performed a series of experiments to evaluate the implementation of the PROSDIN framework. The purpose of these experiments was to: (a) measure the overhead of SLA negotiation (whether reactive or proactive) on the execution time of the runtime service discovery process, and (b) assess the effectiveness of proactive SLA negotiation over reactive SLA negotiation during runtime service discovery process. In the experiments, we have used an SBS implemented as a BPEL service orchestration process. The service discovery query used in the experiments was specified in order to identify candidate replacement services for a constituent service of the SBS. In the experiments, we also used an SLA template with four QoS terms for negotiation. For negotiation, we specified a set of 15 service consumer negotiation rules (CNR set), and 20 different sets of provider negotiation rules (PNR sets). During negotiation with each of the candidate services identified by the discovery process, the negotiation broker of the service provider side picked up randomly one of the PNR sets and carried out the negotiation based on it. To assess whether the number of considered services affects the performance of the service discovery and SLA negotiation processes, we performed the experiments with three different service sets (registries). These sets contained 100, 300 and 500 services, respectively, and were populated with appropriate service specifications.

As shown in Table 3.2, the time required to select a replacement service in case of service discovery with proactive SLA negotiation is slightly larger than the time required to identify a replacement service in case of service discovery without any SLA negotiation is used. This is because in the former case, the pre-agreed SLA needs to be activated before the replacement service is returned. It should be noted, however, that the main benefit shown in the table is that the time required to select and bind a replacement service at runtime in the case of service discovery with reactive SLA negotiation is significantly larger than the service selection and binding time in the case of service discovery with proactive SLA.
Table 3.2: Proactive SLA Negotiation Evaluation Results

<table>
<thead>
<tr>
<th></th>
<th>SD Only</th>
<th>SD with Proactive SLA</th>
<th>SD with Reactive SLA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>300</td>
<td>500</td>
</tr>
<tr>
<td>Replacement Service</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Maintenance</td>
<td>690.5</td>
<td>698.3</td>
<td>673.5</td>
</tr>
<tr>
<td>(avg time in ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection of Replacement</td>
<td>22</td>
<td>28</td>
<td>21.8</td>
</tr>
<tr>
<td>Service (avg time in ms)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relation to Research Framework: The approach presented in this paper focuses on the proactive SLA negotiation (JRA-1.3). Moreover, this approach performs service discovery (JRA-2.3) and the monitoring of agreed SLAs to enable proactive negotiation that facilitates the proactive adaptation of a service based system (JRA-1.2).

Discussion and Future Work: The objective of proactive SLA negotiation in PROSDIN is to ensure that a service, which could be potentially used by a service client application, will have an agreed set of guaranteed provision terms, if the need to deploy it arises at runtime. Hence, when this need arises, it will not be necessary to engage in a lengthy negotiation process interrupting the operation of the service client application. Our approach has been evaluated through an initial set of experiments showing that proactive SLA negotiation leads to significant reduction of the time required to perform service replacement at runtime, if the existence of agreed SLAs is a prerequisite for service use. PROSDIN opens a spectrum of possible lines for future investigation. These include support for proactive negotiation of hierarchical SLAs, i.e., SLAs of complex composite services deploying other composite services with their own sub-SLAs which will need to be negotiated separately and before coming to a higher level service level agreement. Also the framework can be extended to support dynamic adaptation of the negotiation rules, i.e., the participants will be able to dynamically change the negotiation rules during the negotiation process.

3.2.5 “SALMonADA: A platform for Monitoring and Explaining Violations of WS Agreement compliant Documents” [24]

Context and Background: Service Level Agreements (SLAs) establish the service quality between consumers and providers of service-based systems (SBS), so there is a need for monitoring techniques to control the SLA fulfillment. However, monitoring has been signaled as a cornerstone from the raising of the first SLA specifications to the more recent ones, such as WS–Agreement. Many research efforts have been made to provide information about the service level fulfillment of SLAs in order to adopt them in B2B or B2C scenarios.

Problem Statement: Once the agreement has been established between the SBSs consumers and providers, techniques to assure the agreed service quality must be developed. Such techniques require both: monitoring platforms to achieve information while the service is being consumed, and analysis engines to reason about the monitored information in order to extract useful information. Such service level fulfillment information must be clearly exposed to the parties to help in the problem solutions.
Relevance of Problem and Progress from State of the Art: The aforementioned problem has been partially covered by other works. Several proposals can be found providing violation detection for SLAs described in WS–Agreement but in the SOA testing context, and not at service consuming time. Other proposals provide asynchronous violation detection reports to subscribed clients but they deal with ad-hoc SLA specifications. There are also proposals that go further and when they detect a SLA violation, they dynamically adapt the SBSs following different strategies. Finally, there are proposals that are able to detect and explain violation causes but with the following drawbacks: (1) the analysis of the SLA fulfillment is not performed just when a violation takes place; (2) the client (end-user) must be an expert in the reasoning paradigm to specify the SLA, but also to understand the violation explanations; (3) the violation cause is difficult to grasp by the client (application or end-user), especially when several service properties are related in the violated SLA term.

Relation to WP Challenges: This paper addresses two key objectives of the workpackage. The proposed approach addresses the end-to-end quality assurance at run-time, as well as the SLA conformance. To this aim, we have introduced an automated framework for monitoring and analysing SLAs.

Solution / Research Method: We propose SALMonADA, a service-based system (SBS) with a decoupled component architecture that integrates monitoring (SALMon) and analysis (ADA) components supporting: (1) asynchronous service monitoring of WS–Agreement documents to analyse the SLA fulfillment and report a violation when the event that causes it has been monitored (reducing the client notification time), (2) violation analysis to detect and explain the violation causes, and finally (3) notifying the clients in their own easy-to-understand specification terms. Other technical contributions are: decoupling of the monitored service technology by using an enterprise service bus (ESB); the scalability control by adding more ESBs when needed; and the use of a monitoring management document to store the monitoring information. Such a document is updated when new monitoring information is achieved.

Benefits and Evaluation: Those proposals, which assume the availability of a monitoring and analysis engine, benefit from using SALMonADA since they are provided with such a service level fulfillment information needed to perform various important activities, such as SBS adaptation, SLA renegotiation, and reputation statistics derivation. For demonstration purposes, we have implemented a web application as a SALMonADA client in order to specify or upload the WS-Agreement documents to monitor, execute SALMonADA and receive the results. In this web application, we have introduced the WS-Agreements of ADA and SALMon. By monitoring the SLAs of these services, we are able to assess both the functionality of SALMonADA and the non-functional aspects of its main components. Moreover, as part of the demonstration, we have simulated the consumers of the service of ADA and SALMon to prove how SALMonADA monitors, analyses and reports the service level fulfillment to their clients.

Relation to Research Framework: The JRA-1.2 is also closely related with our work since the SBS adaptation is one application of the service level fulfillment analysis reported by our proposal.

Discussion and Future Work: The conceptual model of our proposed monitoring and analysis framework is included in the paper, showing one of its instantiation by a decoupled architecture model supporting any monitoring and/or analysis component inside. We have developed such an architecture model with monitoring and analysis capabilities of previous authors proposals SALMon and ADA, respectively. Moreover, a web application client for the framework has been implemented to demonstrate the asynchronous capabilities of our proposal to monitor, analyse, and ultimately notify the service level fulfillment to the client. However, some aspects are out of the scope of this work such as the use of some temporal analysis capabilities of ADA that are currently not considered in our SALMonADA framework.
3.2.6 “Preventing Performance Violations of Service Compositions using Assumption-based Run-time Verification” [47]

**Context and Background:** Service-orientation is increasingly adopted as a paradigm to build highly dynamic, distributed applications from individual software entities, offered as services. In this work we refer to such applications as Service-based Applications, or SBAs for short. There is a clear trend that future SBAs will be increasingly composed from third-party services that are accessible over the Internet. As a consequence, SBAs will increasingly depend on the functionality and quality offered by those third parties. To prevent menacing requirements violations, SBAs should be equipped with monitoring, prediction and adaptation capabilities which are able to foresee and avert menacing violations.

**Problem Statement:** In an emergency situation, reactive adaptation can lead to critical situations, as it might delay the timely dispatch of operational forces, for example fire engines or ambulances. To address these problems, researchers have proposed to employ preventive adaptation, which enables SBAs to predict future failures and perform preventive actions. Although several approaches for preventive adaptation have been presented in the literature, they pose certain limitations, such as the need for cost models or comprehensive training data.

**Relevance of Problem and Progress from State of the Art:** Researchers have proposed to employ preventive adaptation, which enables SBAs to predict future failures and perform preventive actions. Although several approaches for preventive adaptation have been presented in the literature, they pose certain limitations, such as the need for cost models or comprehensive training data. This work aims at addressing these limitations. For critical application domains (such as emergency or financial) and important customers (such as key accounts), the SBA developer needs to ensure that each individual SBA instance will live up to its expected requirements even though its constituent, third-party services might fail.

**Relation to WP Challenges:** “Run-time Quality Assurance Techniques” and “Online Quality Prediction Techniques to Support Proactive Adaptation”, as the prediction occurs during run-time.

**Solution / Research Method:** SPADE equips SBAs with adaptation capabilities, empowering them to adapt themselves preventively. To achieve this, SPADE uses run-time verification techniques, execution data of the monitored instances and assumptions concerning the SBAs’ context, derived from Service Level Agreements (SLAs) of third-party services. Together, these mechanisms are used for performance prediction, which is able to detect menacing performance requirements violations of running SBAs. SPADE can thus be used in settings where no cost models or training data are available.

**Benefits and Evaluation:** Our measurement of SPADE’s efficiency is twofold. First, we examined unnecessary adaptations, i.e., false positives, such as adaptations that might lead to avoidable costs, e.g., when replacing a free service with a commercial service to compensate for faults. Secondly, we count the amount of situations in which SPADE cannot perform an adaptation. It can happen that a service invocation leads to a violation of an SLA, such that the end-to-end requirement is already violated. In those situations, the SBA instance obviously cannot be adapted preventively in order to avert this requirement violation, as the requirement has already been violated. Both values are expected to be low, as a low value implies a high number of cases where SPADE was successfully applied. The conducted experiments show that SPADE actually has a small amount of false positives. Also SPADE run in just a few situations where adaptations were not possible.
Relation to Research Framework: The approach focuses on the Service Composition and Coordination layer (JRA-2.2), as individual services are monitored and runtime checks are performed to determine violations of e-2-e requirement. In addition, the approach is relevant to Monitoring and Adaptation (JRA-1.2), as it exploits monitoring data together with assumptions for predicting e-2-e requirement violations and performing preventive adaptation if needed.

Discussion and Future Work: We plan to continue our work on preventive adaptation in two directions. First, we will combine SPADE with our PROSA approach. The PROSA approach is capable of predicting quality violations of individual services. The combined approach is expected to act in situations in which SPADE is not able to prevent requirements violations as intended. Secondly, we plan to apply SPADE in a cross-layer adaptation setting. In this setting, SPADE is expected to exploit the adaptation mechanisms of two different layers: the service composition and the service infrastructure layer. We expect that harmonizing the adaptation on both layers will increase the number of situations in which SPADE is able to compensate for deviations, which thus may increase SPADE’s success in avoiding requirements violations.

3.2.7 “Future Internet Apps: The Next Wave of Adaptive Service-Oriented Systems?” [29]

Context and Background: The Future Internet will emerge through the convergence of software services, things, content, and communication networks. Service orientation is expected to play a key role as an enabling technology that allows the provisioning of hardware and software entities and contents as services. The dynamic composition of such services will enable the creation of service-oriented systems in the Future Internet (from now on such systems are called FI Apps), which will be increasingly provided by third parties. Together with increased expectations from end-users for personalization and customization, FI Apps will thus face an unprecedented level of change and dynamism.

Problem Statement: The capabilities and features of FI Apps will be increasingly provided and “owned” by third parties. Examples include Internet-based software services, public sensor networks, and cloud infrastructures. Due to this “shared ownership”, FI Apps will face an unprecedented level of change and dynamism. Further, expectations from end-users for what concerns the personalization and customization of those FI Apps are expected to become increasingly relevant for market success. For instance, a FI App should be able to adapt depending on the usage setting (e.g., office vs. home) or based on the available communication infrastructure (e.g., sensors vs. WiMAX). It will thus become increasingly important to engineer FI Apps in such a way that those applications can dynamically and autonomously respond to changes in the provisioning of services, availability of things and contents, as well as changes of network connectivity, end-user devices, and user expectations. Ultimately, this means that adaptation will become a key capability of FI Apps. Nevertheless, the characterization of the adaptation capabilities to be considered in FI Apps remains a topic not fully explored.

Relevance of Problem and Progress from State of the Art: There has been significant progress for what concerns principles and techniques for building adaptive service-oriented systems. However, if we consider the Future Internet setting, those solutions will need to be significantly augmented, improved and integrated with a complete systems perspective. Specifically, this requires significant progress towards novel strategies and techniques for adaptation, addressing key characteristics of adaptive FI Apps. To enable the next wave of adaptive service-oriented systems in the Future Internet, it will thus be critical to understand the importance of the various adaptation characteristics to specifically target research and development activities.

Relation to WP Challenges: This work is relevant to the research challenge: “Quality Prediction Techniques to Support Proactive Adaptation”, as it discusses how proactive adaptation triggered by predic-
tion is expected to gain increased importance for FI Apps.

**Solution / Research Method:** This work identifies and analyses key characteristics of adaptive FI Apps. Those characteristics are illustrated with examples from a scenario of the application domain of transport and logistics. The relevance of those different characteristics is scrutinized through an empirical study. As a research method we have employed an exploratory survey study, involving 51 respondents from the Future Internet community.

**Benefits and Evaluation:** We believe that the results of this survey study can enable understanding where adaptation can play a key role for FI Apps. The survey has confirmed some of the typical expectations (e.g., the importance of adaptation for service-oriented systems). However, the survey also lead to unexpected outcomes. In particular, distributed and cross-area adaptation capabilities have been deemed least important, although one would have expected those characteristics to become highly relevant in the Future Internet. We believe this deserves further investigation.

**Relation to Research Framework:** The discussion promoted in this work are related to the current elements of the S-Cube Research Framework and also with follow up issues in the scenario of Future Internet Application that consider the integration across JRA-1 and JRA-2: Monitoring and Adaptation, Engineering and Design, BPM, Service Composition and Coordination, Service Infrastructure.

**Discussion and Future Work:** One direction of future work is to better analyze the need for cross-cutting and decentralized characteristics on adaptive FI applications. We believe that there are at least two reasons to explain why these characteristics were deemed less important. First, FI areas are not consolidated enough and, currently, to the best of our knowledge, there are not proposals addressing the cross-area adaptation aspects in Web services and service-based applications, because so far it was not possible to put together IoS and IoT, for instance. But, as discussed in the beginning of this chapter, both cross-layer and cross-area are aspects related to cross-cutting adaptation characteristics in FI applications which are much deeper than the cross-layer ones in the service-oriented architecture. Second, decentralization and adaptation in the scope of Web services and service-based applications is currently associated with the very limited and self-owned environment. We believe that differently from the scope of Web services and service-based applications, FI applications assume a larger and more heterogeneous scale (e.g., Transport & Logistics applications have to deal with partners around the globe and interact with real-world devices - sensors, trucks). In this case, decentralized solutions for adaptive Web services and service-based applications proposed so far will not be suitable anymore.

3.2.8 “Adaptive Future Internet Applications: Opportunities and Challenges for Adaptive Web Services Technology” [26]

**Context and Background:** Adaptive capabilities are essential to guarantee the proper execution of Web services and service-oriented applications once dynamic changes are not exceptions but the rule. In fact, the importance of adaptive services significantly increases in the context of Future Internet (FI) applications. Applications in this context will have to autonomously adapt to changes on service provisioning, availability of things and content, computing resources, and network connectivity. The unprecedented level of heterogeneity and dynamic changes of FI applications will demand a transition from adaptive Web Services and service-oriented systems to adaptive FI applications.

Three major pillars constitute the FI: the Internet of Services (IoS, e.g., software services based on third-party services), the Internet of Things (IoT, e.g., smart sensors and devices) and the Internet of Content (IoC, e.g., video streams and online games). These pillars will all converge into an integrated environment. It is expected that – to a large extent – FI applications will thus be composed of third-party offerings deployed on federated service delivery platforms through different cloud delivery models, such
as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).
Ultimately, this means that loosely-coupled Internet services will form a comprehensive base for
developing value-added applications in an agile way. This is unlike traditional application development,
which uses computing resources and software components under local administrative control. To main-
tain their quality of service, FI applications therefore need to dynamically and autonomously adapt to
an unprecedented level of changes that may occur during runtime.

Problem Statement: Over the past decade a wealth of technologies for engineering adaptive Web-
based services and service-oriented systems has emerged. Those technologies offer significant advance-
ments for what concerns furnishing applications with self-adaptive capabilities. Still, those solutions
focus on isolated pillars of the Future Internet only. For example, many solutions consider software
services (i.e., IoS) but fall short of integrating things (i.e., IoT) which leads to different levels of hetero-
genicity and unprecedented level of change on available resources and data.

Relevance of Problem and Progress from State of the Art: Different solutions for dynamic adapta-
tion have been developed for various areas of the FI, such as software services (IoS), as well as data and
media (IoC). This work presents those solutions and suggests areas for future research to augment and
integrate them towards solutions for self-adaptive FI applications.

Relation to WP Challenges: The contribution to the WP is related to the research challenges of
“Quality Prediction Techniques to Support Proactive Adaptation” and ‘Run-time Quality Assurance
Techniques”, as the paper discusses challenges and requirements for enabling truly adaptive FI appli-
cations. These include a seamless and consistent way of monitoring, detecting and predicting critical
events.

Solution / Research Method: In this work, we review trends and current solutions for adaptive Web
services and service-oriented applications. Based on real-world use cases from multimedia applications,
as well as transport & logistics, we examine the transition from adaptive Web services to technology sup-
porting the engineering and operation of adaptive FI applications. We demonstrate that FI applications
promise full integration and combination of real, physical world services, business objectives and ICT
services. This means a shift from considering adaptation only in the ICT level (and business level, as
some initiatives already show), i.e., such as service-oriented applications, but extending this concept to
unprecedented levels.

Benefits and Evaluation: This work analyzes and justifies the need for the transition from adaptive
Web services and service-based applications to adaptive FI applications. We examine how current adap-
tive solutions need to be enhanced to properly address the adaptive needs of FI applications. Finally, we
propose future challenges that need to be considered in adaptive FI applications.

The discussions in this work are based on two real-world use cases from the multimedia and transport
& Logistics domains. The first use case explores how adaptation of IoC and IoS should be considered
in FI multimedia applications. The second use case demonstrates the importance of combining IoT, IoS,
and business objectives for the success of FI transport & logistics applications.

Relation to Research Framework: The contribution is relevant to Monitoring and Adaptation (JRA-
1.2) and to the Service Composition and Coordination layer (JRA-2.2) as it discusses how existing solu-
tions for monitoring services and adapting service-based applications need to be extended/integrated to
meet the requirements of FI Apps.
Discussion and Future Work: There are many questions to be answered, and for each question new ones emerge. Despite all the uncertainties surrounding FI applications, there is at least one certain and incontestable fact: FI applications will have to be engineered explicitly considering adaptation aspects. This work discusses the need of thinking and designing FI applications considering aspects beyond the ones considered by current adaptive Web services and service-based applications. To enable adaptive FI applications, it is clear that a seamless and consistent way of monitoring, detecting and predicting critical events, dealing with cross-cutting aspects, decentralization, and boundaries between application logic and adaptation needs is required. Besides other challenges, applications need to be able to dynamically adapt to an unprecedented level of changes that can occur during runtime.

3.2.9 “SLAs for Cross-layer Adaptation and Monitoring of Service-Based Applications: A Case Study” [5]

Context and Background: One of the main barriers to the adoption of service-based applications (SBA) is the concern raised over the trust-worthiness and reliability of third-party services utilised in an SBA. The third-party software services are often implemented as Web services that realise business activities, such as paying with a credit card or shipping purchased goods, and they are beyond the control of the SBA provider. The problem of reliability becomes more complex when third-party cloud computing services are utilised as the underlying infrastructure for provisioning the SBA. Given that the SBA provider does not have control over the quality of the third-party services, unreliable third-party services could threaten the quality of the SBA and result in lower business performance, software faults, and performance degradation that could consequently lead to the total collapse of the SBA. Therefore the dependability of the third-party business, software, and infrastructure services utilised in an SBA becomes a principal concern for the SBA provider, who will require to adopt mechanisms within the SBA for quality assurance during run-time.

The functional layers of an SBA have been introduced in [17] and comprise the business process management (BPM), service composition and coordination (SCC), and the service infrastructure layers (SI). As such, quality assurance approaches need to consider the layered nature of an SBA. Such an approach to the run-time quality assurance of SBAs is the cross-layer or multi-layer adaptation and monitoring (CLAM), which aims at timely detecting problems in the SBA layers and co-ordinating effective corrective actions across the SBA layers, such that problems are compensated for, or even prevented from occurring [33].

Problem Statement: An important aspect of cross-layer adaptation and monitoring is the identification and the definition of the appropriate Service-Level Agreements (SLAs) for the third-party services utilised in the different layers of the SBAs. As such, it is necessary to analyse the SBA in order to identify the business, software, and infrastructure services and their characteristics, such that Service-Level Agreements (SLAs) are established for the third-party services. An important research question, therefore, is which process must be followed to identify the different types of third-party services and their characteristics, in order to define the appropriate SLAs in each layer.

Relevance of Problem and Progress from State of the Art: Recent research into run-time quality assurance has focused on implementing CLAM techniques for SBAs [33], by integrating the existing fragmented work in the field of adaptation and monitoring of service-based systems. Gjørven et al. [10] introduce a middleware for supporting the implementation of cross-layer self-adaptation of SBAs. Kazhamiakin et al. [17] describe a conceptual framework comprising the definition of the SBA layers and a set of requirements needed to be addressed by the mechanisms and techniques for CLAM of SBAs. Popescu et al. [36] present a methodology for cross-layer adaptation using adaptation templates. Latest research has focused on SLAs for CLAM. More particularly, Fugini et al. [4] describe an SLA contract that comprises parameters from user goals, business service and IT infrastructure for CLAM of
SBAs. Schmieders et al. [48] propose the combination of SLA prediction, which uses assumptions about the characteristics of the execution context, and cross-layer adaptation mechanisms for preventing SLA violations.

SLAs for third-party services utilised in each SBA layer are an important element in such approaches, since SLAs specify the expected characteristics of each third-party service, named Service-Level Objectives (SLOs), to be monitored, and which are mapped to adaptation strategies for compensating or even proactively preventing violations of SLOs. This paper supports the research directions towards the runtime quality assurance of SBAs using CLAM techniques, while suggesting that such techniques could greatly benefit from an analysis approach of SBAs for identifying third-party services and their characteristics across the SBA layers for the definition SLAs. To the best of the authors’ knowledge, there exists only one recent work that is related to the definition of SLAs for cross-layer adaptation and monitoring [8]. Although this work describes a methodology for creating SLAs, it focuses on the dependencies between the characteristics of services, and it does not follow the SBA layers as they have been defined in [33, 17]. The authors focus more on how KPIs and IT infrastructure metrics impact the goals of a service user, and they introduce a new indicator named Key Goal Indicator. In contrast, the goal of the presented paper is different since it presents ideas for defining SLAs, by performing analysis of an SBA to identify the third-party services utilised in each SBA layer and their characteristics, in order to define the appropriate SLAs in each layer.

Relation to WP Challenges: This contribution targets the research challenges associated with the quality definition in an SBA. More specifically, it investigates the challenge of End-to-End Quality Reference Model. This contribution exemplifies an approach for establishing SLAs across the layers of an SBA, based on the characteristics of the different types of third-party services consumed in an SBA.

Finally, this contribution is associated to a lesser extend with the challenge of Automatic quality contract establishment. Even if this contribution does not present an automated approach to contract establishment, it exemplifies the required manual steps, which could be potentially automated, for establishing SLAs across the layers of an SBA.

Solution / Research Method: The paper present insights into how to define SLAs for CLAM, by analysing SBAs to identify the third-party business, software and infrastructure services utilised by the SBA. This paper views each layer from the perspective of the type of services utilised in the layer and it suggests that each layer concerns different types of services. The BPM layer concerns business services or business activities realised through software-based services. For instance, a shipping provider exposes a Web services API for shipping goods. The shipment of goods is a business activity provided through a Web service. The SCC layer concerns software services that implement a specific functionality or a business activity. For instance, in the case of the shipping provider, the Web service API is the software-based service. The SI layer concerns the infrastructure services used by an SBA. For instance, an SBA could be running on a third-party cloud computing infrastructure and rely on shared computing, storage, and networking resources. Based on the aforementioned suggestions, the authors argue that an SBA is a software application that outsources business activities, consumes software services, and uses infrastructure services.

Benefits and Evaluation: The analysis process is exemplified through a case study that concerns the definition of SLAs in an existing platform-as-a-service framework, developed during the European project CAST [18]. The analysis reveals the different third-party services and their characteristics, as a precursor to defining SLAs. In the BPM layer, two business services were identified, so two separate SLAs are required between the platform provider and the two service providers. In the SCC layer, two software services were identified, so two separate SLAs are required between the platform provider and the two service providers. In the SI layer, one infrastructure service was identified, so one SLA is required between the platform provider and the service provider.
Each of the services identified in the study was then analysed to reveal its individual characteristics, prior to drawing up appropriate SLAs. The study clearly demonstrates the utility of separating the runtime quality assurance concerns at each layer of the SBA. The case study successfully demonstrates how distinct SLAs for business, software and infrastructure services may be applied respectively in the BPM, SCC and SI layers of an SBA, to provide the basis for building monitoring and adaptation mechanisms, which utilise SLAs across layers.

Relation to Research Framework: With respect to the S-Cube research framework, the perspective presented in the paper is mainly related to the research area of "Quality Definition, Negotiation, and Assurance" across BPM, SCC and SI layers, and is to a lesser extent related to the areas of "Adaptation and Monitoring" and "Engineering and Design".

Discussion and Future Work: Although the approach was exemplified through a case study, its applicability needs to be examined in more scenarios involving diverse SBAs. Additionally, the approach does not consider the existence of potential dependencies across SLAs.

As future work the authors suggest to investigate existing methods for representing SLAs for the business, software, and infrastructure services. Finally, they plan to extend previous work [4] related to the implementation of an extensible monitoring architecture for Web services, in order to support the development of a CLAM framework for SBAs that will utilise multiple SLAs for monitoring of business, software, and infrastructure services.

3.2.10 “Negotiation towards Service Level Agreements: A Life Cycle Based Approach” [13]

Context and Background: Service Level Agreements (SLAs) play a major role in ensuring the quality of Service Based System (SBSs). They stipulate the availability, reliability, and quality levels required for an effective interaction between service providers and consumers. It has been noticed that because of having conflicting priorities and concerns, conflicts arise between service providers and service consumers while negotiating over the functionality of potential services.

Problem Statement: Negotiation is carried out between the service provider and the consumer before any kind of agreements can be established. This negotiation is likely to raise conflicts because of difference in Quality of Service (QoS) priorities. Opposing concerns of stakeholders on the provider as well as on the consumer side, across different phases of the life cycle, may raise conflicts on negotiating QoS capabilities (such as response time). It is really important to mitigate these conflicts so that SBS stakeholders, who contribute towards the business value, can mutually agree upon an SLA. In this research, we propose a stakeholder negotiation strategy for Service Level Agreements, which is based on prioritizing stakeholder concerns based on their frequency at each phase of the SBS development life cycle.

Relevance of Problem and Progress from State of the Art: Conflicts which arise during the SLA negotiation are likely to be overcome either by going for an alternative service provider, or by renegotiation among stakeholders. The former one may not be a good idea as it could involve more overheads in terms of looking for a new provider and finalizing the agreement with it. Recent research in the area has not focused on stakeholders at different phases of the life cycle and their potential role in the negotiation process which eventually leads to SLAs. As they are the most common mechanism used to establish agreements on the quality of service between the service provider and the service consumer. In addition, it is important to take the stakeholders into account considering that SBSs are developed, owned, and used by different stakeholders with different perspectives, i.e. developer and provider, broker and composer, and consumer and end user respectively.
Relation to WP Challenges:  The paper aims to address the core issue of end to end quality provision by means of SLA negotiation, which is one the focus areas of the deliverable JRA-1.3.6.

Solution / Research Method: We identify stakeholders and roles associated with them based on their key responsibilities at each phase of the SBS development life cycle. Then, we identify how conflicts may occur between the service provider and the consumer while negotiating towards SLA. An example scenario is presented using the Collaxa BPEL Loan Flow Service to demonstrate the potential conflicts which can occur between the stakeholders. Our goal has been twofold: propose a life cycle based negotiation methodology which could involve stakeholders, information at each phase of the SBS development life cycle, and validation of the approach in terms of numbers to demonstrate the usefulness of the approach.

Benefits and Evaluation: We have proposed a life cycle based methodology for negotiation between service providers and service consumers. The results suggest that assigning priority based on the proposed approach could reduce cost of SLA negotiation. We used a Loan Flow service example to identify a scenario which may lead to conflicts on QoS between service provider and the service consumer. The identified conflicting node was mapped back to the SBS life cycle to investigate the potential involvement of each life cycle phases. Using this information, we measured the preference of a stakeholders role on the conflicting node by calculating its relative frequency in comparison to the other roles. The greater the relative frequency value is, the more importance that stakeholder category has in the SLA negotiation process. These priority values were simulated to observe the potential impact of the corresponding stakeholders on the cost of the SLA negotiation.

Relation to Research Framework: The work presented in this paper mainly covers the theme Quality Definition, Negotiation, and Assurance which is likely to facilitate the topics of interests in JRA-2 as well; for example, Monitoring and Adaptation, Engineering and Design, Service Composition and Coordination.

Discussion and Future Work: We have proposed a life cycle based methodology for negotiation between service providers and service consumers. It is important to understand and implement a good negotiation process as it leads towards a formal agreement between the two parties in the form of the SLA. The numbers associated with SBS stakeholders across different life cycle phases may vary in different environments but basic theme for assigning priority to any of the stakeholder involved in the negotiation process remains the same. It should be noted that we did not include service consumers as application users as it is rather impossible to predict the exact number of the potential service users. In our future work, we plan to implement the proposed approach with automated contract negotiation. This will allow us to measure cost as well as other quality attributes associated with electronic contract negotiation.


Context and Background: Run-time adaptability of service-based business processes is a key feature of dynamic business environments, where the processes need to be constantly refined and restructured to deal with exceptional situations and changing requirements. The execution of such a system results in a set of adapted process variants instantiated on the same process model but dynamically restructured to handle specific contexts. We propose a framework supporting context-aware evolution of business processes based on process instance execution and adaptation history. Instead of looking for recurring adaptations, we propose to look for recurring adaptation needs (i.e., process instances with the same context constraint violation and system configuration). Based on the analysis of adapted instances, we
automatically construct and rank corrective evolution variants which can handle the problematic context. At the same time, we try to identify preventive evolution variants by constructing process variants which can prevent the raising of the adaptation need. We demonstrate the benefits of our approach using a car logistics scenario.

Problem Statement: The need for continuous adaptation results in a system characterized by a huge set of process executions that, although instantiated on the same process model, strongly differ in terms of process structure. Providing support for process model evolution is becoming one of the main requirements for managing the lifecycle of dynamic processes [51]. In particular, the set of adapted process instances together with the information concerning their execution should be used as training cases for evolution mechanisms in order to progressively improve process models that are then used to instantiate future process instances. Most existing approaches addressing this problem (e.g., [40, 38]) derive model-level changes by analyzing frequently occurring changes at the instance-level. In other words, if an instance-level change/adaptation occurs more frequently than a predefined threshold, the change will be propagated at the model-level. These evolution approaches present two major drawbacks. First, an instance-level adaptation variant is not good in general, as it is good for just a specific context/situation, and thus cannot simply be propagated to the process model without taking into account the adaptation need it was devised for. Second, plugging-in adaptation variants in the original process model is not always a good solution, since it may result in embedding fault-handling activities rather than trying to solve the problem that required runtime adaptation.

Relevance of Problem and Progress from State of the Art: The problem of supporting the evolution of business processes models has been addressed by several works. Some of these approaches are able to capture a precise set of contexts and to use for each of them a predefined process variant [12, 41], others support the evolution of processes by analysing previous executions and adaptations [40, 38, 35, 11, 22]. Most existing approaches focus on the problem of extracting useful information from the adaptation logs. The approaches differ both in what they log and in the techniques that they use to analyze the logs (e.g., [40, 38, 11, 22, 49]). These approaches may be used in the analysis phase of the proposed evolution framework. One direction is to use process mining techniques, as in [38], considering large collections of structurally different process variants created from the same process model. The authors use a heuristic search to find a new process model such that the weighted average distance between the new model and the variants is minimal. The approach in [11] also uses mining techniques to analyze change logs. The evolution result of this approach is an abstract change process consisting of change operations and causal relations between them. These change processes can be used as an analysis tool to understand when and why changes were necessary. In [40], concepts and methods from case-based reasoning (CBR) are used in order to log, together with the change operations, also the reasons for and context of each change. Change information is stored as cases in a case-base specific to the process model. The case-bases are used to support process actors in reusing information about similar ad-hoc changes, and are also continuously monitored to automatically derive suggestions for process model changes. Change analysis and reuse is also relevant for loosely specified process models. [22] facilitates change reuse by providing a search interface for the repository of process variants. For declarative processes, [49] supports users through recommendations, which are generated based on similar past executions and considering certain optimization goals. The approaches in [40, 38] are closest to our work, since they also generate new process models based on the information from the adaptation logs. A limitation of the approach in [38] is that it does not consider the context requiring adaptation. An instance-level adaptation may be useful only for a particular context, and therefore should not be included in the process model even if it occurs relatively often. Although [40] considers also the context of changes, this is specified as natural language question-answer facts. Such facts are useful for supporting process actors in identifying the existing cases with the same context. They are also useful for the process engineer, who can manually determine the context for a new process model change.
However, these tasks cannot be done automatically. Further, in both [40, 38] the decision whether to integrate a change operation in the process model is determined by the frequency of the change operation. However, it can be the case that the need for adaptation was a fault in the process, and the adaptation contains fault handling and compensating activities. In these cases, rather than including the adaptation in the process model, we may be interested in avoiding the adaptation need altogether. In contrast to [40, 38], we use the context as the main driver for evolution. This allows us to determine if an instance-level adaptation is useful for a particular context, and it allows us also to search for an alternative process model which avoids a problematic context. The importance of context for improving process models is recognized also in [35]. Here, the authors propose a context-aware process management cycle, with context-awareness spanning all the stages of the process lifecycle. While [35] remains at a very general and abstract level, we are taking the approach one step further, and provide concrete ideas for implementing the context-aware process management cycle.

Relation to WP Challenges: The framework proposed in the paper contributes to the challenge "Quality Prediction Techniques to Support Proactive Adaptation". This because its main goal is to evolve service-based business process (i.e., long-term adaptation) considering the set of adapted process instances together with the information concerning their success and their execution context. All this collected information are used as training cases for a performance analysis in order to progressively improve the process models with respect to a set of KPIs. The results of this analysis can be used to generate a set of evolution variants which optimize the existing process model with respect to the KPIs. The evolution variants are then proposed to the process designer. In case the process designer decides to evolve the process model, all new process instances will be based on the evolved model.

Solution / Research Method: To overcome the limitations, presented in the problem statement paragraph, we present a context-aware evolution framework that, instead of searching for recurring process changes, searches for recurring adaptation needs. At run-time, we may determine that a certain context which at design-time was assumed to occur rarely, actually occurs for a high percentage of the process’ instances. In this situation, we analyze the instance-level adaptations that have been used for handling the unexpected context. Based on these adaptations, we determine the changes that must be performed on the process model, in order to handle the new context. On the basis of the analysis results, the framework automatically proposes process variants that either embed general corrective solutions derived from context-specific adaptations, or restructure the original model to prevent the recurring adaptation need from occurring. Figure 3.1 shows an architectural overview of the framework presenting the relations between the different components and positioning them with respect to the the three main phases of the evolution lifecycle, namely (1) execution phase, (2) analysis phase, and (3) evolution phase.

During the first phase, the framework is responsible for managing the execution and adaptation of the system and for logging all the information that may be useful to the other phases (e.g., execution traces, adaptation needs, adaptation variants, execution performances). During the analysis phase, the framework controls and evaluates the quality of execution of the processes with respect to the KPIs, decides the need for evolution for a certain process model, and, on the basis of the execution history, identifies the contextual evolution problem in terms of recurring system configuration that required adaptation. Finally, in the evolution phase, the framework uses the information obtained from the analysis phase to compute process model variants that either embed the best performing (with respect to KPIs) adaptation variants or prevent the violation of the context constraint. These evolution variants are then presented to the process designer, who decides whether they should be adopted for future executions. The process designer obtains the evolved process model using a set of supporting tools.

Benefits and Evaluation: We have presented a framework for evolving process models based on a history of process instance executions and adaptations. Our approach is context-driven. If the need to evolve the process model is detected, we analyze the relevant adapted process instances and look for
recurring adaptation needs (i.e., the same constraint violation and system configuration). This allows us to construct and rank evolution variants which can handle the problematic context (corrective evolution). It also allows us to construct evolution variants which can prevent the adaptation need (preventive evolution). We introduced the modeling artifacts for evolvable process-based applications, and a detailed framework for context-aware evolution. Finally, we demonstrate the benefits of our approach using a car logistics scenario.

**Relation to Research Framework:** The proposed approach is related with both WP-JRA 1.1 and WP-JRA 1.2. With respect WP-JRA 1.1, our approach covers both the adaptation and evolution part of the S-Cube life-cycle to manage adaptable SBA. This life-cycle was refined with explicit artifacts needed to specify and manage adaptable and evolvable service-based business processes. Concerning WP-JRA 1.2 we re-use some results already presented in the WP concerning context-aware adaptation of service-based business processes.

**Discussion and Future Work:** We have presented a framework for evolving process models based on a history of process instance executions and adaptations. Our approach is context-driven. If the need to evolve the process model is detected, we analyze the relevant adapted process instances and look for recurring adaptation needs (i.e., the same constraint violation and system configuration). This allows us to construct and rank evolution variants which can handle the problematic context (corrective evolution). It also allows us to construct evolution variants which can prevent the adaptation need (preventive evolution). In our future work, we will develop concrete solutions for corrective and preventive evolution. For corrective evolution we plan to develop techniques to automatically transform instance-level adaptation variants into evolution variants using the built-in adaptation tools. For preventive evolution we will apply AI planning techniques to re-plan the process model in order to avoid the critical configurations. We will implement and evaluate our solutions on realistic scenarios, such as the car logistic scenario introduced in this work.
3.2.12 “Constraint-Based Runtime Prediction of SLA Violations in Service Orchestra-
tions” [16]

Context and Background: Quality of Service (QoS), such as execution time, availability, reliability or cost, is crucial for the usability and economic value of services and service compositions. Service Level Agreements (SLAs) define the QoS levels expected to be met by a service provider to the service clients. SLAs usually specify ranges of values of QoS properties in the form of Service Level Objectives (SLOs), which represent the constraints on the design and behavior of services and service compositions. Predicting an SLO violation ahead can trigger adaptation to avoid this violation by changing the data or behavior of an executing instance.

Problem Statement: The problem addressed in this contribution is efficient and accurate prediction (ahead of time) violations of SLA objectives (SLOs) for a running instance of a service orchestration. Furthermore, we are concerned with fine-grained predictions that can be made at arbitrary points in the execution, and even continually. We also want to minimize the dependence on historic data (that is usually used in the data-mining approaches) and concentrate on the structure of the orchestration and the tasks ahead in its execution.

Relevance of Problem and Progress from State of the Art: The proposed approach makes a detour from the usual statistical prediction models for SLA violation that typically require significant amount of data on both component services in an orchestration, as well as on the behavior of the orchestration itself. Several such approaches, based on decision tree induction or regression models have been proposed [52, 21]. While such approaches in principle abstract from the structure of a service orchestration and avoid dependencies on particular orchestration language and constructs, they have several problems that the present approach aims at solving. First, the data mining techniques used in these approaches use depend on historic probability distributions, which tend to reflect past conditions in the service system and its environment, rather than the current ones. Adding more information to the statistical prediction models is computationally expensive, and therefore updating the model is done only periodically, and usually not on a running instance level. If the definition of the orchestration changes, the collected historic data becomes obsolete, and new critical mass of data has to be collected. Furthermore, if the structure of an executing instance changes, data mining based predictions become unreliable, unless a model that corresponds to the new structure has been precomputed.

The constraint based approach to QoS prediction also relies on historic data that describes the ranges of QoS values for component services, but in a different manner. First, it is not concerned with probability distributions inside of these ranges, or with stochastic (in)dependence between QoS of different services and their joint probability distributions that require significant data sets to be statistically significant, and encode historic trends. Second, updating the bounds from new execution data is quite a simple operation in comparison to (re)building of data mining models, and can be performed to update accuracy of predictions as soon as such data is available.

Instead of trying to historically correlate orchestration QoS and that of its components, the constraint based approach relies on the structure and the current state and control point in execution of a given instance of the orchestration. That information is always implicitly present in the interpreter (i.e., the orchestration execution engine), from which it can be extracted either by design of the engine, or in principle (e.g., by “doctoring” the existing open-source engines). Since the prediction is made at the instance level, and can be performed continually, an instance-level adaptation does not decrease the accuracy of prediction. Furthermore, the prediction is based on efficient and industry proven Constraint Logic Programming [2] tools and techniques that present little overhead on top of the regular execution.

Relation to WP Challenges: This contribution is related to the following research challenge within the workpackage JRA-1.3: 
Quality Prediction Techniques to Support Proactive Adaptation

Solution / Research Method: The solution is based on using a continuation [39] that is emitted by the process execution engine, which describes the steps in the service orchestration that remain to be performed from the moment of observation until the end of its execution (including remaining iterations of the started loops). The continuation is used for constructing a set of equalities and inequalities (i.e., the constraints) over variables that represent QoS metrics [3] (such as running time or availability) of the building blocks in the continuation, starting from elementary state, and upwards. For service invocations, predefined upper and lower bounds for QoS are used, which can be updated by observing the actual invocation and reply events exchanged between the execution engine and the external services. For if-then-else branches, the Boolean value of the condition is explicitly included into the constraints to allow reasoning on what conditions may lead to SLO compliance and violation, respectively. For loops, an integer loop counter is explicitly introduced into the constraints to enable reasoning on what number of loop iterations may lead to SLO compliance and violation, respectively. The loop counters can be further constrained using either known data at run-time to compute their exact value (such as in a foreach over data structures of the known size), or by performing computational cost analysis and expressing the (safe and conservative) upper and lower bounds for loop iterations as functions of data in the orchestration [15, 14, 30].

The set of constraints generated from the given continuation (together with the constrained variables and their domains) constitutes a Constraint Satisfaction Problem (CSP) [7, 1] that is solved, using an interval constraint solver [2], for both the case of SLO compliance and the case of SLO violation. Existence of solution just in one of these two cases signifies prediction of certain SLO compliance or violation, respectively. When solutions for both cases exist, the differences between the resulting intervals for the constrained variables (for instance, branch conditions or loop iteration counters) can be used for detecting events that necessarily lead to the one or the other outcome. For instance, the predictor can say that an SLO may be satisfied if the number of loop iterations is below 12, and that the SLO may be violated if the number of loop iterations is greater than 2. This allows us to infer, e.g., that exiting the loop after less than three iterations leads to the SLO’s satisfaction, and that at entering the twelfth iteration the SLO violation is imminent. The ranges for all QoS metric variables included in the CSP, including all component metrics, are given as numeric intervals.

Benefits and Evaluation: We have evaluated the proposed constraint-based QoS predictor on a sample industrial service composition that involves collection of purchase orders, product planning, assembly, billing and delivery. We have observed 100 instances and used the information on the execution ranges of component services to predict whether the composition’s time limit (the SLA objective under consideration) can be met and under what conditions.

The results have shown that performing around 160 predictions during the lifetime of an average instance takes between 1 and 2% of its execution time, thus proving that this prediction technique does not incur significant computational overheads. The accuracy of prediction, across different time limits, ranges between 94% and 100%, provided that the assumptions on the running times of the component services are correct. The time lead between a SLO failure prediction and the actual violation ranged on average between 15% and 20% of the time limit.

Relation to Research Framework: Relative to the service life-cycle, the proposed constraint-based predictor of SLA violations is applied at service composition run-time. The prediction is applied at the Service Composition Layer, and relates to the End-to-End Quality Assurance as the cross-cutting concern in the IRF.

Discussion and Future Work: The constraint based prediction of SLA violations for service orchestration is a very accurate and efficient QoS prediction technique that can be applied to individual service
instances and can accommodate both static and dynamic instance-level adaptation. It is subject to availability of process continuations, which are provided by or extracted from the process execution engine. Its accuracy is affected by the knowledge of safe QoS bounds for component services, which can start from initial assumptions and be continually updated from the new data.

Our future work will concentrate on applying this approach to the existing process execution engines, and on evaluating the effect of incomplete and imprecise knowledge on the precision and accuracy of prediction.
Chapter 4

Conclusions

Adaptation has been long been proposed as a solution to ensure that service-based applications become resilient against failures and changes of third-party services. For those service-based applications, we see a recent trend to complement solutions for reactive adaptation (i.e., repairing a system in response to failures that have actually occurred) with proactive capabilities (i.e., modifying the system before an imminent failure actually occurs). Proactive adaptation thus means that the service-based application “can try to apply countermeasures in order to prevent the occurrence of a failure, or it can prepare repair mechanisms for the upcoming failure in order to reduce time-to-repair” [44].

This deliverable presented 12 contributions that introduce novel and improved (based on validation results) approaches for proactive negotiation and quality assurance, providing essential capabilities for proactive adaptation. Out of the 12 publications (with overlaps), 3 focus on quality negotiation and 10 focus on run-time quality assurance and quality prediction.

Clearly, the majority of the presented contributions is concerned with principles and techniques for online quality prediction in the context of SBAs. These approaches represent a significant progress towards the main focal point of WP-JRA-1.3, which is on enabling proactive SBAs through exploiting run-time, dynamic quality negotiation and quality prediction techniques.

4.1 Future Research Activities on Online Quality Prediction

Adaptation capabilities will become even more relevant in the extremely dynamic and complex setting envisioned in the Future Internet, where – besides others – services provided by the Internet of Services (IoS) and the Internet of Things (IoT) will converge and will thus jointly form service-based applications.

Those trends mean that opportunities and benefits for proactive adaptation will even increase. However, the Future Internet will amplify existing research issues on online quality prediction, as well as lead to additional research issues. As a final outcome of S-Cube, the network will publish a research roadmap looking in detail at those issues (see http://www.s-cube-network.eu/icse). To give a flavour, some of these issues include:

- Ensuring that online failure prediction is accurate is critical [28]. Otherwise, wrong predictions may lead to the execution of unnecessary adaptations (false positives) or missed adaptation opportunities (false negatives). As an example, unnecessary adaptations may introduce severe problems; e.g., if a working service is replaced by one with bugs. Providing accurate failure predictions becomes extremely challenging in the setting of Future Internet applications, especially if they consist of third-party IoS and IoT services due to the heterogeneity and dynamicity of the entities.

- Traditionally, accuracy of predictions is assessed in a “post-mortem” may such as to select a matching prediction technique for a specific usage setting. However, in the Future Internet those usage settings or contexts will continuously change. This means that even if high accuracy is
achieved in an initial setting, accuracy may quickly decrease over time. We thus need new ways to assess the accuracy online, such as to determine during run-time whether to trust the predictions.

- The challenges towards online quality prediction are further amplified in the presence of noisy and uncertain data. Open issues involve how to reason and predict in the presence of such noisy data and to understand how such uncertainties may impact on the accuracy of predictions.

- The Future Internet will lead to a proliferation of data sources and an increased amount and timeliness of operational data (aka. “Big Data” phenomenon). On the one hand, such data may provide better prediction techniques, as more data is available to reason on future situations. On the other hand, “Big Data” leads to issues such as which of the data to store, what to filter and how to process huge data streams in real-time.

***

Acknowledgments: This deliverable constitutes the final document of WP-JRA-1.3. As workpackage leader, I would like to extend my gratitude to all researchers that jointly produced excellent, novel research outcomes concerning the challenges pursued within the workpackage and with whom I had many fruitful collaborations and inspiring discussions.

—Andreas Metzger
Bibliography


[24] C. M


Usage-based Online Testing for Proactive Adaptation of Service-based Applications

Osama Sammodi∗, Andreas Metzger ∗, Xavier Franch †, Marc Oriol †, Jordi Marco †, Klaus Pohl
Paluno (The Ruhr Institute for Software Technology), University of Duisburg-Essen, Essen, Germany
Email: osama.sammodi, andreas.metzger, klaus.pohl @paluno.uni-due.de
Universitat Politècnica de Catalunya, Barcelona, Spain
Email: franch@essi.upc.edu and moriol, jmarco @lsi.upc.edu

Abstract—Increasingly, service-based applications (SBAs) are composed of third-party services available over the Internet. Even if third-party services have shown to work during design-time, they might fail during the operation of the SBA due to changes in their implementation, provisioning, or the communication infrastructure. As a consequence, SBAs need to dynamically adapt to such failures during run-time to ensure that they maintain their expected functionality and quality. Ideally the need for an adaptation is proactively identified, i.e., failures are predicted before they can lead to consequences such as costly compensation and roll-back activities [4].

A. Problem Statement

Key to proactive adaptation is the ability to predict the future quality of the SBA and its constituent services. Quality prediction was targeted as early as SBAs emerged, cf. [5]–[7]. Typically, monitoring is used to assess the quality of the SBA and its constituent services during their operation. Based on monitoring data, failures are predicted and thus the need for adaptations is identified [8]. However, monitoring only observes services or SBAs during their actual use in the field [8]. Due to its “observational” or “passive” nature, monitoring does not ensure a comprehensive coverage of the test object, i.e., monitoring might not cover all relevant service executions. This can diminish the precision of failure prediction, i.e., the ability to correctly predict deviations in expected functionality or quality [9].

To address the shortcomings imposed by the “passive” nature of monitoring, researchers have suggested performing test activities during the operation of the SBA [2], [4], [10]. Such an online testing means that the constituent services of an SBA are systematically tested in parallel to the normal use and operation of the SBA. In online testing, like in traditional testing, we face the problem of determining when, how and how much to test [10]. However, the existing literature does not provide concrete and systematic techniques for exploiting online testing for quality prediction, and thus for proactive adaptation.

B. Contribution of the Paper

The paper presents a novel online testing technique that provides enhanced proactive adaptation capabilities to SBAs. More specifically, the paper provides the following major contributions: (1) A framework for proactive adaptation, which exploits synergies between monitoring, online testing and quality prediction. The framework’s core element is a test selection activity that utilizes information about the usage of the SBA’s services in order to select test cases that lead to better coverage of service executions, while utilizing a limited number of online test executions (see Section III). Our work presented in [4], [9] can be considered the seed of the framework presented in this paper. (2) A prototypical implementation

I. INTRODUCTION

Service-orientation is increasingly adopted as a paradigm for building highly dynamic, distributed and (self-)adaptive software systems, called service-based (or service-oriented) applications (SBAs). An SBA is realized by composing individual software services. Future SBAs will increasingly be composed from third-party services that are accessible over the Internet [1]. Such SBAs take the concept of ownership to the extreme: not only is the development, quality assurance, and maintenance of the software under the control of third parties, but the software itself is also operated and managed by them [2].

Third-party services can change or evolve in ways not anticipated during design-time [3]. For instance, a service can become unavailable, or can reply too slow due to network latencies or overload at the provider’s side. Therefore, SBAs need to dynamically adapt to such failures during run-time to ensure that they maintain their expected functionality and quality [2]. Ideally, the need for an adaptation is proactively identified, i.e., failures are predicted before they can lead to consequences such as costly compensation and roll-back activities [4].
of that framework based on the existing SALMon monitoring framework [11] with components to collect and report monitoring information, and to execute usage-based online tests (see Section IV). (3) Simulation and experimental assessment of the proposed techniques and framework, assessing and analyzing the improvements of our approach compared to monitoring in isolation, on the precision of failure predictions for the case of performance testing (see Section V).

II. PROGRESS FROM THE STATE OF THE ART

In the literature several techniques and platforms propose to perform online testing periodically or event-driven, e.g., after the reconfiguration of the SBA (see [4], [12]–[14]). However, they do not provide concrete techniques for test case selection. Some proposals for test case selection, like the regression-oriented one by Ruth et al. [15], assume the availability of information like the control flow graphs, which is generally not available in the case of third-party services. There are also traditional partition-based approaches that derive test cases from WSDL specifications [16], [17], finite state machines [18], graph transformation rules [19], or OWL-S [20], [21]. However, in all these approaches, test case generation is insensitive to the actual usage of the services as well as to existing monitoring data, leading to redundant or insufficient coverage.

Usage-based testing is a technique aimed at testing software from the users’ perspective. It drives the allocation of test cases in accordance with use, and ensures that the most-used operations will be the most tested. Typically, either Markov chains or operational profiles are used to represent usage models. Markov chains represent the system states and transitions between those states, together with probabilities for those state transitions [22]. Operational profiles are defined as a set of operations and their probabilities [23]. Several approaches for usage-based testing have been proposed not only in classical software development, but also in the service domain. However, the approaches in the service domain neither target the online test case selection problem [24], nor provide precise guidelines [25], nor focus on runtime [26].

Tow main directions for combining online testing with monitoring can be observed in the literature. Firstly, using monitor data to build the usage profiles that represent usage models. Bai et al. [24] propose an ontology-based approach for capturing service usage profile by intercepting the SOAP messages, and then log the input/output data to be used in test cases. However, the idea was proposed for test case definition and not for the test case selection based on the usage profile. Tsai et al. [26] presented a reliability assessment and prediction model for SOA-based systems based on combining data from operational profile testing with monitoring data during runtime. Operational profile testing was used at design time and not at runtime, which this is different from our focus on online testing. Secondly, exploiting monitoring data to reduce the number of service invocations when executing a test suite (i.e., reducing the cost of testing). Approaches like [3], [27] used this idea to mimic the service responses.

III. PROACTIVE ADAPTATION FRAMEWORK

A. Overall Framework

An overview of our proposed framework for online testing and its key activities is provided in Fig. 1. The framework consists of two main loops: one for testing and another for monitoring. It should be noted that our solution is targeted towards service integrators that offer the composed service (i.e., SBA) to external service consumers. Thus, during the operation of an SBA, several users could employ it simultaneously (i.e., multiple SBA instances can run in parallel).

The framework prescribes the following activities:

1) Test initiation: This activity constitutes all preparatory activities to enable online test selection and execution during run-time, such as the definition of potential online test cases. We suggest exploiting existing techniques for test case generation from service descriptions such as WSDL (cf. [8] for a survey). Additionally, test cases from design phase can be re-used if sufficient [3].

2) Test case selection: This is the central activity of our framework. Section III-B provides further details about how the usage-based test case selection approach has been realized.

3) Test execution: The responsibility of this activity is to execute the test cases that have been selected by the previous activity. This means that services are fed with concrete inputs (as defined in the test cases) and the responses are observed. It is important to note that invoking services can lead to certain “side effects” [28]. As an example, when invoking the service of an online book seller for testing purposes, one would not like to have the “ordered” books actually delivered. In this paper we consider stateless services and assume that testing has no functional side effects [29].

4) Aggregation of monitoring data: During the operation of the SBA, monitoring data of the services is collected. Monitoring data is used for both updating the usage model as the SBA operates (usage frequencies) and also, along with the data resulting from test cases execution, for making predictions about the quality of the services.

![Fig. 1. Overall Framework for Proactive Adaptation of SBAs Based on Usage-based Online Testing](image-url)
5) Usage-model building/updating: The usage model is a key element in our framework as it is used in the test case selection activity. For adaptive SBAs, the model needs to be updated as the SBAs operate in order to reflect changes in usages and the adaptation of the SBAs along their life-cycle. In our approach, we suggest building the initial usage model using results from requirements engineering [30]. During operation, usage frequencies computed from monitoring events are used to automatically update the usage model.

6) Prediction: This activity augments the data points collected by testing with the data points collected during monitoring and performs the actual failure prediction for the services in the SBA. Several prediction techniques exist in the literature (e.g., ARIMA [7]). Based on the prediction results, adaptation requests are issued if the expected quality will be violated.

7) Adaptation: A wide range of adaptation strategies and mechanisms for SBAs exists in the literature (cf. [8] for a survey). In our approach, we focus on the adaptation by dynamic service binding. This means that the services are selected and dynamically substituted at runtime.

B. Usage-based Online Test Case Selection

For test case selection, we took the following design choices:

We divide the execution time of the SBA into discrete “periods” (see Fig. 2). This allows us to determine usage models per period (aggregating monitoring data over a predefined duration) and thus to have a “discrete” control over test case selection and execution. For example, the usage model computed from is used for test case selection in (see Fig. 2).

Each period is divided into equidistant points in time. At each point in time, a monitoring event can occur (if an event occurs between points we consider the following point in time for it), and/or a test can be executed, i.e., (see Fig. 2).

We then compute the testing frequency as follows: Using , at each time point our technique then decides based on a random variable whether to invoke a test case for service or not.

IV. Prototypical Implementation

The architecture of our framework for proactive adaptation (see Section III) is composed of three main subsystems: The decisional, the execution and the adaptation subsystem (see Fig. 3). The decisional subsystem is composed of three components: the Test Case Selector (TCS), the Usage Profile Calculator (UPC) and the Predictor component. Each of these components is in charge of one activity in our proactive adaptation framework (see Fig. 1). TCS selects the test cases from the Test Case Repository (activity 2: “Test Case Selection”).

UPC is responsible for updating the usage model to TCS (activity 5: “Usage Model Building/Updating”) and finally the Predictor component performs the failure predictions (activity 6: “Prediction”).

The execution subsystem is based the SALMon framework [11] and provides monitoring and testing facilities. The main components of this subsystem are the Test Executor (TE) and the Monitor. TE performs the execution of the tests (activity 3: “Test Execution”) whereas the Monitor is in charge...
of the monitoring (activity 4: “Aggregation of Monitoring Data”).

The adaptation subsystem is composed of one particular component, the Adaptor, which is in charge of activity 7 (“Adaptation”).

Currently, we have a first prototype of the execution subsystem, a general-purpose testing and monitoring infrastructure able to record in an internal database the QoS of services with respect to different quality attributes (e.g., response time, execution time and availability). To make it usable in the context of our proactive adaptation framework, it has been extended with the concepts of “invocation of the service” and “input of the requests”, so that it is possible to know which are the QoS results for a particular invocation with a particular input. The Monitor has been developed as a service in order to offer the QoS results to the interested users and software components. Thus, the Predictor and UPC components may query at the appropriate moments the Monitor to obtain the information needed to undertake their tasks, whilst the TCS component may order the execution of particular test cases to the TE.

V. EVALUATION

We conducted an experiment to evaluate our approach. It was designed with the aim to provide evidence in support of the following hypothesis: The complementary use of online testing & monitoring (our approach) improves the precision of failure prediction (i.e., the ability to correctly predict violations of the expected service quality) when compared to monitoring in isolation.

A. Measurement

To quantify the precision of a prediction approach, we take into account the number of “false positive” ( ), “true positive” ( ), “false negative” ( ), and “true negative” ( ) adaptation triggers it produces:

In the “false positive” case, failure prediction predicts a deviation from the expected quality although the service turns out to work as expected when invoked during the actual execution of the SBA. In the opposite case, i.e., when it turned out that the adaptation was indeed needed, it is counted as a “true positive”. Likewise, the “false negative” case refers to the situation in which failure prediction was not able to predict a deviation although the service turns out to fail during the execution of the SBA. In the “false positive” case, it turned out that the adaptation was in fact not required.

Considering the above four cases, this leads us to the following overall computation of precision as the proportion of true negatives and true positives over the total number of positives and negatives (normalized in the interval [0, 100]):

B. Experimental Setup

The experiment is based on: (1) the simulation of an example SBA and its associated services, together with (2) the prototypical implementation of the online test case selection, execution and prediction components (see Section III).

Specifically, we simulate the execution of the workflow of an example SBA (100 instances) and retrieve the response times for its constituent services (i.e., for the respective point in “simulation” time) from a large set of QoS data that has been collected by Cavallo et al. [7]. The QoS data was collected by invoking real services every one hour for about four months, providing up to 2000 data points (simulation period) after removing significant outliers that have values above 20s. This data set serves two purposes: (1) it is exploited as monitoring data to predict the quality of future service invocations; (2) it is used to determine whether at the current point in time a failure occurs and thus is used to compute the precision as explained in Section V-A. In our experiment we focus on response time as the QoS property to predict. Thus, we define one performance test case for each of the services of the SBA, assuming a representative workload for each service. The simulation of the example SBA and its services using the QoS data allows comparing the precision of the two approaches (the complementary use of testing and monitoring vs. monitoring) for same “input” data, thus ensuring an objective comparison of the two approaches. Fig. 4 depicts the workflow of the example SBA as a Markov chain. We exploit this Markov chain to simulate the SBA instances. At each decision in the workflow, we use a random variable to decide on the branches to take. For the example SBA, this means that we will take the branch from “s1” that leads to “s3” with a probability of 20%. Although the example SBA is constructed arbitrarily (to cover some typical transition probabilities), it involves the invocation of real services as shown on the bottom of Fig. 4.

As we know the number of SBA instances that will be simulated to be 100, we can compute the usage probabilities for the individual services as shown in Table I. In our experiment,
those usage probabilities serve as an optimal estimate of a usage model.

<table>
<thead>
<tr>
<th>Service</th>
<th>Transition Probability</th>
<th>Usage Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>100%</td>
<td>5.0%</td>
</tr>
<tr>
<td>s2</td>
<td>80%</td>
<td>4.0%</td>
</tr>
<tr>
<td>s3</td>
<td>20%</td>
<td>1.0%</td>
</tr>
<tr>
<td>s4</td>
<td>75%</td>
<td>3.0%</td>
</tr>
<tr>
<td>s5</td>
<td>25%</td>
<td>1.0%</td>
</tr>
<tr>
<td>s6</td>
<td>100%</td>
<td>1.0%</td>
</tr>
<tr>
<td>s7</td>
<td>100%</td>
<td>5.0%</td>
</tr>
</tbody>
</table>

C. Execution

During the execution of our experiment and the assessment of its results, we have focused on two services, namely “s1” and “s3” (see Fig. 4). These two services represent two extreme cases that can occur in the SBA execution. For “s1” we have the highest usage frequency (see Table I) and thus can expect the highest number of monitoring data. For “s3” we have the lowest usage frequency, and thus can only expect a small number of data points from monitoring.

Further, to perform the experiments, we selected concrete values for the following independent variables and parameters:

- **Target number of tests:** To understand what a realistic amount of test data could be assumed that the maximum number of tests allowed (or feasible economically) is 3 tests within 10 hours, i.e., \( \frac{3 \times 24}{10} = 7.2 \) tests. Per 83.3 days (2000) tests.

  Based on the usage model (Table I), we can expect the following number of monitoring data. For “s1”: \( \frac{500}{s1} = 100 \) tests. For “s3”: \( \frac{500}{s3} = 25 \) tests. Based on the monitoring data to be expected, this means that we run 500 test cases for “s1” and 580 test cases for “s3” during period.

- **Prediction technique:** In this experiment, we use average prediction considering different numbers of past observations. During initial exploratory experiments, we observed that the number of past data points considered for prediction has a significant impact. We thus decided to run the experiments for varying numbers, namely the last 1, 5, and 10 data points.

| Repetition of individual runs: To even out random effects (we use random generators for simulating the SBA instances and test case selection and execution), we repeated the run of the experiment 100 times for each parameter value and averaged the precision gained in the respective runs for the two different approaches. |

D. Results and Discussions

Tables II and III show the results of our experiment. We use to refer to the precision of the prediction of our approach, and to refer to the precision of the prediction using monitoring only. The results for both “s1” and “s3” show that in general both approaches perform better when the number of last observations is smaller. Furthermore, the results clearly indicate that the precision of failure prediction using the complementary use of testing and monitoring (as advocated by our framework) is better than the prediction using monitoring data only. Our approach is more beneficial when the number of monitoring data is small. For example, in the case of “s1”, where the number of monitoring data is high, the improvement is only 4%. Whereas, in the case of “s3”, where the number of monitoring data points is small, the improvement is higher (22%). Although, the maximum improvement is not high, the initial results are still promising.

Considering the costs associated with our approach, the usage model can be used to estimate the number of monitoring data that will be available to decide ahead whether or not to perform online testing on specific services of an SBA. Still, the decision needs to be supported by more appropriate costs models which we will consider in future work.

VI. CONCLUSION AND PERSPECTIVES

In this paper we addressed quality prediction for proactive adaptation of SBAs. To this end, we presented a framework for proactive adaptation of SBAs based on usage-based online results.
testing. The framework exploits synergies between monitoring, online testing and quality prediction to assure better coverage of service executions, and thus, enables more precise prediction of adaptation triggers. Initial experimental results show that the complementary use of testing and monitoring as advocated by our framework, improves the precision of failure prediction (i.e., the ability to correctly predict violations of expected service quality) when compared to using monitoring in isolation.

As for future work, we plan to use more advanced prediction techniques in order to enhance the prediction and to further reduce the number of tests that are needed. We will also consider metrics and cost models that would, for example, relate the cost of testing to the cost of compensation activities of wrong adaptations. Furthermore, we plan to run live experiments with executions of SBAs and test cases in order to more realistically assess our approach.

Acknowledgments: The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube), and was partially supported from the Spanish project TIN2010-19130-C02-01. Marc Oriol has an FPI grant bound to the project TIN2007-64753.

REFERENCES


Towards Accurate Failure Prediction for the Proactive Adaptation of Service-oriented Systems
(Invited Paper)

Andreas Metzger
Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen
Gerlingstr. 16, 45127 Essen, Germany
andreas.metzger@paluno.uni-due.de

ABSTRACT
Furnishing service-oriented systems with self-adaptation capabilities allows those systems to become resilient against failures of their constituent services. Especially proactive adaptation capabilities, which strive to prevent the impacts of pending failures, provide significant benefits, such as avoiding costly compensation and repair activities. An important challenge is to trigger proactive adaptations accurately; firstly, because executing unnecessary proactive adaptations can lead to additional costs or failures that would not have arisen in the non-adapted systems; secondly, because missed proactive adaptation opportunities diminish the benefits of such adaptations. This paper discusses two directions along which accurate proactive adaptations can be achieved: (i) by improving the failure prediction techniques that trigger the adaptations (i.e., during design time); (ii) by dynamically estimating the accuracy of the predicted failures during the operation of the service-oriented system (i.e., during run-time). The discussion is backed by concrete examples of existing prediction techniques for service-oriented systems and supported by experimental results.

1. MOTIVATION
Service-orientation is increasingly adopted as a paradigm for building highly dynamic, distributed service-oriented systems. A service-oriented system is realized by composing individual software services. In contrast to a software component, not only the development, quality assurance, and maintenance of the software can be under the control of third-parties, but the software can also be executed and managed by third-parties [16].

There is a clear trend that in the emerging “Future Internet”, service-oriented applications will be increasingly composed of third-party services accessible over the network [21]. As a consequence, the capabilities and quality of service-oriented systems more and more will depend on the quality of its third-party services.

Specifically, this means that service-oriented systems will need to become resilient against failures of their third-party services. As a simple example, a service might become unavailable due to overloads on the service provider’s side. Furnishing service-oriented systems with self-adaptation capabilities is considered a key solution to address this challenge.

1.1 Reactive vs. Proactive Adaptation
There exist three important classes of adaptation [13]: Reactive adaptation, in which case the system is modified in response failures that have actually occurred, preventive adaptation, in which case an actual local failure is repaired before its consequences become visible to the end-user, and proactive adaptation, in which case the system is modified before a pending failure actually occurs and thus impacts on the system.

In a nutshell, proactive or preventive adaptation means that the service-oriented system “can try to apply countermeasures in order to prevent the occurrence of a failure, or it can prepare repair mechanisms for the upcoming failure in order to reduce time-to-repair” [17].

Proactive or preventive adaptation of service-oriented systems thus promises to mitigate some of the key problems faced when resorting to reactive adaptation only, such as the need for compensation and costly repair activities (cf. [6, 14, 19]).

Please note that for matters of brevity, “proactive adaptation” will subsume “preventive adaptation” in the remainder of the paper.
1.2 Problem Statement
To trigger the proactive adaptation of a service-oriented system, pending failures need to be predicted. It is important that such a failure prediction is accurate, such as to avoid the execution of unnecessary proactive adaptations, as well as not to miss proactive adaptation opportunities.

Unnecessary adaptations can have the following severe shortcomings: Firstly, unnecessary adaptations can be costly. For instance, additional activities such as Service Level Agreement (SLA) negotiation for the alternative services might have to be performed, or the adaptation can lead to a more costly operation of the service-oriented system, e.g., if a seemingly unreliable but cheap service is replaced by a more costly one. Secondly, unnecessary adaptations could be faulty (e.g., if the new service has bugs), leading to severe problems as a consequence. Thirdly, as executing the adaptation takes time, this means that in the worst case, an unnecessary adaptation will leave less time to address actual failures.

In case an adaptation opportunity is missed due to inaccurate failure predictions this obviously can lead to the same shortcomings as faced in the setting of reactive adaptations, i.e., it can require compensation or costly repair activities. This means, inaccurate predictions would diminish the benefits of proactive adaptation.

Providing accurate failure predictions is extremely challenging in the setting of service-oriented systems if third-party services are present. The observed quality and functionality of those third-party services can significantly vary between different service invocations. For instance, the performance of a third-party service might depend on the load of the infrastructure at the provider’s side or the network latency if services are offered over the Internet. As an example, a failure observed at one point in time (e.g., unavailability of a service because of an overload at the service provider side) can disappear at a later point in time (e.g., the same service is now executed because of a lower load at the service provider side).

1.3 Contribution of the Paper
This paper introduces two directions along which accurate failure predictions for proactive adaptation could be established; firstly, by improving the prediction techniques themselves; secondly, by dynamically estimating the accuracy during run-time. Based on selected prediction techniques from the literature (see Section 2) and metrics to assess the accuracy of predictions (see Section 3), those two directions for achieving accuracy are critically discussed with support of experimental data (see Section 4).

2. RELATED WORK
Failure prediction – also referred to as quality prediction – has received considerable attention in various areas of computer science and software engineering. A recent survey by Sallner et al. provides an excellent overview of the related work in the more traditional area of computer-based systems [17].

Compared with the increasing complexity, dynamics, and flexibility in those more traditional areas [17], service-oriented applications face unprecedented levels of dynamism, together with a lack of control over third-party services (see above). Thus, different classes of novel techniques as well as adaptations of existing techniques have emerged for predicting the quality of services and service-oriented systems. Those include (as summarized in [18]):

- **Data Mining** [12, 10, 5, 2]: Data mining refers to extracting or “mining” knowledge from large amounts of data. Data mining from monitoring data of previous executions and/or the previous history of the current execution (e.g., collected in form of monitoring logs) is used to predict future behavior and quality. Proposed solutions include variants of statistical methods (such as regression).

- **Online Testing** [4, 19, 14]: Online testing means that the service-based application is tested (i.e., fed with dedicated test input) in parallel to its normal use and operation. Online testing of constituent services of a service-based application is proposed to actively collect additional data points to the ones available from monitoring.

- **Run-time Verification** [20, 15, 22]: Run-time verification is a formal analysis technique used to ascertain whether some predefined properties are met at run-time. The proposed solutions include the use of run-time model checking, which checks whether it will effectively be possible for the execution of the service-based application to finish successfully, and soft constraints, which are used to model SLAs and to relax those constraints to predict the quality and to decide how to rebuild a composition even in the case that not all the requirements can be satisfied.

- **Static Analysis** [7]: Static analysis systematically examines an artifact to infer certain properties. Those properties can include approximations of the future of the computation; e.g., it can be used to predict ranges (i.e., upper and lower bounds) for the behavior of the remaining of the computation.

- **Simulation** [8, 9]: Dynamic models are executed to simulate the behavior of service-based applications and thus to predict their future behavior and thus quality properties. For example, orchestrations are transformed into a dynamic model which also models the resources available to execute the orchestration (e.g., number of simultaneous threads). Alternatively, Discrete-Event-Simulation is used, which allows the prediction of the performance of service-based applications in different settings for load conditions.

Each of these quality prediction approaches works differently, in different settings and with different assumptions; e.g., machine learning works well once a significant amount of training data has been collected. Still, all of those techniques share the same concern: they need to accurately predict the future quality, resp. future failures, of a service-oriented system.

In this paper, we scrutinize two of those approaches: (1) a simple “data mining” approach (using averages as prediction models); (2) an online testing technique that complements current monitoring techniques. The aim is to better understand the potential improvements as well as the challenges involved in establishing accurate quality predictions for service-oriented systems.
3. METRICS
To measure how well a prediction technique works, different metrics have been recommended in the literature. In the following we discuss one major classes of such metrics. So called contingency table metrics allow assessing the “accuracy” of a prediction considering different cases for what concerns the success of a prediction. A subset of those metrics has been identified as useful to assess quality prediction techniques in the service-oriented setting [2].

As described by Salfner et al. [17], contingency table metrics, rely on the following four cases:

- **False positive (FP):** A failure was predicted, but no actual failure occurred.
- **True positive (TP):** A failure was predicted, and an actual failure occurred.
- **False negative (FN):** A non-failure was predicted, but an actual failure occurred.
- **True negative (TN):** A non-failure was predicted, and no actual failure occurred.

Different metrics based on those cases are defined, including the following (which we will use in Section 4).

- **Precision (p):** The ratio of correctly predicted failures to all predicted failures: \( p = \frac{TP}{TP + FP} \).
- **Recall (r):** The ratio of correctly predicted failures to all true failures: \( r = \frac{TP}{TP + FN} \).

As the above metrics do not take into account true negative predictions, i.e., correctly predicted non-failures, the following two metrics provide evidence about a prediction techniques capabilities in this regard.

- **Negative predictive value (v):** The ratio of correctly predicted non-failures to all predicted non-failures: \( v = \frac{TN}{TN + FN} \).
- **False positive rate (f):** The ratio of incorrectly predicted failures to the number of all non-failures: \( f = \frac{FP}{FP + TN} \). Please note that a smaller false positive rate is preferable, provided that the above rates, specifically \( p \) and \( r \), do not decrease.

A final metric, which takes into account both failures and non-failures is:

- **Accuracy (a):** The ratio of all correct predictions to the number of all predictions that have been performed: \( a = \frac{TP + TN}{TP + TN + FP + FN} \).

However, Salfner et al. recommended not to use accuracy as the sole indicator for how well a prediction technique works, as “due to the fact that failures usually are rare events [...] a strategy that always classifies the system to be non-faulty can achieve excellent accuracy since it is right in most of the cases, although it does not catch any failure (recall is zero)” [17].

As a note on terminology, we will use accuracy in the remainder of the paper as a generic term (as already used in the beginning) and will explicitly mention if we refer to accuracy with the more concise meaning defined by the above metric \( a \).

4. ACCURATE FAILURE PREDICTION
This sections employs experimental data collected for assessing quality prediction techniques (Section 4.1) to discuss (i) how the accuracy of failure prediction techniques can be improved (see Section 4.2), (ii) how the accuracy can be dynamically estimated during run-time (see Section 4.3).

4.1 Experimental Setup
This section introduces the general setup of the experiments that we performed to assess the accuracy of different quality prediction techniques [19, 20]. In our experiment we focus on response time as the QoS property to predict.

The experiments are based on: (1) the simulation of an example service-oriented system and its associated third-party services, together with (2) a prototypical implementation of the selected quality prediction techniques (see Section 4.2).

Fig. 1 depicts the workflow of the example service-oriented system that we used in our simulations. It is given in a Markov-like representation, i.e., in addition to the service invocations and control constructs, we have added transition probabilities for our experiment. Based on those transition probabilities, the simulation determines the actual execution paths of a service-oriented system. At each control construct in the workflow, a random variable is used to chose one of the possible branches. As an example, the branch from S1 to S3 will be chosen with a probability of 26%. Although the workflow is artificial (constructed to cover some typical transition probabilities), it involves data from real services as listed on the bottom of Fig. 1.

To simulate the execution of a service, we retrieve its response time – for the respective point in (simulation) time – from an existing data set. The data set we employed has been collected by Cavallo et al. [2]. The QoS data was collected by invoking real services every one hour for about four months, providing up to 2000 data points for our simulation.

4.2 Direction 1: Improved Prediction Techniques
As introduced above, a first direction to improve the accuracy of failure prediction is to employ techniques that promise a higher accuracy than other techniques.

To this end, we analyze two potential approaches; firstly, using different prediction models (see Section 4.2.1); sec-
ondly, using advanced prediction techniques that exploit additional data points (see Section 4.2.2).

Using the experimental setup described above, we have experimentally assessed the different prediction models and prediction techniques by performing 100 runs for each of the prediction models and techniques. During each of those runs, the execution of 100 service-oriented systems has been simulated.

For the purpose of this paper, we will focus our analysis on two services, namely S1 and S3 (see Fig. 1). These two services represent two extreme cases that can occur in the execution of our example service-oriented system. Based on the transition probability, S1 will be invoked with the highest frequency, whereas S3 will be invoked with the lowest frequency.

4.2.1 Monitoring

To demonstrate the effect that choosing a prediction model has on the accuracy of the prediction, in this section we summarize the results of using the arithmetic average as a very simple prediction model. During initial exploratory experiments and as indicated in [2], we observed that the number of past data points considered for prediction can have a significant impact on accuracy. We thus have executed our experiments for varying numbers of past observations, namely 1 (aka. “point prediction”), 5, and 10 data points [19].

Fig. 2 presents the results of our experiments as box-plots for service S3. We used the five contingency table metrics as introduced in Section 3, viz. precision (p), recall (r), negative predictive value (v), false positive rate (f), and accuracy (a).

![Figure 2: Prediction for service S3: Based on last 1, 5 and 10 monitoring data points; using contingency table metrics (Section 3)](image)

It can be observed that although accuracy and false positives rates\(^1\) are better in the case of point prediction (with statistically significance), recall is significantly worse; with almost no difference in precision.

Similar observations can be made for S1, as shown in Fig. 3. Generally, a higher recall appears to be correlated with a higher false positive rate. Although, in the case of S1, the prediction model using 5 data points provides slightly better recall.

As an initial conclusion, it appears that – at least simple prediction models – do not allow determining the “best” prediction technique; not even in a controlled and simple setting such as in our experiment. Of course, the above experiments should be performed with more advanced prediction models (such as time-series models). This is part of our ongoing work.

\(^1\)Please note that, as mentioned in Section 3, a lower false positive rate (f) is preferable

4.2.2 Online Testing

Understanding that, in our experimental setting, the different prediction models turned out to provide the highest accuracy in different situations respectively, this section investigates whether the prediction in general can be improved by exploiting additional information or data for the prediction. In the “machine learning” approach used by Leitner et al. (briefly discussed in Section 2), for instance, estimators for different attributes of the running workflow are employed to improve accuracy. The approach that we will scrutinize in this paper is based on “online testing”, and aims at producing more frequent data points such as to ensure more timely data for the predictions.

Online testing is based on the observation that due to its “observational” or “passive” nature, monitoring does not ensure a comprehensive coverage of the “test object”, i.e., monitoring might not cover all relevant service executions. To address the shortcomings imposed by the “passive” nature of monitoring, researchers have suggested performing test activities during the operation of the service-oriented system [16, 1, 6]. Such an online testing means that the constituent services are systematically tested in parallel to the normal use and operation of the service-oriented system.

In [19], we have introduced a novel approach for online testing, which basically performs an “inverse” usage-based test of the services. This means that if a service has seldom been used in a given time period, and thus not enough monitoring data has been collected, dedicated tests are performed during run-time in order to collect additional evidence of the quality of the service.

Similarly to above, Fig. 4 presents box-plots comparing the accuracy of predictions based on monitoring with predictions that exploit online testing in addition. As a prediction model, point prediction has been employed.

![Figure 3: Prediction for service S1: Based on last 1, 5 and 10 monitoring data points; using contingency table metrics (Section 3)](image)

As visible from the diagrams (and backed by a more thorough analysis for statistical significance), the online testing
approach promises improvements for what concerns the accuracy of quality prediction. Especially, it can be observed that – in contrast to Section 4.2.1 – the prediction is improved for what concerns all five contingency table metrics used.

To demonstrate the potential improvements, we have chosen service S3 (one with few monitoring data), where the impact of online testing is higher than for S1 (one with a high number of monitoring data). In the setting of S1, the use of online testing only provided marginal improvements (see Fig. 5).

![Figure 5: Prediction for service S1: Online testing (ot) vs. monitoring (mon) using contingency table metrics (Section 3)](image)

### 4.3 Direction 2: Dynamic Assessment of Accuracy

As we have seen in the previous section, the accuracy of prediction can depend on many factors, such as the chosen prediction model (last 1 vs. last 5), the actual usage setting in which the prediction is employed (few vs. many monitoring data), etc.

Also, even if we can achieve good accuracy in some of the settings, such as for point prediction using online testing of S3 (see Fig. 4), the variance of accuracy can strongly vary. As an example the variance of accuracy in that case is 0.0172, compared with a variance of accuracy of 0.0015 in the case of S1 (see observations Fig. 5).

If we take those observations into account, one additional direction to achieve accuracy could be to assess the accuracy of the predictions during run-time. Thereby, it would become possible for the system to decide whether to trust the predictions or even dynamically use other prediction techniques that offer a better accuracy at that point in time.

The literature provides first indications on how such a run-time assessment of accuracy could be performed. In the setting of service-oriented systems, Leitner et al. [11] propose to compare actual predictions with the outcome of instances during runtime and to use the prediction error or mean prediction error (i.e., difference between predicted value and actual value). Further, they propose taking into account the prediction error standard deviation (in order to understand whether the the error relatively constant, or whether there are many outliers). However, the error is only used to trigger the re-training of the prediction model, once a predefined threshold has been reached. Dinda [3] proposes computing confidence intervals together with the point predictions to take an informed decisions about the deployment of computational tasks. In a similar fashion, we sketched a potential approach for service-oriented systems [14].

All those proposals are based on predicting a QoS value (and possible errors derived from those values). Thus, those approaches could provide a baseline for new techniques to dynamically compute the accuracy during run-time. However, one issues would need to be critically reflected upon, which is based on an observation made by Cavallo et al. In their paper [2] they observed that prediction error metrics might not provide enough evidence to assess the accuracy of the prediction, as the different cases that can occur (and are reflected in contingency table metrics) are not considered explicitly. Thus, even if the prediction error could dynamically be computed, it might not be what is needed to drive adaptation decisions.

### 5. CONCLUSIONS AND PERSPECTIVES

Failure prediction (or quality prediction) techniques are key to engineer service-oriented systems with proactive adaptation capabilities. However, as discussed in this paper, those predictions have to be accurate, as – for instance – false predictions can lead to additional operational costs and severe failures.

Research in the field has produced a diverse range of prediction techniques that are applicable in different settings and have various benefits and shortcomings. Of course, one could strive to design techniques that provide high accuracy for a known setting. However, if we consider the highly dynamic nature of service-oriented systems in the “Future Internet”, even a failure prediction technique which provided good accuracy for known settings can quickly become “ob-solene”, as it will be highly probably that new, unforeseen settings will dynamically arise.

Ultimately, this suggests to further investigate into quality prediction techniques that also provide capabilities to dynamically assess their accuracy during runtime.

### Acknowledgments

I cordially thank my team members Osama Sammodi and Eric Schmieders for implementing the simulations and prototypes based on which many of the experiments presented in this paper have been performed. Further, I would like to thank Clarissa Cassales Marquezan for her helpful comments on earlier drafts of the paper. Finally, my gratitude goes to the members of the S-Cube Quality Prediction working group for stimulating discussions around the topic of the paper.

Research leading to these results has received funding from the European Community’s 7th Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

### 6. REFERENCES


http://www.s-cube-network.eu/qp/


Accurate Service Failure Prediction through Online Testing

Osama Sammodi∗, Andreas Metzger∗, Marc Oriol†, Jordi Marco†, Xavier Franch†, Klaus Pohl∗
∗Paluno (The Ruhr Institute for Software Technology), University of Duisburg-Essen, Essen, Germany
Email: {osama.sammodi, andreas.metzger, klaus.pohl}@paluno.uni-due.de
†Universitat Politècnica de Catalunya, Barcelona, Spain
Email: franch@essi.upc.edu and {moriol, jmarco}@lst.upc.edu

Abstract—Service consumers have limited control over web-based services offered by third-party parties. Those services thus may behave in ways not anticipated during design time, resulting in failures during run-time. Online failure prediction allows anticipating imminent failures and to proactively initiate repair and compensation. A broad range of online failure prediction techniques has been presented in the literature – all relying on monitoring. As monitoring only passively observes services during their actual use, the amount and timeliness of data may be limited, undermining the accuracy of predictions.

The accuracy of online failure predictions of web-based services can be increased when using online testing (i.e., actively testing service in parallel to normal use). However, online testing may imply additional cost for systems operation. We perform extensive experiments – using QoS data from real web-based services – to measure the dependency of accuracy gains on factors such as usage frequencies and online test rates. Based on our experimental findings, we introduce an extension of an existing Web services monitoring framework that has the capacity to trigger online tests only if the cost of online testing may pay off with respect to accuracy gains.

Keywords—Online failure prediction, testing, monitoring, web services, performance.

I. INTRODUCTION

Web-based services provide unprecedented opportunities to build highly flexible systems by integrating service offerings from third parties [1]. However, service consumers and integrators have limited control over third-party web-based services. Those services thus may behave in ways not anticipated during design time, leading to failures during run-time [2], [3]. For example, those services may exhibit a degradation of quality of service (QoS), such as reduced performance or low reliability.

Online failure prediction of third-party services allows anticipating degradations in expected QoS (e.g., as stipulated in SLAs). Online failure prediction thus allows, for instance, to plan and implement proactive repair or compensation activities [4]. Several online failure prediction techniques have been presented in literature (e.g., [5]–[7]). Those techniques all rely on monitoring of QoS data to predict failures.

Monitoring only passively observes services during their actual use [8]. The amount and timeliness of QoS data collected by monitoring thus may be limited; e.g., when a service is only seldom invoked by users. Sparse monitoring data may undermine the accuracy of failure predictions [9]. On the one hand, this may lead to false positive predictions, which imply unnecessarily switching to an expensive service, delays due to unnecessary repair activities, or replacing a working service with one that has severe defects. On the other hand, this may lead to false negative predictions, which mean that the opportunity for proactive repair and compensation may be missed altogether.

A. Problem Statement

Online testing has been proposed as an active quality assurance technique to complement passive monitoring [9], [10]. Online testing means that constituent services of a service-oriented system are systematically tested in parallel to its normal use. QoS data of tested services can thus be used to augment QoS data from monitoring. Our previous work indicates that such augmented data improves accuracy of failure predictions of web-based services [11].

However, online testing implies additional costs for operating software systems; e.g., when online testing a pay-per-use service [2]. To become applicable in practice, one has to understand when online testing pays off with respect to improvements in accuracy gains.

Different factors can impact on accuracy gains achieved by online testing. Fig. 1 shows empirical results and it compares the prediction accuracy for the response time of an individual service using monitoring data only with that of using complementary QoS data from online testing: Gains in accuracy diminish as more frequent monitoring data is available. After a certain point, online testing thus may not pay off.

B. Contribution of the Paper

We present the setup and results of extensive experiments to empirically assess how accuracy gains achieved by online testing depend on different factors, including online test rates, usage frequencies, and failure prediction models (see Section II). The experiments are based on QoS data of real-world web-based services.

Based on our experimental findings, we introduce an extension of an existing Web Services monitoring framework. The framework can be parameterized such that online tests are only triggered when the costs of online testing may pay off with respect to accuracy gains. We describe
the conceptual design of the framework (Section III) and the technical implementation of its extended monitoring engine (Section IV). We scrutinize related research efforts (Section V) and conclude our paper with perspectives for future work (Section VI).

II. EXPERIMENTS

A. Goal and Research Questions

The goal of our experiments is to analyze the factors that can influence the gains in prediction accuracy achieved by online testing. In initial, exploratory experiments and analyses, we identified the following factors to have impact on the accuracy gains and thus motivate our research questions:

- **Usage frequency**: The more often a service is invoked in a given time window, the more QoS data from monitoring is available. This leads us to RQ 1: How does service usage frequency influence accuracy gains achieved by online testing?

- **Prediction model**: The models used for anticipating service failures differ in the amount of past QoS data they use and how they analyze the past data. This leads us to RQ 2: How does the prediction model influence accuracy gains achieved by online testing?

- **Online test rate**: The more often a service is tested in a given time window, the more and more timely QoS data is produced, but also increased costs. This leads us to RQ 3: How does online test rate influence accuracy gains achieved by online testing?

- **Failure rate**: The number of times a service fails in a given time window impacts on the amount of actual failures occurring. As observed in the literature, failure rate has a general influence on accuracy gains [7]. This leads us to RQ 4: How does service failure rate influence accuracy gains achieved by online testing?

To address these research questions, we compare prediction accuracy using online testing with the prediction accuracy using monitoring only (i.e., making predictions using only the QoS data from monitoring actual service invocations).

B. Metrics

To measure the accuracy of failure predictions, we employ contingency table metrics, which are commonly used in literature to assess online failure prediction techniques [4]. They are based on the four cases \((TP = \text{true positives}, FP = \text{false positives}, TN = \text{true negatives}, FN = \text{false negatives})\) that may occur when predicting failures (see Fig. 2).

- **Precision** \((p)\): The ratio of correctly predicted failures to all predicted failures: \(p = \frac{TP}{TP + FP}\).
- **Recall** \((r)\): The ratio of correctly predicted failures to all failures: \(r = \frac{TP}{TP + FN}\).

To perform well, a prediction model must achieve both high precision and high recall. However, a trade-off exists between precision and recall. Improving precision, i.e., reducing the number of false positives \((FP)\), often results in worse recall, i.e., increasing the number of false negatives \((FN)\). In order to integrate the trade-off between precision and recall, we use the \(F - \text{measure}\) metric as proposed in the literature [4]:

\[
F = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}
\]

As the above metrics do not take into account true negatives \((TN)\), i.e., correctly predicted non-failures, we use:

- **Specificity** \((s)\): The ratio of correctly predicted non-failures to all non-failures: \(s = \frac{TN}{TN + FP}\).

The higher the values of those metrics, the better the accuracy of a prediction.

C. Setup

In our experiments we focus on response time as the QoS property to predict. To address our research questions, we setup our experiments as follows:

- **Usage frequencies**: We simulate the responses of 7 Web services by randomly retrieving their response times – for the respective point in simulation time – from an existing QoS data set. The QoS data set we employed has been collected by Cavallo et al. [7]. The QoS data was collected by invoking the Web services each hour for a period of about four months, providing up to 2000 data points for our simulation. To analyze
the influence of usage frequencies, we simulate the Web services using the following range of usage frequencies: {0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.08, 0.10, 0.12, 0.16, 0.20}. Table I summarizes the Web services and the associated usage frequencies as used in our experiments.

Table I

<table>
<thead>
<tr>
<th>Service</th>
<th>Usage Frequency</th>
<th>c1 (ms)</th>
<th>c2 (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuickOfTheDay</td>
<td>0.01, 0.02, 0.04</td>
<td>4215</td>
<td>4455</td>
</tr>
<tr>
<td>BLoggivity</td>
<td>0.01, 0.02, 0.04</td>
<td>2530</td>
<td>3180</td>
</tr>
<tr>
<td>StockQuotes</td>
<td>0.01, 0.02, 0.04</td>
<td>2140</td>
<td>2360</td>
</tr>
<tr>
<td>Amazon</td>
<td>0.03, 0.06, 0.12</td>
<td>5910</td>
<td>6360</td>
</tr>
<tr>
<td>GetJoke</td>
<td>0.04, 0.08, 0.16</td>
<td>2624</td>
<td>2900</td>
</tr>
<tr>
<td>Google</td>
<td>0.05, 0.10, 0.20</td>
<td>2930</td>
<td>3250</td>
</tr>
<tr>
<td>XMLDailyFact</td>
<td>0.05, 0.10, 0.20</td>
<td>1753</td>
<td>1862</td>
</tr>
</tbody>
</table>

Table I: Web Services along with the usage frequencies and response time constraints used in the experiment.

Predicion models: We selected typical prediction models which are used for response time prediction [7], [12]. These models work on a series of observed response times: \(m_1, m_2, \ldots, m_{i-1}, m_i\), with \(m_i\) being the response time at point \(i\). They predict the response time for time point \(n + 1\): \(\hat{m}_{n+1}\).

- **Last**: This model is simple and just uses the last observed value as the prediction value: \(\hat{m}_{n+1} = m_n\).
- **Windowed Mean (BM(\(k\))**: In this model, the arithmetic average of the past \(k\) values (\(1 \leq k \leq n\)) is used as the prediction value, where \(k\) is chosen to minimize prediction error: \(\hat{m}_{n+1} = \frac{1}{k} \sum_{i=0}^{k} m_{n-i}\).
- **Simple Exponential Smoothing (SEM)**: BM(\(k\)) treats past observations equally. However, intuitively, the most recent observations should get higher weights than the “older” observations. SEM thus places more weight on more recent observations: \(\hat{m}_{n+1} = \alpha \cdot m_n + (1 - \alpha) \cdot \hat{m}_n\), where \(\alpha \in [0, 1]\).

After initial exploratory runs of our experiments, Last, BM(5), BM(10) and SEM with \(\alpha = 0.3\) led to highest accuracy for monitoring and online testing. Additionally, although Last appears overly simplistic, it showed to be more responsive to abrupt changes in QoS predictions than the other models. Thus, to analyze the influence of prediction models, we vary the prediction model using Last, BM(5), BM(10) and SEM.

Online test rates: To simulate testing the Web services, we randomly use their response times in the QoS data set – for the respective point in simulation time – as the tests results. To analyze the influence of online test rates, we vary the number of online tests using 300, 600, and 1200 tests (that is, 3, 6, 12 online tests per 20 hours).

Fig. 2 illustrates how we make predictions. At each time we obtain a QoS data from simulating the response of a service or testing it, we use a prediction model to predict its next response time. We compare the result of the prediction with the next response time of the service as it is obtained from simulating its responses.

**Failure rates**: Finally, to determine failures of the Web services, we use the response time constrains \(c_1\) and \(c_2\) as summarized in Table I. These constraints provide 20% and 25% failure rates for all the services, and thus, allow us to analyze the influence of failure rates.

D. Execution and Results

Using the experimental setup described above, we have computed the \(F\) and \(s\) metrics by executing 100 simulations for each combination of making prediction using online testing (OT) and using monitoring (M). To quantify the differences between the results of OT and M, we have used the non-parametric Hodges-Lehmann estimator with a 95% confidence interval.

**RQ 1** (influence of usage frequency): Fig. 3-(a) and 3-(c) show the box plots of the absolute values and Fig. 3-(b) and 3-(d) visualize the differences of the \(F\) and \(s\) results using fixed “Last” prediction model, 600 tests and 25% failure rate. The results show that in general online testing improves the accuracy of prediction both in terms of \(F\) (up to 0.20) and \(s\) (up to 0.11). Furthermore, the results show that at the beginning, when the usage frequency is low, the accuracy gain is high, but also that the gain decreases as the usage frequency increases. Thus, the accuracy gain achieved by online testing may pay off as long as the QoS data available from monitoring is little or infrequent.
RQ 2 (influence of prediction model): Fig. 4 shows the empirical results when varying the prediction model for fixed 600 tests and 25% failure rate. The results show that the accuracy gain varies between the models. In addition to Last, BM(10) has the highest accuracy gain when the usage frequency is low. BM(5) comes next and the lowest accuracy gain is achieved by SEM. Furthermore, Last using online testing outperforms the other models. Thus, the accuracy gain achieved by online testing depends on the prediction model used. Nevertheless, the influence of varying prediction models is small when compared to the influence of the usage frequencies.

RQ 3 (influence of online test rate): Fig. 5 shows the empirical results when varying the number of online tests for fixed “Last” prediction model and 25% failure rate. The results show that, for the 300 tests the accuracy gain is lowest and reaches a maximum of 0.14. For the 600 tests it is up to 0.20, and for the 1200 tests it reaches the highest gain of 0.24. However, compared with 600 tests, this is only a small gain of 0.04 while it requires doubling the number of tests indicating a saturation effect.

RQ 4 (influence of failure rate): Finally, Fig. 6-(a) and Fig. 6-(b) show the empirical results when varying the failure rate for fixed “Last” prediction model and 600 tests. The results show that for 20%, the accuracy gain achieved by online testing is low when the usage frequency is low, whereas, it is high for 25%. However, as the usage frequency increases, the accuracy gain of 20% becomes close to that of 25%. To understand the behaviour of PROSA when the usage frequency is low, we additionally show the results for doubling the number of tests (i.e., 1200) using 20% failure rate. This will lead to more data points to be used by the prediction. Fig. 6-(c) shows that the accuracy gain increases as we increase the number of tests. Still, its less than that of the 1200 tests using 25% failure rate (see Fig. 5). Thus, in the case of lower failure rates, when the number of data points used by prediction is low, the accuracy gains achieved by PROSA is also lower. This due to the fact the there are not many failures occurring.

E. Threats to Validity

Concerning the use of metrics for measuring prediction accuracy (construct validity), the metrics we have used depend on the criteria used to determine failures (i.e., response time constraints in our case). To address this, we have varied the response time constraints in our experiment. Concerning the way we have simulated the execution of the service-oriented system and the test cases (internal validity), we have strived to be as realistic as possible. However, we used the QoS data of the Web services provided by [7] to represent both monitoring data and test case execution results. Furthermore, so far we have considered only one test case per service. In future work, we will thus consider live executions of service-oriented systems and test cases by (re-) running extensive experiments using the PROSA framework we describe in Section III. This will also allow us to explore additional influential factors on accuracy gains. Concerning the generalization of the results (external validity), so far, we have considered only response time as the QoS property to predict and a fixed static usage model. In future work, we will thus consider other QoS properties (e.g., availability and reliability) and more complex prediction models (e.g., ARIMA, Markov Chains) while dynamically updating the usage model. Finally, we have made publicly available the QoS data for the Web services we used in our experiments, and all experimental results at: NNN (repeatability).

III. PROSA: FAILURE PREDICTION FRAMEWORK

Based on our our experimental findings in the previous section, we introduce PROSA, an extension of an existing Web service monitoring framework. In this section, we present PROSA at a conceptual level focusing on its main modules and their interactions. In Section IV, the technical description of the Web service monitoring framework is presented.

PROSA is composed of three main modules as shown in
Fig. 7. The Monitoring Module collects information (i.e., QoS Data and Usage Frequencies) about the services. The Online Testing Module actively invokes the services with Test Data based on parameters identified in Section II-A (i.e., online test rates, usage frequencies, and failure prediction models). Following the approach in [13], PROSA utilizes the Monitoring Module to collect QoS Data resulting from monitoring, as well as online testing. Finally, the Prediction Module uses the collected QoS Data to predict failures of the services.

A. Online Testing Module

The Online Testing Module is composed of two key elements:

1) Usage Model Updater: The Usage Model is a key element in PROSA as it represents the actual usage of the services in a service-oriented system. The Usage Model is used to assess whether or not to online test the services.

As shown in Section II-D, accuracy gains achieved by online testing strongly depend on the usage frequencies of the services. Thus, to reflect actual usage of the services, these frequencies need to be determined during runtime. To this end, the Usage Model Updater obtains the Usage Frequencies collected by the Monitoring Engine for updating the Usage Model.

3We use the Technical Architecture Modeling notation, see: http://www.fmc-modeling.org/download/fmc-and-tam/SAP-TAM_Standard.pdf
The Usage Model can be updated using, e.g., simple statistics by computing for each service the number of invocations in a given (moving) time window \( |w| \), or using other more advanced techniques such as exponential smoothing. The Usage Model can be bootstrapped using the estimated usage information resulting from requirements engineering or from past usage logs [14].

2) Online Testing Engine: The Online Testing Engine actively triggers the services in a systematic manner to obtain additional QoS Data. For Online Testing, the Test Data are defined and stored in a Test Data Repository\(^4\). The Online Testing Engine feeds the Test Data (i.e., Test Input) into the Monitoring Engine which then invokes the service under test.

Our approach for triggering online tests is inspired by usage-based testing, which drives the allocation of test cases in accordance with use, and ensures that the most-used operations will be the most tested (cf. [16]). However, our approach ensures that the least-used services will be tested. Hence, PROSA uses the Usage Frequencies in the Usage Model to trigger online tests as follows:

In a subsequent time window \( |w| \), if a service \( S \) is likely to be used with a low frequency, it will be tested to complement the QoS data points from monitoring. Otherwise, it will not be tested. Determining the threshold for the usage frequency (which determines whether the usage frequency is low or high) depends on the application settings (e.g., number of allowable tests, used prediction models, required accuracy, etc...).

It is important to note that online testing services can lead to certain undesired “side effects”. As an example, testing the service of an online book seller may lead to delivering the “ordered” books as defined in the test data. In [3], [17] the authors summarized the potential side effects that may occur during online testing along with solutions that can be employed to minimize their impact. To keep the technical discussions focused, we assume that the online testing side effects can handled by using one the mentioned solutions.

B. Monitoring Module

The Monitoring Module is used to observe the QoS Data by passively monitoring and actively testing the services in the system. During the operation of the service-oriented system, the Monitoring Engine observes the execution of services and logs this Monitoring Data which arrives through Monitoring Events. The Online Testing Module feeds the Test Data into triggers the Monitoring Engine to invoke the service under test. The Monitoring Engine takes this data and coordinates the invocations of the service while observing its QoS. Feeding the Test Data into the Monitoring Module to trigger testing is an important design decision.

As both monitoring and online testing data are observed using the same mechanism, we need to differentiate between the collected QoS Data of the services and the services’ Usage Frequencies, otherwise, online testing results would influence the Usage Model.

The detailed technical description of the Monitoring Module of PROSA is presented in Section IV.

C. Prediction Module

The combined QoS Data coming from the Monitoring Module and the Online Testing Module is fed into the Prediction Engine. The Prediction Engine employs Prediction Models to predict the QoS of the future service invocation. PROSA allows using different Prediction Models such that the ones introduced in Section II-C.

The Prediction Engine compares the predicted QoS with the expected QoS (e.g., response time constraint). Based on the result of the comparison, it reports whether or not the service is predicted to fail (e.g., the predicted response time will be slower than expected).

IV. SALMon: Extended Monitoring Framework

To realize the Monitoring Module of PROSA, we extended SALMon, an existing monitoring framework for monitoring the QoS of Web services\(^5\).

The rationale to use SALMon to implement the Monitoring Module, is that SALMon is a framework that combines both testing and passive monitoring approaches in a single platform, sharing the same infrastructure to gather the QoS data in both of the two approaches. Furthermore, SALMon has been developed as a service-oriented system itself, providing an easy integration on frameworks developed as a service-oriented system. Beyond others, SALMon provides the Monitor Service, which can be invoked through the standard SOAP-based web service protocols.

Fig. 8 depicts the software components of the extended SALMon framework:

A. Monitor Service

The Monitor Service (offered as a Web service\(^6\)) is responsible for retrieving the Monitoring Data of the different services in a service-oriented system. To do so, the Monitor Service creates the required Measure Instruments to obtain the QoS Data. Monitoring is performed by means of an enterprise service bus (ESB) (i.e., in the given service-oriented system, instead of invoking the constituent services directly, all requests and responses are sent through the ESB) which in turn feeds the Measure Instruments. The Monitor Service is configured through the methods provided in its WSDL interface. Through these methods, the services, the

\(^4\)As we focus on QoS testing, we do not need to deal with determining complete test cases and test oracles as we do not check the expected result. The results of testing are the observed QoS (cf. [15]).

\(^5\)See http://gessi.lsi.upc.edu/salmon/web/index.jsp

\(^6\)WSDL description is available at: http://gessi.lsi.upc.edu/salmon/services/Monitor.wsdl
setService(serviceInformation): is used to register the services to monitor. The parameter of the request is ServiceInformation, which includes the following data:

- Name: The name of the service.
- WSDL: the URL of the WSDL file.
- User and Password: If authentication is required, the user and password should be set, it is set blank.
- URL and Port: URL and Port of the endpoint to access the service.
- MeasureInformation: It is used to set the frequency at which the service must be tested (monitoring interval), and also the timeout.

```xml
<xs:element name="SetServiceRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="service" type="tns:ServiceInformation"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

setOperation(serviceID, operationInformation): used to register the operations of a service to monitored. The parameters of the request are the serviceID and OperationInformation. OperationInformation specifies, beyond others, if the operation should be monitored through testing or passive monitoring. Particularly, it includes the following data:

- Name: The name of the operation.
- SOAP Action: The address that identifies the operation within the server in the target endpoint.
- MonitoringInterval: The frequency in which the operation of the service must be tested. (0 for passive monitoring)
- SOAP Message Request: in case of testing, the SOAP message that must be sent to this operation.
- Timeout: The timeout for the calls to this operation.

```xml
<xs:element name="SetOperationRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="serviceID" type="xs:int"/>
      <xs:element name="operation" type="tns:OperationInformation"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

setServiceProperty(serviceID, operationID, metric): is used to set the QoS metric to be monitored. Current implemented metrics are: response time, availability, round trip time and execution time.

```xml
<xs:element name="SetServicePropertyRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="metric" type="tns:Metric"/>
      <xs:element name="serviceID" type="xs:int"/>
      <xs:element name="operationID" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
Once the monitor is configured, the QoS Data and the Usage Frequencies (which are retrieved by the Measure Instruments) can be obtained through the method GetAllInputInformationFromService(serviceID, timeinterval), where timeinterval defines the period of time of the data to retrieve.

The response of this method provides the QoS Data and Usage Frequencies of the service in the following format:

```
<xs:complexType name="QoS">
  <xs:sequence>
    <xs:element name="input" type="tns:Input" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="invocation" type="tns:Invocation" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

B. Measure Instrument

A Measure Instrument is a component that implements the logic needed to obtain the value of a concrete basic quality metric (e.g., current response time, current availability, round trip time, etc.) of a service or operation. Derived metrics are calculated by computing the required formula (e.g., averages, maximums, minimums) over the gathered basic metrics. The concrete set of Measure Instruments are activated dynamically by the Monitor Service depending on the required QoS metrics to measure. Once activated, Measure Instruments receive SOAP requests and responses through an ESB that intercepts the messages. Once the messages are captured, each Measure Instrument computes the required quality metric. In addition to quality metrics, service Usage Data (Usage Frequencies and invocation input) is recorded by the Measure Instruments. The Usage Data is only stored if the invocation has been performed through passive monitoring (i.e., from a real user). The distinction is done by means of the header of the soap message, since the soap requests sent from SALMon testing include a special tag that identifies it.

C. Publisher Service

The Publisher Service implements the Observer Pattern for services in a service-oriented system as described in [18]. The Publisher Service notifies to any subscribed service any relevant state change. This pattern requires that the subscribed services (the observers) implement the required interface to receive such a notification. This is achieved by defining a common WSDL-interface with the notify() method. The rationale behind the Publisher Service is to notify the Monitor Service once the Measure Instruments have collected new QoS Data.

V. RELATED WORK

This section covers the efforts related to the contribution of this paper.

A. Online Failure Prediction for Web-based Services

Shao et al. [19] propose a collaborative filtering framework for quality prediction in the context of service selection. In their approach, they use the observed QoS data of different service consumers (globally distributed) to predict the QoS of a particular user. In the context of proactive adaptation, several approaches have been presented in the literature. For instance, a framework for event-driven QoS prediction was presented by Zeng et al. [20]. The framework exploits event-processing mechanisms to perform real-time predictions to provide early warnings to prevent QoS degradations or violations of commitments. In [21], Shi et al. present an approach for improving QoS prediction accuracy which exploits clustering users into different groups based on locations and network conditions, invocation time and the
workload. In [22], Lorenzoli et al. present a runtime prediction model of service availability based on predicting mean-time-to-failure (MTTF) and mean-time-to-repair (MTTR) measures. However, none of the approaches exploits online testing to lead to gains in the accuracy of online failure prediction.

In addition to predicting failures of individual services, in the literature, here exist approaches for predicting failures of a service-oriented system (or a composition). Although their goal is different, we briefly mention some of them as: 1) some of the approaches use similar prediction models to the ones we use, 2) PROSA could be used to complement these approaches. For instance, in [23], Strunk proposes an algorithm which predicts the QoS-reliability of a service composition by using monitored history of services to predict QoS-violation of the composition using Markov chains. Similarly, Leitner et al. [6] present a framework, coined PREvent in order to predict end-to-end performance violations of service-oriented systems with machine learning techniques. In [24], Aschoff et al. present a framework coined ProAdapt, for proactive adaptation of service composition due to changes in service operation response time, or unavailability of operations, services, and providers. In ProAdapt, the prediction of problems and consider a group of services in a composition flow, instead of isolated services.

As discussed in Section I, in general, all failure prediction approaches rely on QoS data from monitoring. However, due its “passive” nature, the amount and timeliness of the collected data may be limited, which may undermine the accuracy of failure predictions. PROSA complements such approaches by collecting timely QoS data using online testing.

B. Online Testing

In [25], Di Nitto et al. motivated the general need for runtime quality assurance (QA) techniques, such as online testing, for service-oriented systems. In the literature, several techniques and platforms exist that advocate to perform online testing periodically or event-driven (c.f. [2], [10], [26]–[31]) but only for runtime QA. However, exploiting online testing for online failure prediction has not been addressed.

Concerning the evaluation of online testing for services, in [32], Greiler et al. evaluated the capabilities of online testing to detect faults during system reconfiguration that could not be addressed in the test environment. Thus, the focus of their work is runtime QA and not online failure prediction. Additionally, in [11], we presented our initial evaluation of accuracy gains achieved by online testing in the context of online failure prediction. In this paper, we progress that work by extending the experiments to assess the influential factors on the accuracy gains.

C. Combined Monitoring and Testing

For combining online testing with monitoring, two main directions can be observed in the literature. On the one hand, using monitoring data to build the usage profiles or usage models. Bai et al. [33] propose an ontology-based approach for capturing service usage profile by intercepting the SOAP messages, and then log the input/output data to be used in test cases. However, the idea was proposed for test case definition and not for triggering online tests. In [34], Chalagulla et al. presented a reliability assessment and prediction model for SOA-based systems based on combining data from operational profile testing with monitoring data during runtime. Operational profile testing was used at design time and not at runtime. On the other hand, monitoring data was exploited to reduce the number of service invocations when executing a test suite (i.e., reducing the cost of testing). Approaches like [2], [35] used this idea to mimic the service responses. However, they do not complement monitoring data with online testing for failure prediction.

VI. CONCLUSION AND PERSPECTIVES

Online testing promises to improve the accuracy of online failure prediction by complementing passive monitoring. However, online testing can incur additional costs for operating software systems. Understanding when online testing pays off with respect to accuracy gains is thus an important issue for the applicability of online testing. To this end, we presented the results of extensive experiments conducted to empirically assess how accuracy gains from online testing depend on different factors, including online test rates, usage frequencies, and failure prediction models. Based on our experimental findings, we introduced PROSA, an extension of an existing monitoring framework for Web Services. PROSA can be parameterized to trigger online tests when the costs of online testing may pay off (depending on the application settings) with respect to accuracy gains in failure predictions.

Future work includes using PROSA to complement approaches for predicting failures for a service-oriented system or a composite service.

ACKNOWLEDGMENT

We cordially thank NN, NN, NN for very helpful comments on earlier drafts of the paper.

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreements 215483 (S-Cube) and 285248 (FI-WARE).

REFERENCES


Proactive SLA Negotiation for Service Based Systems: Initial Implementation and Evaluation Experience

Khaled Mahbub
Department of Computing
City University London
United Kingdom
K.Mahbub@soi.city.ac.uk

George Spanoudakis
Department of Computing
City University London
United Kingdom
G.Spanoudakis@soi.city.ac.uk

Abstract—This paper describes a framework that we have developed to integrate proactive SLA negotiation with dynamic service discovery to provide cohesive runtime support for both these activities. The proactive negotiation of SLAs as part of service discovery is necessary for reducing the extent of interruptions during the operation of a service based system when the need for replacing services in it arises. The developed framework discovers alternative candidate constituent services for a service client application, and negotiates/agrees but does not activate SLAs with these services until the need for using a service becomes necessary. A prototype tool has been implemented to realize the framework. This prototype is discussed in the paper along with the results of the initial evaluation of the framework.

Keywords—Service discovery, Service level agreements; Proactive SLA negotiation; service monitoring

I. INTRODUCTION

A trustworthy use of software services often requires as a prerequisite the existence of a service level agreement (SLA) between the provider and the consumer of a service. SLAs define quality of service (QoS) and functional properties, which should be guaranteed during the provision of a software service, as well as the penalties that should be applied in cases where the properties are not fulfilled [7][10][11]. An SLA is set through a negotiation between the provider and the consumer of a service [4][12]. SLA negotiation can be particularly complex depending on the requirements and affordances of the two parties. Furthermore, it may need to be carried out at runtime, if a constituent service of a service based client application (SCA) becomes unavailable whilst SCA is in operation, or it fails to perform according to its established SLA. In such cases, SCA should be able to discover alternative replacement services for the failed service, and negotiate SLAs with them at runtime.

To minimize the runtime interruption of the SCA in such circumstances, the discovery of back up replacement services for SCA constituent services should be performed proactively before any of these constituent services becomes unavailable or fails to perform according to its established SLAs [15]. This is important since service discovery and SLA negotiation are time consuming processes that would delay significantly the responsiveness of SCA if they were executed every time that it becomes necessary to identify and use a new service at runtime.

Existing work on service level agreements has focused on SLA specification [13][14], negotiation [5] and monitoring [9]. The need for runtime SLA negotiation or renegotiation has been acknowledged in [2][3][5][10]. Existing approaches, however, are reactive supporting corrective actions only after SLA violations and, thus, they cannot ensure uninterrupted runtime SCA operations when services fail. To address the above shortcoming, we have developed a proactive runtime SLA negotiation tool, and integrated it with a tool supporting proactive runtime service discovery, which has been previously described in [15]. The integrated tools constitute a framework called PROSDIN (PROactive Service Discovery and Negotiation). In PROSDIN, SLA negotiation has been developed as an integrated part of the service discovery process enabling the execution of both activities in a coordinated manner. More specifically, proactive SLA negotiation is performed immediately after the execution of service discovery queries to ensure that adequate SLAs are provisionally agreed for given periods of time with the providers of the discovered services if possible. Also when a pre-agreed SLA expires and it is proactively re-negotiated.

The initial design of our approach to proactive SLA negotiation has been discussed in [24]. The contributions of this paper with respect to [24] are that: (a) it describes the implemented version of RROSDIN, which realises a new version of the SLA negotiation process and the use of a rule-based approach to negotiation using the Jess rule engine [21], and (b) it presents the results of an initial experimental evaluation of the framework.

The rest of this paper is structured as follows. In Sect. II, we discuss the architecture of the service discovery and SLA negotiation framework. In Sect. III, we describe the negotiation process. In Section IV, we provide an overview of the language for specifying SLAs. In Sect. V, we discuss the representation of negotiation rules and the realisation of the negotiation process by the Jess rule engine. In Sect. VI, we describe the results of an initial evaluation of the framework. Finally, in Sect. VII and VIII, we review related work and provide concluding remarks and directions for future work, respectively.
II. FRAMEWORK OVERVIEW

The architecture of PROSDIN is shown in Fig. 1. More specifically, PROSDIN consists of a (runtime) service discovery tool, a service listener, and a negotiation broker. It also interacts with external service registries and is available itself to external service client applications as a service.

The negotiation broker in PROSDIN manages the negotiation process on behalf of service client applications. More specifically, it provides access to different negotiation engines that may be plugged into the framework by translating negotiation rules expressed in the common language of the framework into the different negotiation specifications accepted by these engines (see Sect. V) and realizes the interface for interacting with brokers carrying out the negotiation process on behalf of services (aka service negotiation brokers). The latter may be the same as the broker used by PROSDIN or other brokers that realize the same SLA negotiation interaction interface with it. The interface of the negotiation broker provides operations for: (i) initializing the broker, (ii) starting the negotiation process, and (iii) notifying a broker that an SLA offer (or counter-offer) that has been generated/rejected/accepted by the other party in the negotiation process.

The negotiation process is carried out according to a two-phase protocol that may result in a pre-agreed but not activated SLA or fail. Pre-agreed SLAs have an expiry period within which they can become active, if the service client application decides to activate them. The SLA negotiation process is described in Sect. III.

III. SERVICE DISCOVERY/SLA NEGOTIATION PROCESS

The activity of service discovery and SLA negotiation realized in PROSDIN is shown in Fig. 2.

According to the UML activity diagram in the figure, the process starts with the submission of a service discovery query by an SCA. The initial execution of the query (see Execute Query in Fig. 2) is followed by the build of the set RS. RS includes the best N candidate services, ranked in ascending order of their distance to the query. RS is updated by executing the service discovery query when the framework is informed by the service listener that a new service has become available in the service registry or the description of an existing service has been modified (see the New/Amended Service Description signal in Fig. 2). Hence, the process considers new and updated services.

After RS is initially built or updated, the framework selects the first service in it that does not have a pre-agreed SLA, and starts a proactive negotiation of an SLA with it (see Select Service in RS for Negotiation and Negotiate SLA activities in Fig. 2, respectively). In this phase, the QoS characteristics of the candidate service are negotiated in order to achieve the best possible SLA given the boundary constraints of the two parties.

If the negotiation with a service S fails, S is removed from RS and discovery is re-triggered to find another service to replace it. If negotiation succeeds, a provisional SLA is established and the candidate service in RS is updated to flag the existence of a pre-agreed SLA with it. Subsequently, the process continues by attempting to negotiate SLAs with all the services in RS, which do not have a pre-agreed SLA, until all of them have pre-negotiated SLAs or it is known (through unsuccessful earlier negotiation attempts) that an SLA cannot be established with them.

The negotiated SLAs of services in RS do not come into force immediately. For each pre-agreed SLA, the negotiation
process establishes a time period over which the pre-agreed SLA can be automatically brought into force without further negotiation. This happens when a service with a pre-agreed SLA in RS is selected for binding to SCA. If the validity period of a pre-agreed SLA expires without the candidate service being bound to SCA, the SLA between the service and SCA will be re-negotiated.

Figure 2. Service discovery and SLA negotiation process

Following the selection of a service S in RS for binding to SCA at runtime, its SLA is automatically activated (see Activate SLA in Fig. 2), the service is removed from RS (see Remove Service from RS) and the discovery query is re-executed to identify if there is a new service that could be included in the RS set.

IV. SPECIFICATION OF SLAS

The operation of PROSDIN is driven by specifications of: (a) discovery queries, (b) SLAs and SLA templates, and (c) SLA negotiation rules. In this section we give an overview of the languages for specifying (b) and in Sect. V we overview the language for specifying (c) (the language that is used to specify service discovery queries is beyond the scope of this paper and can be found in [15]).

A. Specification of Service Level Agreements

In PROSDIN, SLA templates, offers, agreed and activated SLAs are specified using an XML schema whose high level structure is shown in Fig. 3. According to this schema, an SLA is specified by an SLA contract element, containing one or more SLA terms. An SLA term specifies one or more guaranteed quality constraints (i.e., constraints over values of QoS attributes). It also refers to the actor(s) who have proposed it in the negotiation process (see Actor element in SLATermType). An actor may take different roles in the negotiation process (e.g., service requester or service provider) and have a negotiation strategy, i.e., a set of rules governing the negotiation process and the communication (e.g. multiphase, multi issue negotiation) with other negotiating parties.

Figure 3. High Level Schema for SLA Specification

An SLA contract may also describe the penalties that will apply in case that any of the parties who have agreed the contract (contractors) fail to fulfill the SLA terms (see the sub-element Penalty). Furthermore, SLA contracts have: (i) a contractID attribute, (ii) an attribute, called status, signifying the status of the contract (i.e., under negotiation, pre-agreed or active), and (iii) a time validity attribute signifying the period for which the contract is valid.

Figure 4. SLA Specification – Constraint element

Figure 5. Example SLA
Fig. 4 shows the part of the SLA schema that is used to specify constraints for SLA terms. A constraint is defined as an atomic logical expression or a conjunction/disjunction of two or more logical expressions. Atomic logical expressions are conditions over quality attributes of services. These conditions are defined as a relation between two arguments (e.g., equalTo, lessThan, greaterThan) and can be negated. The arguments of a relation can be a quality attribute of a service, constant, or an arithmetic expression over quality attributes and constants.

Fig. 5 shows an example of a pre-agreed SLA for service X between a company XYZ that provides X and the service consumer C. The SLA sets a conjunction of two conditions. The first of these conditions states that the availability of the service should be greater than 80%, and the second condition states that the response time of the service should be less than 9 milliseconds.

V. NEGOTIATION RULES AND BROKER

The SLA negotiation process in PROSDIN is executed by the negotiation broker according to negotiation rules. These rules are specified using the XML schema that is partly shown in Fig. 6. This schema allows the expression of negotiation rules as condition-action rules of the form: IF (condition) THEN (action) ELSE (action).

The conditions in the negotiation rules are either atomic conditions or logical combinations of atomic conditions over QoS attributes of services having the same structure as the SLA term conditions discussed in Sect. IV. Rule actions can be of three types: (i) accept actions that are used to accept the value of one or more QoS attributes in a given SLA offer, (ii) reject actions that are used to reject the value of one or more QoS attributes in a given SLA offer, and (iii) set actions that are used to propose a new value or range of values for one or more QoS attributes as part of an SLA offer.

An example of a negotiation rule is shown in Fig. 7. The rule is used by the negotiation broker of a service provider and states that if the consumer of a service has made an offer (or counter-offer) where the response time of the service must be less than 10 milliseconds (ms) and the price to be paid per service use is 0.5 pounds, the offered values will be accepted.

The negotiation rules expressed in the common XML language of PROSDIN are translated into the negotiation specification of the particular negotiation engine plugged into the broker. The negotiation engine used in the current implementation of the framework is a rule-driven negotiation engine that we have developed based on Jess [17].

A rule in Jess has the form

\[
(defrule rule-name
  (logical-operator (cond-i . cond-n))
  => action)
\]

where cond-i is defined as an atomic condition of the form \((\text{fact-pattern-i} <\text{cond-i}>)\) or a complex logical condition over such atomic conditions. The fact patterns in a rule define logical conditions over facts known to the Jess engine. Jess uses a form of the algorithm Rete [6] to match rules against facts and when a match is found the actions specified in a rule are taken. Such actions can assert or modify the values of facts.
Based on the transformations listed in Fig. 8, the Jess rule generated for rule1 is:

\[
\text{(defrule rule1} \\
\text{ (SLA [RESPONSE\_TIME\_CONSUMER < 10] [PRICE-CONSUMER = 0.5]) } \Rightarrow \text{ (modify 0 } \\
\text{ (PRICE\_PROVIDER} \\
\text{ (fact-slot-value 0 PRICE\_CONSUMER)))} \\
\text{(modify 0 (RESPONSE\_TIME\_PROVIDER) } \\
\text{ (fact-slot-value 0 RESPONSE\_TIME\_CONSUMER)))})
\]

VI. IMPLEMENTATION & EVALUATION

All the major components of PROSDIN (i.e., the negotiation broker, service discovery tool, and service listeners) have been implemented in Java and are available as a web service. This service can be deployed by service client applications programmed in a way that can notify service discovery queries and SLA negotiation rules to PROSDIN, and receive endpoints of discovered services with negotiated SLAs from it. The external service registry used in the current implementation is a faceted registry as the one developed by the SECSE project [22]. This registry has been implemented using eXist [18] database and is accessed by PROSDIN through Java remote method invocation (RMI).

To evaluate the implementation of PROSDIN, we performed a series of experiments. The purpose of these experiments was to: (a) measure the overhead of SLA negotiation (whether reactive or proactive) on the execution time of the runtime service discovery process, and (b) assess the effectiveness of proactive SLA negotiation over reactive SLA negotiation during runtime service discovery process.

A. Experimental Setup

In the experiments, we have used an SCA, called Route-Planner, as a case study. Implemented by a BPEL service orchestration process, this system allows a user to find an optimal route from his/her current location to another location by using a Global Positioning Service (GPS), and displays electronic maps of the area where the user is located and the identified route between two points. The latter functionality is supported by the use of an electronic map service (eMapS). The service discovery query used in the experiments was specified in order to identify candidate replacement services for the GPS service of Route-Planner. The query expressed structural discovery criteria and a soft quality constraint. The structural criteria referred to the required (WSDL) interface for possible alternative services that could be used in the place of the GPS service should this service fail at runtime, and the quality constraint expressed a condition about service availability.

In the experiments, we also used an SLA template with four QoS terms for negotiation. These QoS terms were related to the service price, availability and response time, and the mean number of service requests per hour. For negotiation, we specified a set of 15 service consumer negotiation rules (CNR set), and 20 different sets of provider negotiation rules (PNR sets). Each of the PNR sets contained between 5 and 20 negotiation rules. During negotiation with each of the candidate services identified by the discovery process, the negotiation broker of the service provider side picked up randomly one of the PNR sets and carried out the negotiation based on it. In this way, we simulated the different behaviour that different service providers who participate in the negotiation process might have. The specifications of the SLA template, CNR and PNR sets and discovery query used in the experiments cannot be listed here due to space restrictions but can be found in [19].

To assess whether the number of considered services affects the performance of the service discovery and SLA negotiation processes, we performed the experiments with three different service sets (registries). These sets contained 100, 300 and 500 services, respectively and were populated by geographic location related services taken from the SEEKDA [20] and the SECSE service registry [22]. Each service that was used in the experiments had a WSDL (i.e., a structural) description and a quality of service description. These descriptions were used during the service discovery process.

All experiments were carried out using a Pentium 2.33 GHz with 3.23 GB RAM machine.

B. Results

In the experiments we measured the time needed to:

(a) Build the initial RS set (see Sect. II);

(b) Maintain the RS set at runtime due to the arrival of new services (type_1 events) or change in the description of an existing service in the registry (type_2 events); and

(c) Select a service for replacing a service S in the service based application due to unavailability of S (type_3 events).

The times required for (a), (b) and (c) were measured, for executions of the service discovery process without SLA negotiation (SD Only), with proactive SLA negotiation (SD with Proactive SLA), and with reactive SLA negotiation (SD with Reactive SLA). Table I presents the formulas used to measure execution times in cases (a), (b) and (c).
Table II presents the time needed to build the initial replacement services set RS. As expected, the total time required for building the RS set for a given service in the case of service discovery with proactive SLA negotiation is longer than the time required for building the same set in the cases of service discovery without SLA negotiation and service discovery with reactive SLA negotiation. This difference was observed across all the different sizes of service registries and occurred because when service discovery with proactive SLA negotiation is used, an SLA should be negotiated and (possibly) pre-agreed with each candidate service, whilst when the initial construction of the RS set is based on service discovery only or on service discovery with reactive negotiation, no SLA negotiation is required.

Overall, during the initial phase of building the RS set, the use of proactive negotiation has an overhead between 8% (500 services) and 7% (100 services) of the time required for pure service discovery. However, it should be noted that the initial phase for building RS is performed only once for each subscribed query, and in parallel to the execution of the service client application.

The time needed for maintaining the replacement service set (RS) and selecting a replacement service due to events of type 1, type 2, and type 3 is shown in Table III. The time measures shown in the table are averages (and total sums where applicable) taken across five different executions of the discovery query for each of the three event types.

As shown in Table III, the time required to select a replacement service in case of service discovery with proactive SLA negotiation is slightly larger than the time required to identify a replacement service if service discovery without any SLA negotiation is used. This is because in the former case, the pre-agreed SLA needs to be activated before the replacement service is returned. It should be noted, however, that the main benefit shown in the table is that the time required to select and bind a replacement service at runtime in the case of service discovery with reactive SLA negotiation is significantly larger than the service selection and binding time in the case of service discovery with proactive SLA negotiation: 53.2 vs. 453 milliseconds for the registry with 100 services, 45.8 vs. 459.4 milliseconds for the registry with 300 services, and 50.2 vs. 443.8 milliseconds for the registry with 500 services.

Table IV shows the aggregate effectiveness of proactive SLA negotiation in absolute and relative terms by presenting the AG_pro, RG_pro, and BE_pro measures. More specifically, as shown in the table, the average gain in service replacement time of service discovery with proactive SLA negotiation over service discovery with reactive SLA negotiation is between 187 ms and 230 ms per service replacement (see the AG_pro column in the table).

**Table I. Basic Time Measures**

<table>
<thead>
<tr>
<th>Time</th>
<th>Definition/Calculation</th>
</tr>
</thead>
</table>
| t_match       | This is the time needed to execute a service discovery query against n services from the registry. This is calculated as: $t_{match} = t_{neg} + t_{struct} + t_{non-context}$, where:  
- $t_{neg}$ is the time needed to retrieve n services from the registry.  
- $t_{struct}$ is the time needed to evaluate the structural constraints of a query against n services.  
- $t_{non-context}$ is the time needed to evaluate non contextual constraints of a query against n services. |
| t_SLAMNeg     | This is the time needed to perform SLA negotiation with n services.                      |
| t_SLAMAct     | This is the time needed to activate a pre-agreed SLA.                                    |
| t_DEL          | This is the time needed to delete a service from the candidate service set (RS) and pick up the best service from RS.                             |
| t_ADD          | This is the time needed to add a new service to RS and/or sort the services within RS according to their total distance to the query used to build RS. |
| tADD           | This is the time needed to select an alternative service for replacement due to unavailability of a service. This time is calculated as follows,  
- SD Only case: $t_{ADD} = t_{SLAMNeg}$  
- SD with Proactive SLA case: $t_{ADD} = t_{SLAMNeg} + t_{match}$  
- SD with Reactive SLA case: $t_{ADD} = t_{SLAMNeg} + t_{match}$ |
| tDEL           | This is the time needed for runtime maintenance of the candidate service set (RS) due to arrival of a new service or change in the specification of an existing service. This time is calculated as follows,  
- SD Only case: $t_{DEL} = t_{SLAMNeg}$  
- SD with Proactive SLA case: $t_{DEL} = t_{SLAMNeg} + t_{match}$  
- SD with Reactive SLA case: $t_{DEL} = t_{SLAMNeg} + t_{match}$ |
| tSLAM         | This is the time needed for runtime maintenance of the candidate service set (RS) due to event of service discovery with proactive SLA negotiation. This time is calculated as: $t_{SLAM} = \frac{\sum_{n=1}^{N} (t_{SLA-Neg} + t_{match})}{N}$ |
| tSLAM         | This is the time needed for runtime maintenance of the candidate service set (RS) due to event of service discovery with reactive SLA negotiation. This time is calculated as: $t_{SLAM} = \frac{\sum_{n=1}^{N} (t_{SLA-Neg} + t_{match})}{N}$ |
| tSLAM         | This is the time needed for runtime maintenance of the candidate service set (RS) due to event of service discovery with reactive SLA negotiation. This time is calculated as: $t_{SLAM} = \frac{\sum_{n=1}^{N} (t_{SLA-Neg} + t_{match})}{N}$ |

**Table II. Time Measures for Building Candidate Service Set (Times in Seconds)**

<table>
<thead>
<tr>
<th>Time</th>
<th>SD Only</th>
<th>SD with Proactive SLA</th>
<th>SD with Reactive SLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>152.062</td>
<td>383.29</td>
<td>642.464</td>
</tr>
<tr>
<td>300</td>
<td>138.152</td>
<td>378.995</td>
<td>620.312</td>
</tr>
<tr>
<td>500</td>
<td>151.447</td>
<td>382.888</td>
<td>643.858</td>
</tr>
</tbody>
</table>

**Table III. Time Measures for Building Candidate Service Set (Times in Seconds)**

<table>
<thead>
<tr>
<th>Time</th>
<th>SD Only</th>
<th>SD with Proactive SLA</th>
<th>SD with Reactive SLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>8.39</td>
<td>24.326</td>
<td>43.561</td>
</tr>
<tr>
<td>300</td>
<td>8.046</td>
<td>24.058</td>
<td>40.78</td>
</tr>
<tr>
<td>500</td>
<td>8.17</td>
<td>24.712</td>
<td>43.687</td>
</tr>
</tbody>
</table>

**Table IV. Aggregate Effectiveness Measures**

<table>
<thead>
<tr>
<th>Measure</th>
<th>SD Only</th>
<th>SD with Proactive SLA</th>
<th>SD with Reactive SLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG_pro</td>
<td>53.2</td>
<td>453.0</td>
<td>459.4</td>
</tr>
<tr>
<td>RG_pro</td>
<td>45.8</td>
<td>459.4</td>
<td>443.8</td>
</tr>
<tr>
<td>BE_pro</td>
<td>50.2</td>
<td>443.8</td>
<td>459.4</td>
</tr>
</tbody>
</table>
Also, in relative terms, the service replacement time with reactive SLA negotiation presents an 8.5 to 10-fold increase over the service replacement time when proactive SLA negotiation is applied (see the RG_init column of the table). This relative increase is anything but negligible when considering that the need for service replacement arises whilst an SCA is in operation. Also, the cost of maintaining the replacement service set (RS) does not exceed the gain achieved by service discovery with proactive SLA negotiation (see the BE_neg column in Table IV).

Overall, albeit preliminary, our experiments have shown that proactive SLA negotiation can provide a substantial improvement of the time that will be required for dynamic replacement of services when agreed SLAs must be in place before using a service.

### Table III. Performance Measures for Maintaining Candidate Service Set and Service Replacement (Times in Milli-seconds)

<table>
<thead>
<tr>
<th>Replacement Service Set (RS) Maintenance</th>
<th>SD Only</th>
<th>SD with Proactive SLA</th>
<th>SD with Reactive SLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg(t_init) × RUR</td>
<td>690.5</td>
<td>698.3</td>
<td>730.5</td>
</tr>
<tr>
<td>avg(t_rep) × RUR</td>
<td>690</td>
<td>698.3</td>
<td>673.5</td>
</tr>
<tr>
<td>Selection of Replacement Service from RS</td>
<td>22</td>
<td>28</td>
<td>21.8</td>
</tr>
<tr>
<td>avg(t_init) × RUR</td>
<td>110</td>
<td>140</td>
<td>109</td>
</tr>
</tbody>
</table>

### Table IV. Effectiveness of Proactive SLA Negotiation

<table>
<thead>
<tr>
<th>Service Registry</th>
<th>AGneg</th>
<th>RGneg</th>
<th>BE_neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>187</td>
<td>8.51</td>
<td>0.999</td>
</tr>
<tr>
<td>300</td>
<td>230.4</td>
<td>10.03</td>
<td>1.028</td>
</tr>
<tr>
<td>500</td>
<td>192.8</td>
<td>8.84</td>
<td>1.089</td>
</tr>
</tbody>
</table>

It should also be noted that although proactive service discovery and SLA negotiation are essential for achieving efficient service replacement at runtime, they also create the possibility of inefficient resource utilization. More specifically, the efficiency of resource utilization with proactive SLA discovery/negotiation over a period of time T can be measured by the formula:

$$ U = \frac{T \times SRRR \times (t_{match} + t_{SLA-Neg})}{T \times SRUR \times (t_{init-RS} + t_{SLA-Neg}) + t_{init-RS}} $$

In this formula, SRRR is the service request replacement rate; SRUR is the service registry update rate; t_{match} is the average time required to match a query with a service; t_{SLA-Neg} is the average time required to negotiate an SLA with a service; and t_{init-RS} is the time needed to build the initial copy of RS (t_{init-RS} = R_{init} \times t_{match} + t_{SLA-Neg} where R_{init} is the number of services in the service registry at the time of the initial build of RS).

Hence, to have efficient resource utilization when deploying proactive service discovery and negotiation, it should be that $SRRR \geq SRUR + R_{init}/T$ or that $SRRR \geq SRUR$ since the factor $R_{init}/T$ becomes arbitrarily close to zero as $T$ increases. This means that the service replacement request rate must be higher or at least equal to the service registry update rate. Establishing the validity of this condition would require a long-term study. However, it is not unreasonable to expect that the condition $SRRR \geq SRUR$ holds in the long term.

### VII. Related Work

Proactive approaches to dynamic adaptation of service-based applications are increasingly appearing in the literature [16][23]. Most of the work in this area, however, focuses on mechanisms for forecasting operational problems that may require adaptation (see [16][23] for example) rather than focusing on proactive SLA negotiation. Also, existing work on SLA negotiation tends to focus on the mechanics of the negotiation process itself (e.g., [4][10]) rather than wider procedural issues as to when and under what conditions the negotiation process may be triggered.

An agent based framework for SLA management is presented in [9]. In this framework, an initiator agent from the service consumer’s side and a responder agent from the service provider’s side take part in the negotiation process. The responder agent advertises the service level capabilities and the initiator agent fetches these advertisements and initializes the SLA negotiation process. Different stages of SLA life cycle e.g. formation, enforcement and recovery is performed through the autonomous interactions among these agents. In the case of an SLA violation, the initiator agent may either claim compensation and renegotiate with the service provider or select a new service provider. The provision of compensation in case of violation of SLA is also the focus of [1]; an approach focusing on several aspects of operational problems that may require adaptation (e.g., [4][10]) rather than wider procedural issues as to when and under what conditions the negotiation process may be triggered.

Runtime SLA re-negotiation has been suggested in [2][3][4][7][5] to manage SLA violations. In [2] service level objectives are revised and renegotiated at runtime and deployed services are adjusted to dynamically agreed service level objectives. A similar approach allowing the change of service level objectives whilst keeping the existing SLA is described in [5]. In [3] a renegotiation protocol is described that allows the service consumer or service provider to initiate renegotiation while the existing SLA is still in force when this becomes necessary for service providers or consumers for different reasons (e.g., changes in the business requirements of a party).

Note, however, that all the above approaches are reactive, i.e., renegotiation starts only after an existing SLA is violated. Hence, they do not address the main
problem that is the focus of our work, i.e., the development of a proactive SLA negotiation approach that can increase the chances of uninterrupted service provision when SLA negotiation is required at runtime. Furthermore, our framework integrates SLA negotiation with dynamic service discovery and it can, therefore, provide integrated runtime support for both these key activities, which is necessary for achieving runtime service based application with minimized interruptions.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a framework that integrates service discovery with proactive SLA negotiation, called PROSDIN.

The identification of alternative services in PROSDIN is based on various characteristics of published services including structural, behavioural and QoS characteristics. PROSDIN also negotiates a service level agreement over QoS levels with each alternative service identified by the discovery process. The negotiation process is carried out according to a two-phase protocol and may result in a provisionally agreed but not activated SLA or negotiation failure. A provisional SLA has an expiry date by which it should either be activated or cease to exist.

The objective of proactive SLA negotiation in PROSDIN is to ensure that a service, which could be potentially used by a service client application, will have an agreed set of guaranteed provision terms if the need to deploy it arises at runtime. Hence, when this need arises it won’t be necessary to engage in a lengthy negotiation process interrupting the operation of the service client application.

Our approach has been evaluated through an initial set of experiments showing that proactive SLA negotiation leads to significant reduction of the time required to perform service replacement at runtime if the existence of agreed SLAs is a prerequisite for service use.

PROSDIN opens a spectrum of possible lines for future investigation. These include support for proactive negotiation of hierarchical SLAs, i.e., SLAs of complex composite services deploying other composite services with their own sub-SLAs which will need to be negotiated separately and before coming to an higher level service level agreement. Other aspects for further investigation include the use of heuristics for tuning the triggering the proactive SLA negotiation process so as to reduce the number of cases where pre-agreed SLAs never get used, and the study of the performance of the framework when the negotiation rules used by different participants might change dynamically.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community’s 7th Framework Programme under Grant Agreement 215483 (S-Cube).

REFERENCES

SALMonADA: A platform for Monitoring and Explaining Violations of WS–Agreement–compliant Documents

C. Müller(1), M. Oriol(2), M. Rodríguez(2), X. Franch(2), J. Marco(2), M. Resinas(1), A. Ruiz–Cortés(1)
(1) University of Seville, LSI, Seville (Spain), ISA research group, http://www.isa.us.es/
{cmuller,resinas,aruiz}@us.es
(2) Universitat Politècnica de Catalunya, Barcelona (Spain), GESSI research group, http://www.essi.upc.edu/~gessi/
{moriol,jmarco}@lsi.upc.edu, {marcr,franch}@essi.upc.edu

Abstract—Quality assurance techniques have been developed to supervise the service quality (QoS) agreed between service-based systems (SBSs) consumers and providers. Such QoS is usually included in service level agreements (SLAs) and thus, SLA monitoring platforms have been developed supporting violation detection. However, just a few of them provide explanation of the violations caused by observed QoS at monitoring time, but not in an user-friendly format. Therefore, we propose a general monitoring and analysis conceptual reference model and we instantiate it with SALMonADA, a SBS that notifies the clients with violations and their causes in their own easy-to-understand specification terms. In addition, our platform performs an early analysis notification that avoids delays in the client notification time when a violation takes place. Moreover, we have implemented a web application as a SALMonADA client, to prove how it monitors, analyses and reports to their clients the service level fulfillment of real services subject to a SLA specified with WS–Agreement.

Keywords—monitoring; analysis; violation detection; violation explanation;

I. INTRODUCTION AND MOTIVATION

Service level agreements (SLAs) establish the service quality (QoS) agreed between service-based systems (SBSs) consumers and providers, and thus, quality assurance techniques are needed to supervise the SLAs fulfillment. These techniques require monitoring platforms enhanced with analysis capabilities to reason about the monitored information in order to extract useful information for the parties.

Many research efforts have been made trying to obtain useful monitoring information, starting from general monitoring framework [1], [2]. Thus, several proposals can be found providing a different kind of information from monitored SLAs, such as: violation detection in the SOA testing context [3], [4]; asynchronous violation detection reports to subscribed clients that wait for the monitoring information instead of requesting it [5], [6]; event-based violation explanation in SBSs [7], [8]; and dynamic SBS adaptation when a SLA violation is detected.

In this paper we propose a general monitoring and analysis conceptual reference model in which several agents extract useful information from SLAs at monitoring. For that purpose, three kinds of documents are handled by agents, namely: SLAs, monitoring management documents (MMDs) to configure and manage the monitors, and service level fulfillment (SLF) to report the violations and their causes. In addition, we instantiate the conceptual model with SALMonADA, a service-based system (SBS) that integrates upgraded versions of previously developed SLA monitoring (SALMon [9]) and analysis (ADA [10]) proposals. SALMonADA provides the following contributions to such techniques with the aim of extracting useful information at SLAs monitoring: first, it notifies the client with violations and their causes in their own easy-to-understand specification terms; second, it supports expressive and easy-to-understand SLAs specified with WS–Agreement [11]; and finally it performs an early analysis notification that supports the SLA fulfillment analysis when a violation has just been observed, reducing the client notification time.

Proposals like [12] and [13] assuming the availability of a monitoring and analysis engine, benefit from using SALMonADA since they are provided with such a service level fulfillment information needed to adapt the SBS, renegotiate the SLA, achieve reputation statistics, etc.

The paper is organised as follows. Related work is revised in Section II. WS–Agreement specification is introduced in Section III including an example of our supported SLAs. The conceptual reference model is detailed in IV, while its SALMonADA instantiation is included in V. Section VI and VII detail the monitoring and analysis SALMonADA components, respectively. Section VIII reports an evaluation of our proposal. And Section IX concludes the paper with a discussion of contributions.

II. RELATED WORK

On the one hand, many monitoring frameworks have been proposed, [1], [2], [7] considering monitoring and even analysis agents, but to the best of our knowledge, none of them propose agents to manage SLAs, MMDs, and SLFs, as separated documents.

On the other hand, several techniques to extract useful information at SLA monitoring have been developed. Thus,
Table I

<table>
<thead>
<tr>
<th>EC Formula lines</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀ t1 : time exists t2 : time</td>
<td>-</td>
</tr>
<tr>
<td>Happens (ir:getRate(ID,country2,country1), t1,R(t1,t1))</td>
<td>True</td>
</tr>
<tr>
<td>Happens (ir:getRate(ID,t2,R(t1,t2))</td>
<td>True</td>
</tr>
<tr>
<td>ociself:sub(t2,t1)&lt;100</td>
<td>False</td>
</tr>
</tbody>
</table>

we can find proposals providing violation detection for WS–Agreement documents in the SOA testing context [3], [4]. However, such testing proposals monitor the service to detect violations at testing and not while the service is consumed. Other proposals such as [5], [6], dealing the latter with non–WS–Agreement documents, provide asynchronous violation detection reports to subscribed clients that wait for the SLA monitoring information instead of requesting it, as commonly performed by other proposals. Moreover, there are proposals such as [14], [15], [16], [17] that go further and when they detect a SLA violation they dynamically adapt the SBSs following different strategies, but this dynamic reaction is out of the scope of the paper.

As far as we know, there is only a set of proposals from Mahbub and Spanoudakis that provides violation explanation in SBSSs [7], [8]. Such proposals use event calculus (EC) and they report an event-based explanation of the SBS violation as follows: "The operation event has violated the EC formula \( F \) of the term \( T \)." For instance, Table I depicts a four-lines EC formula that is reported as explanation to a client informing about the response time violation of the \( \text{getRate} \) operation. Such a formula is included inside WS–Agreement document terms.

### III. WS–AGREEMENT IN A NUTSHELL

The WS–Agreement recommendation [11] describes both an XML–based language and a protocol that facilitates the publication, discovery, and monitoring of SLAs between two parties, usually a service provider and a service consumer. The SLAs are created after a negotiation process and they comprise an agreement identifier, an agreement context containing information about the involved parties, and agreement terms that describe both the characteristics of the services to be provided in service terms and the guarantees on such services in guarantee terms. Note that WS–Agreement only defines the general structure of a SLA and the kind of terms it may include. However, it does not specify any vocabulary to express the features of the service.

Service terms are divided into two elements: first, service description terms that define the features of the service that will be delivered under an agreement; and second, service properties that define named, service–related sets of variables that can be used for the specification of guarantee terms and must be therefore considered for agreement

1This sample is included in [7] at page 26.
IV. CONCEPTUAL REFERENCE MODEL

In this section we propose a conceptual architecture as a conceptual reference model. This architecture can be instantiated by several design architectures for monitoring SLAs, ensuring a decoupled structure between the monitoring of the services and the analysis of the SLA compliance. We illustrate the architecture using the SAP-TAM Notation [18].

We briefly describe here the different agents of the system, as depicted in Fig. 2, with their required responsibilities.

**Client:** is the user of the platform. The responsibilities of the client are to provide the SLA to monitor, and its goal is to retrieve the results of the monitored SLA, structured in a document named Service Level Fulfillment (SLF). It is important not to assimilate the SALMonADA client with the consumer of the service (the SALMonADA client could either be the consumer of the service, the provider or a third party interested in monitoring the assessment of the SLA).

**SLA Manager:** is an agent responsible for retrieving and managing the monitored SLAs (storage, provisioning, deletion...). The SLAs are stored in a SLA Repository.

**SLA2MMD:** is an agent that decouples the SLA Manager from the MMD Manager. It works as a bridge between the SLA (a contractual specification understood by SLA-dependent agents) with the MMD (a specification of the monitoring directives to configure a monitor).

**MMD Manager:** Similarly to the SLA Manager, the MMD Manager is responsible of managing the MMD documents. It provides the Monitoring directives to the Monitor, and it is used to provide an updated MMD with the monitored values. The MMDs are stored in a repository.

**Monitor:** is the agent responsible of monitoring the services QoS. The observed QoS are stored in a repository.

**Analyzer:** is the agent used to check if the monitored QoS of a service (obtained through the MMD with values) is compliant with the agreed QoS included in the SLA.

The proposed architecture of the conceptual model provides a reference to instantiate the different agents with concrete solutions, establishing a clear separation of concerns on the management of the SLAs, the MMDs and the SLFs.

V. THE SALMONADA PLATFORM

In this section we provide a design architecture as an instantiation of the previous conceptual model. We propose SALMonADA, a platform able to monitor SLAs specified in WS–Agreement. The proposed solution combines two existing frameworks that have been extended to realize this project: SALMon[9] and ADA[10]. We also describe in this section the two approaches that SALMonADA has to obtain the results of the analysis.

A. Design Level Architecture

As shown in Fig. 3, we have developed SALMonADA as a SBS with the following elements:

**Client:** provides the SLA to monitor expressed in WS–Agreement. It is able to retrieve either the MMD or the SLF, and if desired, it can also receive notifications when the SLA has been violated (see subsection V-B).

**SALMonADA composer:** is the service that composes the internal services of the platform. It provides the interface to the client and manages the execution process of the system. It also adds an independence layer on the interaction required between the analysis of the SLAs (performed by ADA) and the monitoring of the services QoS (performed by SALMon). Such a decoupled structure allows to add or modify the internal components in a very flexible manner. (i.e. allows to replace the monitor or the analyzer without affecting the other elements of the platform).

**SALMon:** is the service responsible for monitoring the services QoS. It acts as both the MMD Manager and Monitor agents of the conceptual model. A detailed description of the SALMon behavior is included in section VI.

**ADA:** is the service responsible of managing and analyze the different WS–Agreement documents. It supports the analysis of WS–Agreements with expressive assertions inside guarantee terms. It acts as both the SLA Manager and Analyzer of the conceptual model. A detailed description of the ADA behavior is included in section VII.

**MMD Parser:** is the service that implements the SLA2MMD agent extracting the monitoring information, and it also implements the functionality to interact with the
MMDs (retrieve or update values). Thus, the MMD structure, whose information is used by all platform components, is decoupled from both ADA and SALMon. Therefore, different MMD structures can be developed, if needed.

### B. The Asynchronous and Synchronous Approaches

SALMonADA platform is designed and developed to support asynchronous and synchronous interaction styles with their clients. Thus, a client, based on its own benefit, may choose its preferred approach. Independently of the selected approach a client must start and stop the SALMonADA monitoring to be subscribed/unsubscribed as client. The start process requires a WS–Agreement document to monitor its fulfillment and such a process slightly varies for clients using asynchronous approach because they must also provide the notification endpoint. Where the notification is awaited (notification endpoint).

**Asynchronous approach:** it is the most convenient way to interact with the platform due to the asynchronous nature of SALMonADA service monitoring and analysing. In this sense, the platform incorporates an early analysis notification that support the SLA fulfillment analysis as soon as a violation is observed. Thus, the SLF notification is sent to the client without more delay than the analysis time. Therefore, SALMonADA notifies their clients only when the monitored service has just been used by a service consumer and a SLA violation is incurred. As depicted in sequence diagram of Fig. 4, once the client has started to monitor, the provider service included in the reported WS–Agreement document is monitored by the SALMon component. Next, the MMD created from the monitored WS–Agreement document is sent to the MMD Parser with monitored measures to be updated. Finally, the new MMD is notified to the SALMonADA composer that sends it to the ADA component to analyse the service level fulfillment of the corresponding WS–Agreement document (cf. Section VII for more details). Then, the client is notified about such SLF informing about: the WS–Agreement document fulfillment or not; and in the latter case, both: the specific violated WS–Agreement terms and the violating metrics, are included as violation explanation. Section VIII includes an example of how this SLF is reported to users. Note that SALMonADA supports the same endpoint acting as different clients, for instance, one of them to get the SLF, other to store reputation analytics of the service consumer and provider, or even to perform self-adaptation strategies.

**Synchronous approach:** it permits the client to control when SALMonADA operations are requested to get: the current MMD with the most recent monitoring information obtained by SALMon; or the current SLF of the WS–Agreement document analysed by ADA. However, the availability of new monitoring information is not assured by the platform due to the aforementioned asynchronous nature of its monitoring and analysis. Thus, it is possible for the client to get the same monitoring information in consecutive MMD requests. Only if the SALMonADA client is acting as consumer or provider service, the availability of new monitoring information is known.

### VI. SALMon COMPONENT IN A NUTSHELL

SALMon is a framework aimed at monitoring the QoS of services [9]. It has been developed as a SBS itself, providing hence an easy integration on frameworks developed as a service-oriented system, such as Self-Adaptive SBS [19] or Cloud monitoring [20] frameworks. In this work, SALMon has been enhanced with the MMD Manager Service, which can be invoked through standard SOAP-based web service protocols. The MMD Manager Service, in turn, invokes the already existing Monitor Service to configure the monitor. Fig. 5 depicts the extended SALMon components.

**MMD Manager:** is the service that stores the MMDs in the repository and configures the Monitor accordingly. To do so, it parses the MMD using the MMD Parser component.

**Monitor Service:** The Monitor Service is responsible for retrieving the QoS of the services. To do so, it creates the required Measure Instruments to obtain the QoS Data. Monitoring is performed by means of an Enterprise Service Bus (ESB) (i.e., instead of invoking the services directly, all...
requests and responses are sent through the ESB) which in turn feeds the Measure Instruments.

*Measure Instrument:* is the component that obtains the values of a basic quality metric, whereas derived metrics are calculated by computing the required formula (e.g. average). The Measure Instruments are activated depending on the quality metrics to measure. Once activated, they receive the SOAP messages through the ESB that intercepts them.

*Publisher Service:* implements the Observer pattern for services in a SBS. It notifies to any subscribed service any relevant state change. This pattern requires that the subscribed services (the observers) implement the required interface to receive such a notification. This is achieved by defining a common WSDL-interface with the notify method.

### VII. ADA COMPONENT IN A NUTSHELL

ADA is an *Agreement Document Analysis* framework aimed at extracting useful information from agreement documents at any SLA life-cycle stage [10]. It has been developed based on our previous theoretical works on applying the constraint satisfaction problem (CSP) [21] paradigm to the automated procurement of web services [22] and explanation of WS–Agreement document inconsistency and non-compliance situations [23], [24]. The main ADA features are: (1) *interoperability* through a triple distribution model, namely: as a Java library, as an OSGi service and as a web service; and (2) *solver independent* through the use of a semantic mapping between WS–Agreement documents and CSP paradigm, that protects our design from the possible variations derived from using different solvers. In this work, ADA has been enhanced with the ADA Manager and several analysis facilities depicted in Fig. 6 and detailed as follows.

*ADA Manager:* is responsible for SLA storage and retrieval from the repository; as well as the translation between several SLA models to a WS–Agreement-based normalised one that ADA is able to analyse.

![Figure 5. Technical Architecture of the extended SALMon](image)

### VIII. EXPLAINING VIOLATIONS WITH SALMONADA

For demonstration purposes, we have implemented a web application\(^2\) as a SALMonADA client in order to specify or upload the WS–Agreement documents to monitor, execute SALMonADA and receive the results. In this web application, we have introduced the WS–Agreements of ADA and SALMon themselves. By monitoring the SLAs of these services, we assess on the one hand, the functionality of SALMonADA, and on the other, the non-functional aspects of its main components. Moreover, as part of the demonstration, we have simulated the consumers that execute ADA and SALMon services. We describe here the asynchronous SALMonADA approach through monitoring the ADA service and analysing the service level fulfillment with the WS–Agreement document in order to report violation explanations.

To monitor the WS–Agreement, the SALMonADA client invokes the startMonitoring method specifying the ADA WS–Agreement and the endpoint for the notification. In the demonstrated scenario, it is the same web application, but any other client can be subscribed to the notification.

\(^2\)SALMonADA web application can be tried at www.isa.us.es/ada.source/SLAnalyzer/ and a screencast is available at gessi.lsi.upc.edu/salmon/ada/
and explaining of the violations of the same ADA WS–Agreement, such as a service reputation agent, service adaptation frameworks, etc. Using this asynchronous approach, as soon as a violation is detected, it is automatically analysed, and then reported to all the subscribed agents. As Fig. 7 depicts, the web application highlights as violation explanation that the AverageResponseTime of the violating metric was measured as 3.421 seconds, while the guarantee term obligates the provider to respond in less than 2 seconds.

The WS–Agreement of ADA, already shown in Fig. 1, includes more guarantee terms to monitor, some of them including SLOs involving more than just one quality metric. Hence, an appropriate explanation identifying not only the guarantee term that are involved in the violation of the SLA, but also the concrete violating metrics, is required. For instance, some operations have a higher priority and are required to be faster than the average response time of the different methods of the service. In this case, our explanation would identify if the violating metric is either the AverageResponseTime or the GeneralResponseTime because a simple identification of the violated term is not enough to grasp the violation cause. Similarly, SALMonADA supports the explanation of violations of more expressive SLOs as follows. The provider may guarantee a different average response time limit for the slower service operations, depending on the general response time of the service: 

\[(\text{GeneralResponseTime} \geq 0 \text{ AND GeneralResponseTime} < 2) \implies (\text{AverageResponseTime} < 3) \text{ AND } (\text{GeneralResponseTime} \geq 2 \text{ AND GeneralResponseTime} < 4) \implies (\text{AverageResponseTime} < 5)\].

IX. Conclusions and Discussion

In this paper we present a conceptual reference model of a monitoring and analysis framework, and SALMonADA as one of its possible instantiations.

The proposed conceptual reference model provides a reference architecture to develop a platform for monitoring SLAs. The proposed architecture ensures and provides the following set of features to implement a concrete platform:

- A flexible and highly decoupled architecture is presented. The different agents of the system deal exactly with the information required in separate documents: SLA, MMD and SLF.
- We propose the MMD as a unique document to manage the monitors, in order to (1) specify the monitoring directives to retrieve the different metrics and (2) report the measured results over the specified metrics.
- We introduce the SLF as the document that explains clearly the violations of the SLA. Other approaches [7], [8] support an event-based violations explanations based on Event Calculus. However, it has in our consideration, the following drawbacks: (1) the client (end-user) must be an expert in EC to specify the guarantee terms with EC formulas, but also to understand the violation explanations; and (2) it is difficult for the client (application or end-user) to grasp the violation origin specially when more than one metric are related in the violating event and/or the violated EC formula.

We present as an instantiation of conceptual reference model, the SALMonada platform. As Section V-B describes, the platform extracts the monitoring information from the client WS–Agreement document as other authors propose [5], [1]. Our proposal differs from these works since we store it in an independent MMD that will be updated with the monitored information when the service is used at service provisioning time. Such updated MMD is used to analyse the service level fulfillment in order to report to the clients an easy-to-understand explanation including the violated SLA terms and its violating metrics. The features that provides our platform can be summarized as follows:

- SBS: It is a SBS by itself, and because of its decoupled structure, the inherent services can be replaced by others if they just implement the required interface.
- WS–Agreement compliant: it is able to analyze expressive SLOs in WS–Agreement documents. Moreover, the clients do not need to be experts in any reasoning paradigm neither EC nor CSP, because we support an assertion language (detailed in Section VII) inside WS–Agreement documents that is more easy to understand.
- Early notification: We provide an easy notification mechanism based on the observer pattern useful for self-adaptive SBS, and other interested parties, such as service reputation agents.

However, the platform presented in [7], [8] have two key points that differs from our proposal and make it very appealing: (1) they consider violations of an expected behavioral of the SBS to monitor by adding assumptions inside the
WS–Agreement document; and (2) as they monitor events with EC, its proposal is able to deal with metrics depending on a time interval. Both are part of the possible improvement of our work, the first in terms of expected operations execution flow (it is possible by defining a precedence order for service operations inside the SLA); and the latter using the temporal analysis of ADA that is currently not considered in the proposal.

ACKNOWLEDGMENT

This work has been partially supported by: S–Cube, the European Network of Excellence in Software Services and Systems; the European Commission (FEDER); the Spanish Government under the CICYT projects SETI (TIN2009–07366) and ProS–Req (TIN2010–19130–C02–01); and by the Andalusian Government under the projects THEOS (TIC–5906) and ISABEL (P07–TIC–2533).

REFERENCES

Preventing Performance Violations of Service Compositions using Assumption-based Run-time Verification

Eric Schmieders and Andreas Metzger
Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen, Essen, Germany
{eric.schmieders, andreas.metzger}@paluno.uni-due.de

Abstract. Service-based Applications (SBAs) will increasingly be deployed in highly distributed and dynamic settings. To a large extent this dynamicity is caused by the trend to increasingly compose SBAs using third-party services. Those services are provided by external organizations and are thus not under the control of the SBA provider. For critical application domains (such as emergency or financial) and important customers (such as key accounts), the SBA developer needs to ensure that each individual SBA instance will live up to its expected requirements even though its constituent, third-party services might fail. To prevent such requirements violations, SBAs should be equipped with monitoring, prediction and adaptation capabilities which are able to foresee and avert menacing violations. Several approaches exploiting preventive adaptations have been presented in the literature, but they rely on the existence of cost models or comprehensive training data that limit their applicability in practice. In this paper we present SPADE, an automated technique that addresses those limitations. Based on assumptions about the SBA’s constituent services (derived from SLAs), SPADE formally verifies the SBA against its requirements during run-time. The experimental evaluation of SPADE, using data collected for six real services, demonstrates its practical applicability in predicting violations of performance requirements.

1 Introduction

Service-orientation is increasingly adopted as a paradigm to build highly dynamic, distributed applications from individual software entities, offered as services. In this paper we refer to such applications as Service-based Applications, or SBAs for short. There is a clear trend that future SBAs will be increasingly composed from third-party services that are accessible over the Internet [15]. As a consequence, SBAs will increasingly depend on the functionality and quality offered by those third parties [5].

If the services that are provided by third parties do not deliver the expected (or contractually agreed) functionality or quality, failures of the running SBA can happen if no countermeasures are taken. As an example, if a service fails
to respond within the contractually agreed 500ms and there is no compensation for this fault, this may lead to a violation of the performance requirements of the overall SBA. Thus, researchers have proposed to equip SBAs with self-adaptation capabilities which enable them to autonomously adapt to the faults of third-party services during run-time [14].

Especially in application scenarios where the provider of an SBA needs to ensure that each running SBA instance provides the guaranteed end-to-end quality properties, there is a strong need to monitor and adapt individual SBA instances. One example are SBAs offered to “key accounts” or “premium” applications, i.e., SBAs for which the provider agreed to pay severe contractual penalties in case of a violation of the promised quality. Another example are applications which might pose financial risks, e.g., caused by fire damage, or endanger human lives. Imagine a fire emergency scenario, in which human beings are endangered if the distributed, service-oriented emergency control system does not react as expected (e.g., does not dispatch ambulances in time).

Current approaches for self-adaptive SBAs follow either reactive or preventive strategies. Reactive strategies propose the execution of monitoring rules (cf. [8] and [16]) to identify deviations during run-time, e.g. using Complex Event Processing. Adaptations are triggered, based on monitored deviations. Monitoring rules cover QoS-properties of a single service, parts of the workflow or an end-to-end requirement. These approaches react on a requirements deviation and are hence not suitable to prevent the deviation itself (cf. further explanations in [6] and [5]). As motivated above, in an emergency situation such a reactive response can lead to critical situations, as it might delay the timely dispatch of operational forces, for example fire engines or ambulances.

To address these problems, researchers have proposed to employ preventive adaptation, which enables SBAs to predict future failures and to perform preventive actions. Although several approaches for preventive adaptation have been presented in the literature (see Section 2), they pose certain limitations, such as the need for cost models or comprehensive training data. In this paper, we aim to address these limitations. Specifically, we will describe and validate the SPADE technique (SSpecification- and Assumption-based DEtection of adaptation needs).

SPADE equips SBAs with adaptation capabilities, empowering them to adapt themselves preventively. To achieve this, SPADE uses run-time verification techniques, execution data of the monitored instances and assumptions concerning the SBAs’ context, derived from Service Level Agreements (SLAs) of third-party services. Together, these mechanisms are used for performance prediction, which is able to detect menacing performance requirements violations of running SBAs. SPADE can thus be used in settings where no cost models or training data are available.

The remainder of this paper is structured as follows. Section 2 provides further insight into the limitations of current approaches for preventive adaptation. In Section 3, we introduce a more complex scenario and example to illustrate the key concepts of SPADE. In Section 4, we describe the SPADE approach in detail. The applicability and effectiveness of SPADE are experimentally evaluated
in Section 5. The experiments were performed on the basis of the example SBA from Section 3 and a monitoring data from six real-world services. Section 6 concludes and provides an outlook on future work.

2 Related Work

Existing approaches related to preventing SLA violations of SBAs can be grouped into two major classes: (1) approaches that aim to predict the SLA violation of an individual SBA instance and to prevent the violation by adapting that SBA instance; (2) approaches that reason about failures of SBAs at the model level (and not at the level of running SBA instances) and which aim to prevent SLA violations of future SBA instances by modifying the specification (model) of the SBA.

**Instance Level prevention (1):** Various approaches which aim at preventing SLA violations of running SBA instances, have been presented in the literature. Ivanovic et al. propose a combination of computational cost analysis and simulations taking the load of allocated resources into account (i.e. invoked services) in order to predict the QoS characteristics of a service composition (see [10] and [9]). This approach has two critical shortcomings. First, the approach requires a cost function offered by the service providers involved in the SBA. This severely impacts on the practical applicability of the approach, as such cost functions are currently not available. Second, the sketched application only has a restricted access to the load of third party services, as the SBA provider can only meter the load produced by its own applications. Thus, the prediction could lack preciseness and reliability. Still, the approach can be considered complementary to SPADE, as, once available, a cost function and assumptions concerning the resource load would provide a good means to refine the present assumption concept used by SPADE.

Several approaches propose machine learning techniques, such as neuronal and Bayesian networks, to predict a menacing SLA violation (e.g. [12], [11], [17], and [13]). The effectiveness of these approaches strongly relies on historical data, which is required as training data. As observed by the authors, several hundreds, or in some cases, even thousands of executions are necessary to ensure the expected precision for the SLA violation prediction. This means that these approaches will exhibit severe limitations as for the applicability if only a small amount of historical data is available. Also, the prediction component usually has to be re-trained after each adaptation of the SBA. Still, those approaches can be employed complementary to SPADE. While SPADE provides reliable results independent of the amount of historical data, it might well be that, as soon as sufficient data is available, the precision of the prediction using machine learning is superior to those of SPADE.

**Model Level prevention (2):** Complementary to the approaches presented above, approaches which aim at preventing SLA violations of potential future SBA instances have been presented in the literature. The approach presented by Ghezzi et al. in [7] suggests to adapt the workflow specification to meet the
SBA’s requirements. The solutions focus on reliability as a quality attribute. To predict the reliability of an SBA the historical data of past SBA instances is fed into a probabilistic model checker. The monitored data of past faulty instances is extrapolated and compared with the reliability values stipulated in the SLA. In case the predicted value violates the SLA, the workflow specification is adapted. As the prediction relies on past faulty instances, the approach cannot be used directly to avoid an SLA violation of an individual instance, which is the aim of approaches of class (1) and SPADE especially.

3 Example SBA

To illustrate and evaluate SPADE in the remainder of this paper, we will use an abstract example of a service-based application, which is depicted in Fig. 1. In the middle of the figure, the workflow is depicted as an activity diagram. On the right hand side of the figure, the third-party services that are invoked by the SBA (i.e., by the actions of the workflow) are shown. Finally, on the left hand side, the service response times for one actual execution of the SBA are given.

![Fig. 1. Example SBA](image-url)
4 The SPADE Approach

In this section we present the SPADE approach. For that reason, we structure the design time and run-time activities along the phases of the SBA life-cycle model as elaborated by the S-Cube Network of Excellence (see Fig. 2). To illustrate the approach, each life-cycle description comprises an example paragraph explaining SPADE by means of the example SBA introduced in Section 3.

![Fig. 2. S-Cube SBA Life-Cycle Model [14]](image)

**Requirements Engineering:** In the requirements engineering phase, the functional and quality requirements for the SBA are elicited and documented.

In order to assess automatically whether the application deviates from its requirements during operation, functional and non-functional requirements are formally expressed as input to SPADE. We propose to already perform this formalization during the requirements engineering phase, as this facilitates an early validation of the requirements, e.g., by means of formal consistency checks (cf. [1]), and hence reduces the risk of expensive corrections in later phases.

To formalize the SBA requirements, SPADE uses the specification language ALBERT (cf. [2]). We choose ALBERT because of its capability to express logical and temporal dependencies of monitoring data along an executed path.

**Example:** For the example SBA from Section 3, we formalize the required response time $r_{per} \epsilon \textit{RASC}$, which demands an end-to-end response time of at most 55 seconds. In ALBERT $r_{per}$ can be formalized as $r_{per} := \text{onEvent}(\text{start},"\text{Action1}") \rightarrow \text{Within}(\text{onEvent}(\text{end},"\text{Action7}"), 55000)$. The $\text{onEvent}$ operator evaluates to true if the activity specified in its second argument performs the state change denoted in its first argument. The $\text{Within}$ operator evaluates to true if its first argument evaluates to true within the amount of milliseconds specified in its second argument.

**Design:** During the design phase, the activities and the control flow of the application are specified (e.g. in BPEL or YAWL). Together with the definition of the workflow, candidate services are identified that can provide the functionality and quality to fulfill the requirements of the SBA [4].
Following the same reasoning as in the requirements engineering phase, we suggest to formalize the workflow during the design phase already in order to reduce the risk of later corrections. We extend the idea presented by Bianculli et al. [2] and propose using the BOGOR model checker during run-time. ALBERT expressions can be executed by BOGOR. We formalize the workflow using BIR, the BOGOR Input Representation.

Example: In order to use the BOGOR Model Checker, we specify the workflow of the example SBA by using BIR. The resulting specification $S_{ASC}$ can then be executed and analyzed by BOGOR.

Realization: To achieve the desired end-to-end quality of SBAs, contracts between service providers and service consumers on quality aspects of services are established. Following [4], this means that for each candidate service, the best quality level for the available budget is negotiated and stipulated in an SLA. In SPADE, the quality levels that have been negotiated and agreed upon with the service providers are formalized. We treat quality levels as assumptions ($A$) about the SBA’s context. We formalize $A$ using ALBERT again.

During design time, a service repository must be established, e.g. based on UDDI. This serves as a pool for alternative services. These services can be converted during run-time.

Example: For our example we use the assumed response time given in the example SBA (see Fig. 1). The assumption for the $a_{FastWeather}$ service bound to Action 1 is formalized as follows: $a_{FastWeather} := \text{onEvent}(\text{start, ”Action 1”}) \rightarrow \text{Within}(\text{onEvent}(\text{end, ”Action 1”}), 5900)$

Deployment: The deployment and provisioning phase comprises all the activities needed to make the SBA available to its users. During the deployment phase, SPADE uses BOGOR to check whether the workflow specification ($S$), under the given assumptions ($A$), satisfies the requirements ($R$), i.e. whether $S, A \models R$. In case the requirements are not satisfied, the phases of the evolution loop (cf. Fig. 2) are re-executed, e.g. in order to bind faster services. If the SBA is successfully verified, the SBA is deployed on the run-time environment. In the case of SPADE, we use the GlassFish application server.

Example: The specification of the abstract service composition evaluates to true, i.e. $S_{ASC}$ and $A_{ASC}$ satisfy $R_{ASC}$. Thus, the SBA is deployed.

Operation and Management: This phase comprises the execution of the SBA and the monitoring of its constituent services using service monitoring techniques.

To identify assumption violations during the operation of the SBA, monitoring mechanisms are employed. SPADE uses the monitoring facilities of the GlassFish application server to check, whether the monitoring data $m_i \in M$ of service $i$ satisfies the related formalized assumptions $a_i \in A$, i.e. whether $m_i \models a_i$.

---

1 The complete set of formalizations is available from http://www.s-cube-network.eu/SPADE
2 http://glassfish.java.net/
Example: During the execution of the abstract service composition let us assume that the monitoring data \( m_{\text{FastWeather}} \) and \( m_{\text{Google}} \) satisfy their related assumptions, leading to an cumulative run-time of 10055 ms. Let us further assume that the subsequent service \( \text{WSAmazonBox} \) responds too late. Instead of the assumed 8400 milliseconds, the invocation lasts 23083 ms, i.e. \( m_{\text{Google}} \not= a_{\text{Google}} \). Due to this deviation, the current instance is suspended and the next phase is entered immediately, as a performance violation could be indicated.

**Identify Adaptation Needs:** If an assumption violation has been identified, SPADE checks whether the requirements are still satisfied. For all services that have been invoked, SPADE replaces the assumption values by concrete monitoring data. This means that only for the services not invoked, the assumptions are used during run-time verification. SPADE thus uses a subset \( A' \subseteq A \) together with the set of monitored data \( M \) to check whether \( S, M, A' \models R \).

Similar to the deployment phase, SPADE utilizes BOGOR to perform this verification during run-time. If the check reveals that \( R \) is still satisfied, the workflow execution is continued. Otherwise, an adaptation must be triggered.

Example: To determine whether the requirement in the example is still met, the workflow specification (\( S_{\text{ASC}} \)), the monitoring data (\( m_{\text{FastWeather}} \), \( m_{\text{Google}} \) and \( m_{\text{WSAmazonBox}} \)) together with the assumptions of the to be invoked services (\( a_{\text{HylinkExtractor}}, a_{\text{CurrencyConverter}}, a_{\text{GetJoke}} \)) are checked against the requirement \( r_{\text{per}} \). The predicted end-to-end duration is 56238 ms, which exceeds the 55 seconds demanded by \( r_{\text{per}} \). In consequence, the workflow actually have to be adapted, as otherwise a performance violation seems to be in all probability.

**Identify Adaptation Strategy:** Subsequent to an identified adaptation need, the next step is to create an appropriate adaptation strategy such that the menacing SLA violation can successfully be averted.

SPADE is equipped with service substitution capabilities, as this is one of the few adaptation techniques that can be used to compensate for lost time. Furthermore, SPADE exploits the CHOCO constraint solver\(^3\) to determine which not-yet-invoked services have to be substituted. We consider the workflow as a graph \( G = (V, E) \), composed from a set of vertices \( V \), representing the workflow actions, and a set of edges \( E \), representing the transitions between these actions. We consider \( p = (v_1, \ldots, v_i) \) as path in \( G \). Path \( p \) includes all v, which are executed until the occued deviation, i.e. \( p = \{(v_n, \ldots, v_m) | v \in V \land \text{invoked}(v)\} \). \( G' \) is a subgraph of \( G \) containing all \( v_1 \ldots v_n \) which are not yet executed, i.e. \( G' = \{(v_n, \ldots, v_m) | v \in V \land \neg\text{invoked}(v)\} \).

We now define a set of paths \( P' = \{p' \in G'\} \) to formalize our constraint as \( p_{\text{per}} + p'_{\text{per}} \leq r_{\text{per}} \), where \( p_{\text{per}} \) is the already consumed end-to-end execution time, \( p'_{\text{per}} \) is the cumulative response time of a possible workflow path in \( G' \) and \( r_{\text{per}} \) is the performance requirement of the monitored service composition. CHOCO has to solve this constraint with respect to the response times of the available alternative services (available from the service repository) for each \( p' \in P' \). CHOCO also has to consider the remaining possible paths of the workflow. Right after the invocation of the deviating service we do not know which path will be

\(^3\) [http://www.enm.fr/z-info/choco-solver/](http://www.enm.fr/z-info/choco-solver/)
followed during the further execution of the workflow instance. Thus, we scrutinize a subset of $P'$, i.e. $P''$. $P''$ contains paths for which performance violations are predicted. The constraint solver calculates a combination of services invoked along each $p'' \in P''$, based on the response time of alternative services. The necessary adaptation actions are easily derived from the results, as each identified combination exposes the services, which need to be substituted as well as their substitution. The required substitutions are summarized (where double entries are avoided) in the adaptation strategy which is propagated to the mechanisms of the next phase.

Example: In our abstract service composition example we are facing two paths in $G'_{ASC}$, i.e. $p_1 = (\text{Action}5, \text{Action}7)$ and $p_2 = (\text{Action}5, \text{Action}6, \text{Action}7)$. Both paths will violate $r_{per}$ and therefore have to be adapted. Based on the output of the constraint solver, services for $\text{Action}5$ and $\text{Action}6$ are chosen to be substituted by faster services, thus satisfying $r_{per}$.

Enact Adaptation: During this last adaptation phase the adaptation strategy is executed. For this purpose, the instructions comprised in the adaptation strategy are dispatched. The dispatching usually utilizes the facilities provided by the chosen run-time environment or involves additional adaptation mechanisms.

In SPADE we use the interception-mechanisms provided by the GlassFish application server. We exploit the built-in Enterprise Service Bus to manipulate the target of a message. These messages are generated by the workflow engine, when a service has to be invoked. We switch the target service of the invocation to the service identified during the previous phase.

Example: In our example, the two service invocations $\text{Action}5$ and $\text{Action}6$ are redirected to the substituting services. For this purpose the message routing table comprising the message destinations is manipulated. Consequently, the execution of the instance is resumed.

5 Experimental Evaluation

This section presents the experiments we performed to assess the efficiency of SPADE in detecting adaptation needs. Experimental validations of comprehensive approaches like SPADE must be accurate and comprehensive as well. To reduce this complexity, we focus on SPADE’s ability in identifying adaptation needs in our first set of experiments.

Our measurement of SPADE’s efficiency is twofold. First, we examined unnecessary adaptations, i.e. false positives, as such adaptations could lead to avoidable costs, e.g., when replacing a free service with a commercial service to compensate for faults. Secondly, we count the amount of situations in which SPADE cannot perform an adaptation. It can happen that a service invocation leads to a violation of an SLA, such that the end-to-end requirement is already violated. In those situations, the SBA instance obviously cannot be adapted preventively in order to avert that requirements violation, as the requirement has
already been violated. Both values are expected to be low, as a low value implies a high number of cases where SPADE was successfully applied.

**Experimental Design:** The performed experiments are based on a simulated execution of the example SBA’s and its services. This enables the reproducibility of the test results for the exact same “input” data, thus allowing other researchers to reproduce the performed tests. We simulate the example SBA, introduced in Section 3. Specifically, we simulate the execution of the workflow and retrieve the response times for its constituent services from a large set of monitoring data [3]. This dataset comprises real monitoring data, which was crucial for our experimental design. By using realistic monitoring data, we show the applicability of SPADE in realistic settings. The number of SBA instances that can be experimentally assessed is limited by the size of the used dataset. To each service invocation within an SBA instance we assign one single data point from the dataset of the respective service. In the example SBA, this allows for the execution of 5884 different SBA instances.

**Determining the Degree of False Positives:** In our experiments, one single SBA instance is represented by one path through the example SBA together with the concrete monitoring data. As part of the simulation, we calculate the end-to-end response time for each SBA instance execution by adding the monitored response times of the invoked services along the SBA instance’s path. Based on this calculated SBA instance response time, we can determine false positive adaptation triggers. Once SPADE has triggered an adaptation, it is checked whether the calculated SBA instance response time violates the SBAs end-to-end requirements. For example, the check indicate that the assumed instance response time exceeds the upper bound of \( r_{per} \leq 55 \) (seconds) as in our example, thus an adaptation is triggered. However, if the calculated instance response time reveals that the requirement would not have been violated in case the SBA execution continues without adaptation, we consider this workflow instance as a false positive. To take the duration of the three phases into account, in which SPADE suspends the execution of the workflow instance, we measured this duration (i.e. ca. 170 ms) and added it to the calculated SBA instance response time.\(^5\)

**Determining the Degree of Impossible Adaptations:** There were situations in which service invocations deviated from their stipulated response time, such that the performance requirement is violated. In these situations, it is too late to apply SPADE as the performance requirement is already violated. To determine the percentage of these situations we put the number of the service

\(^4\) Please note that in the example SBA, the Google service is bound twice. There are 2943 data points in the Google dataset. In order to provide data for each binding, we split the dataset in half and assigned an interleaved subset to each of the two service invocations. Each subset comprises 1471 datasets. As the workflow of the example SBA allows four different paths, this leads to a total number of 5884 SBA instances.

\(^5\) The measuring has been carried out on an Intel Core i5-760 platform with a 2.80 GHz CPU and 4 GB RAM.
invocations which lead to these SLA violations in relation to the amount of all service invocations.

**Data Analysis and Interpretation:** The SPADE approach has been applied to 5884 SBA instances (cf. row \((a)\) in Table 1). 629 of those SBA instances have been executed without any assumption violations \((b)\). During the execution of the complementary 5255 SBA instances assumptions have been violated \((c)\). This number is explained in the discussion on threats to validity. For those SBA instances, SPADE has identified 604 preventive adaptation triggers \((d)\).\(^6\)

<table>
<thead>
<tr>
<th>Description</th>
<th>%</th>
<th>SPADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBA Instances Executed ((a))</td>
<td></td>
<td>5884</td>
</tr>
<tr>
<td>SBA Instances without Assumption Violation ((b))</td>
<td>b/a</td>
<td>629 10.7%</td>
</tr>
<tr>
<td>SBA Instances with Assumption Violation ((c))</td>
<td>c/a</td>
<td>5255 89.3%</td>
</tr>
<tr>
<td>SBA Instances with Adaptations ((d))</td>
<td>d/a</td>
<td>604 10.3%</td>
</tr>
<tr>
<td>False Positive Adaptations ((e))</td>
<td>e/a</td>
<td>72 1.2%</td>
</tr>
</tbody>
</table>

**Table 1. False Positives in Relation to Executed SBA Instances**

72 of the adaptation triggers were false positives \((e)\). Thus, 1.2% of the workflow instances would have been unnecessarily adapted. With respect to the challenging time constraint of 55 seconds, motivated by the introductory emergency scenario, we consider the percentage of false positives extremely promising, especially, as an unnecessary adaptation does not imply an SLA violation. Furthermore, the amount of situations in which SPADE cannot adapt is very low as well. As depicted in row \((c)\) of Table 2, a total of 825 out of 28624 service invocations, i.e. 2.5%, lead to situations where an adaptation is not possible. Nevertheless, this could still mean a threat in an emergency setting. We will discuss this shortcoming in our future work section (cf. Section 6).

<table>
<thead>
<tr>
<th>Description</th>
<th>%</th>
<th>SPADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Invocations in Executed SBA Instances ((a))</td>
<td>32362</td>
<td></td>
</tr>
<tr>
<td>Actual Service Invocations until Adaptation Trigger ((b))</td>
<td>b/a 30784 86.1%</td>
<td></td>
</tr>
<tr>
<td>Assumption Viol., where Adaptation is not Possible ((c))</td>
<td>c/b 825 2.5%</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2. Invocations leading to Performance Violation**

**Discussing Threats to Validity:** First exploratory checks indicate that the efficiency of SPADE depends on the concrete values for assumptions and requirements. This can pose a threat to internal validity, as it might be the case

\(^6\) Please note that only those situations have been counted as triggers, in which adaptations were still possible (cf. below and Table 2).
that the values used in the example SBA are not realistic. We address this issue by referring to failure rates of service invocations observed in a case study, in which 150 different services have been examined [18]. In our experiments we approximate the observed service failure rate of 95% by adjusting the monitoring rules (which use the assumption values) accordingly, thus aligning the experimental design to the observed, realistic values. In order to learn more about the effect of assumptions and requirements values on SPADE’s efficiency, we are planning several series of experiments during which we vary both values.

The example SBA does not cover all possible control constructs available to build SBAs (such as loops and forks), which might pose threats to external validity. We thus will extend our SPADE prototype with a full-fledged model checker to handle more complex workflows and will perform further experiments based on this update to improve the applicability of the SPADE approach.

6 Conclusions and Future work

This paper described the SPADE approach, which is an automated technique to determine adaptation needs to trigger preventive adaptations of SBAs. As SPADE does not rely on historical data, SPADE overcomes some of the shortcomings of the existing solutions. The applicability of SPADE has been supported by experimental results, using monitoring data of real services.

We plan to continue our work on preventive adaptation in two directions. First, we will combine SPADE with our PROSA approach. The PROSA approach is capable of predicting quality violations of individual services. The combined approach is expected to act in situations in which SPADE is not able to prevent requirements violations as intended.

Secondly, we plan to apply SPADE in a cross-layer adaptation setting. In this setting, SPADE is expected to exploit the adaptation mechanisms of two different layers: the service composition and the service infrastructure layer. We expect that harmonizing the adaptation on both layers will increase the number of situations in which SPADE is able to compensate for deviations, which thus may increase SPADE’s success in avoiding requirements violations.

Acknowledgments: We cordially thank Benedikt Liegener for contributing to the set up of the experiments, as well as Marian Benner, Mark Rzepka and Osama Sammodi for helpful comments on earlier drafts of the paper. The research leading to these results has received funding from the European Community’s 7th Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

Future Internet Apps: The Next Wave of Adaptive Service-oriented Systems?

Andreas Metzger¹ and Clarissa Cassales Marquezan¹

Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen, Essen, Germany
{andreas.metzger, clarissa.marquezan}@paluno.uni-due.de

Abstract. The Future Internet will emerge through the convergence of software services, things, content, and communication networks. Service-orientation is expected to play a key role as enabling technology that allows the provisioning of hard- and software entities and contents as services. The dynamic composition of such services will enable the creation of service-oriented systems in the Future Internet (FI Apps), which will be increasingly provided by third parties. Together with increased expectations from end-users for personalization and customization, FI Apps will thus face an unprecedented level of change and dynamism. Based on our understanding of adaptive service-oriented systems, this paper discusses the key adaptation characteristics for FI Apps (illustrated by a concrete application domain). The importance of each of those characteristics has been empirically assessed by means of a survey study. We provide the results of this study which can help in better understanding where future research and development effort should be invested.

1 Introduction

The Future Internet (FI) is expected to become a ubiquitous infrastructure that will overcome the current limitations of the Internet for what concerns interoperability, accessibility, resource efficiency, security, and trustworthiness, as well as integration with the physical world [9]. Specifically, the Future Internet will be built around four major areas: The IoS (Internet of Services) that relates to the provision, discovery, composition, and consumption of (software) services via the Internet. The IoT (Internet of Things) relates to embedded systems, sensors and actuators that collect and carry information about real-world objects via their interconnection with the Internet. The IoC (Internet of Content) relates to the discovery, distribution, combination, and consumption of all kinds of media objects (e.g., text, 3D graphics and audio) which carry meta-information about their content and can be accessed via the Internet. Finally, NoF (Networks of the Future) relates to ubiquitous communication facilities (e.g., mobile, broadband).

In the Future Internet setting, services are expected to play a key role as enabling technology and core building entities. These services will provide the right level of abstraction from hard- and software entities, extending from business functions to data storage, processing and networking, devices and content.
Service-oriented systems in the Future Internet, we call them **FI Apps**, will be dynamically composed from these service-based entities, operating on federated, open and trusted platforms exploiting the Future Internet areas.

The capabilities and features of FI Apps will be increasingly provided and “owned” by third parties. Examples include Internet-based software services, public sensor networks, and cloud infrastructures. Due to this “shared ownership” [1]), FI Apps will face an unprecedented level of change and dynamism. Further, expectations from end-users for what concerns the personalization and customization of those FI Apps are expected to become increasingly relevant for market success. For instance, a FI App should be able to adapt depending on the usage setting (e.g., office vs. home) or based on the available communication infrastructure (e.g., sensors vs. WiMAX). It will thus become increasingly important to engineer FI Apps in such a way that those applications can dynamically and autonomously respond to changes in the provisioning of services, availability of things and contents, as well as changes of network connectivity and end-user devices, together with changes in user expectations. Ultimately, this means that *adaptation* will become a key capability of FI Apps.

There has been significant progress for what concerns principles and techniques for building adaptive service-oriented systems. For example, many solutions have been developed within S-Cube\(^1\). However, if we consider the Future Internet setting, those solutions will need to be significantly augmented, improved and integrated with a complete systems perspective. Specifically, this requires significant progress towards novel strategies and techniques for adaptation, addressing key characteristics of adaptive FI Apps. To enable the next wave of adaptive service-oriented systems in the Future Internet, it will thus be critical to understand the importance of the various adaptation characteristics to specifically target research and development activities.

This paper, therefore, identifies and analyses key characteristics of adaptive FI Apps (Section 3). Those characteristics are illustrated with examples from a scenario of the application domain of transport and logistics (Section 2). The paper scrutinizes the relevance of those different characteristics through an empirical study (Section 4). As a research method we have employed an exploratory survey study, involving 51 respondents from the Future Internet community.

## 2 The Transport & Logistics Scenario

Modern transport and logistics processes are characterized by distributed inter-organization activities often spanning several countries and continents. An illustration of current transport and logistics process associated with the construction of an offshore wind energy plant is presented in Figure 1. Based on this example, we describe some of the limitations of current processes and present insights on how Future Internet can contribute to overcome those limitations.

Figure 1(a) shows the stages required currently to accomplish the construction of a wind energy engine. The individual components are produced by various suppliers that typically are geographically distributed and in the first stage they are delivered to the manufacturer (i.e., the system integrator) of the wind energy station who is responsible for storing these components. To ensure that all resources and semi-finished goods are available on time at the production line, this structure of the supply chain requires a considerable amount of buffer and warehousing space at the sites of each supplier as well as at the system integrator. This inefficient and highly cost intensive process results from the fact that current logistic processes lack the required end-to-end visibility of the supply chain, which itself results from insufficient integration between IT systems of the various logistics stakeholders.

After component production and supply management, the complete wind energy station is constructed in a trial assembly for a full operational test (stage 2). This is required to ensure defect-free operation as expected by the end-customer, and can avoid any costly and time-consuming delivery requirements for missing or defective parts when assembly occurs at the final destination. From the business perspective, the trial assembly is necessary because the information provided throughout the supply chain is often incompatible, so that operational reliability cannot be guaranteed by merely inspecting the documentation of the delivered components. After the trial assembly and a final operational test, the plant is disassembled and shipped to the intended destination (stage 3).

This example shows that current transport and logistics processes face obstacles that prevent the achievement of a more reliable, lower cost and environment-
tally friendly industry. Solutions for these obstacles must be characterized by: integration of different ICT environments, better integration among systems and the real world, reduction of manual intervention and guarantees of ubiquitous access to information among the partners. The Future Internet is an alternative that can encompass all these characteristics. Indeed, the employment of Future Internet areas, as illustrated in Figure 1(b) can enable the end-to-end visibility of the supply chain, where information derived from the real world (e.g., data retrieved from sensors or cameras in the vessel) can be ubiquitously accessed by all the partners of the chain (using for this any kind of network connectivity) and the service based applications of those partners are able to exchange information, negotiate and collaborate in order to accomplish the business goals of the supply chain. The environment aforementioned constitute, indeed, the so called Future Internet application. In such heterogeneous environment, dynamic and unexpected changes can happen and, thus, adaptive characteristics become of great importance for Future Internet applications.

3 Adaptive Characteristics of FI Apps

In this section, we elaborate key characteristics for adaptive FI Apps. Those characteristics have been identified jointly by S-Cube, the EU Network of Excellence on Software Services and Systems [7], and FInest (the EU FI PPP Use case project on transport & logistics). Figure 2(a) shows the projects’ shared understanding of the the layers of the Future Internet. This will be used to explain the different adaptation characteristics and their relationships that are presented in Figure 2(b) and described further below.

The Future Internet Platform layer (in Figure 2(a)) constitutes generic technology building blocks from various Future Internet areas (IoS, IoC, IoT and...
Those building blocks can be instantiated to platform instances on top of which FI Apps can be executed. The Future Internet Application layer constitutes domain-specific technology building blocks, as well as domain-specific FI Apps. As illustrated in Figure 2(a), possible application domains are transport & logistics, eHealth and media. The Socio-Economic layer constitutes networks of people and organizations, which can benefit from the FI Apps.

3.1 Cross-Layer Adaptation

The proposals in the research literature highlighted fundamental aspects of cross-layer adaptation in service-oriented systems [7, 1]. These proposals considered the interaction among the SOA layers (business process management, service composition and service infrastructure). For FI Apps, however, the cross-layer adaptation is expected to gain a much broader meaning. An example of how cross-layer adaptation might have to be reconsidered can be derived from the scenario presented in Section 2. Consider that sensors are used to monitor the conditions of the wind engine parts transported by the vessel (Figure 1(b)) and this information is used by the supply logistic chain to organize and synchronize the collaboration among the parties (all suppliers, warehousing and transport providers). Now, if the vessel informs that goods inside of the containers have been damaged, this information must be escalated up to the Socio-Economic Layer so that the parties can adapt their applications and business values and networks in response to a change in the FI Platform layer. Therefore, future service-oriented systems need to expand on the concept cross-layer adaptation.

3.2 Cross-Area Adaptation

As motivated in the introduction to this paper, the Future Internet will enable end-users to experience an ubiquitous and transparent access to applications, services and information using any kind of device at any time. One important consequence of this vision is thus the anticipated need for FI Apps to adapt not only considering the changes and resources in one specific area (e.g., service availability in the IoS), but also regarding other areas (e.g., adapting according to sensor information availability in the IoT). In our running example, for instance, if the sensors of the vessel stop sending information for the transport & logistics application, it is necessary to start adaptive actions to gather information about the status of the containers in the vessel from other sources. One possible solution would be to exploit the capabilities of the IoC, e.g., by processing the video streams from the surveillance cameras of the vessel. Cross-area characteristics of adaptation, thus, can lead to many new research questions. One of such questions, is related to understand the limits and the interactions between the adaptive capabilities in each of the FI areas.

3.3 Distributed Adaptation

Most of current solutions for adaptation in the scope of service-oriented application (i.e., based on third-party services) typically employ a central entity for
gathering information from distributed sources and taking the decisions. This leads to at least two critical problems. First, in a third-party scenario, the central entity will only have limited access to information from the third-party service providers (e.g., the load of the computing infrastructure at the providers’ side is unknown to the service user). Second, a central entity will present classical problems like single point of failure, lack of scalability, and bottlenecks which will not be acceptable once the scale of the Future Internet. In our view, distribution is not only about retrieving information from distributed entities, but it is also about the distribution of the actual decision-making process. This need is expected to be exacerbated if we also take into account the relevance of cross-layer and cross-area characteristics in FI Apps. In our example, for instance, once the vessel informed to the collaborative supply chain (Figure 1(b)) that a container is damaged and the parts of the wind engine are compromised, the service-oriented application of each partner in the supply chain has to adapt, for instance, by re-negotiating contracts (e.g., delaying the actual physical transport and requesting more warehouse space). In this case, the service-oriented applications could distribute and negotiate the decision of which parts of the supply chain should adapt because in such a distributed business network it will not possible to collect complete information from all the parties (e.g., due to privacy or IP concerns). Thus, distributed decision-making would help to avoid unilateral and isolated adaptations in the supply chain.

3.4 Context-aware Adaptation

The information about the context in which service-based systems are executed as well as their users impact on the expected behaviour and quality of the systems [7]. For the Future Internet, not only information about the users’ context will impact on the applications, but also other context aspects from the FI areas related to the application will gain importance; e.g., information about the environment (from IoT) or about the geographic region (from IoC). For example, if the service-oriented applications of the supply chain of the wind energy plant do not receive the expected report from the vessel, they cannot assume that adaptation actions are needed because no status of the wind engine parts was received. It is necessary to check the context of the vessel. For instance, because of a very strong storm, the radio and satellite communications might have been interrupted. One possible manner to become aware of such a context change would be through the combination of information from GPS and weather forecasts. Another context-aware adaption issue in the transport and logistics scenario is associated with country-specific regulations. In this case, if some change occurs and the freight needs to be delivered through another country, the documents and delivery processes need to be adapted to reflect that country’s regulations.

3.5 Autonomic and Human-in-the-loop Adaptation

One important perspective of Future Internet applications will be the involvement of different user roles in the adaptation activities. Although many adap-
tation decision can be performed autonomously (e.g., communication networks and computing infrastructures are furnished with self-* capabilities), there will be a certain limit for such an automation. Specifically, once adaptation decisions can impact on the business strategy and success of an organization, or it involves creative decision making, humans need to be involved in the adaptation process. This means that there will be a range of adaptation strategies from completely automated (autonomous) to interactive and manual (human-in-the-loop), as illustrated in Figure 2(b). In the wind energy scenario, for example, when the vessel informs the application that parts of wind engine are damaged, there might be the option to automatically contract an alternative provider for those parts. However, a more effective decision would have been to change the assembly order of the parts, thereby saving the costs incurred in starting a new procurement process with an alternative provider. When designing adaptive FI Apps, we thus need to find a proper balance between these two characteristics.

### 3.6 Reactive and Proactive Adaptation

The adaptation of service-oriented applications mostly occurs reactively, i.e., the application is re-configured due to changes in the context or due to faults of third-party services [7]. In contrast to reactive adaptation, preventive and proactive adaptation offer significant benefits. Preventive adaptation is associated with the case in which an actual local failure is repaired before its consequences become visible to the end-user, while proactive adaptation predicts future failures and changes, and modifies the application before those changes impact\(^2\). For instance, if a problem is discovered before it impacts on the actual application, no compensation or roll-back actions have to be performed (cf. [5, 8]). For the FI Apps proactive adaptation is expected to gain further importance, mainly because of the heterogeneity of elements involved in the design of such applications and the intensification of the inter-organizational dependencies. Thus, a local fault can lead to a chain of reactive adaptations across organizations. For example, in the construction of the wind energy plant, the transport and logistics application must be very well synchronized in order to avoid extra costs with transport and warehousing in case of delays. A proactive supply chain application for this scenario (as illustrated in Figure 1), could predict – e.g., based on weather forecasts – if deviations along some of the transport routes might occur (e.g., when severe thunderstorms impact on air-traffic situations). If such a deviation is predicted, the FI App could either modify the transport processes by using alternative transport modes (such as trains), or schedule further warehouse capacity.

### 4 Survey Study

Currently, there are few empirical studies for what concerns assessing the importance of characteristics of FI Apps. The NESSI membership survey (published in May, 2011) is one of those. That survey has identified “adaptable” and

\(^2\) In this paper, preventive adaptation is subsumed in proactive adaptation.
“self-manageable” as being among the top 7 characteristics. However, no further details on what kind of “self-management” or “adaptiveness” would be expected has been provided. Therefore, to gain insight into the practical importance of the various characteristics of adaptive FI Apps (as the ones identified in Section 3), we have performed an exploratory survey study involving practitioners and researchers involved in Future Internet activities. This section reports on the design and the results of this survey study, for which we follow the recommendations for empirical research by Kitchenham et al. [2].

4.1 Context

The survey study was performed by involving the participants of the 6th Future Internet Assembly, which was held in Budapest from 17-19 May, 2011. The Future Internet Assembly serves as a forum, where European and international practitioners, academics and policy makers come together to discuss emerging issues in the Future Internet. More specifically, the survey study was carried out as part of a dedicated session on “Adaptive Future Internet Applications”, co-organized by the authors. We consider the FIA event as an ideal setting for our survey, as its participants are representative for the FI community.

The research question that we aimed to answer with this survey study were associated with the importance of the different characteristics of FI Apps. In addition, we wanted to explore the relative importance of adaptation for the various layers, areas and application domains that we see emerging in the Future Internet (see Section 4.3 for details). We believe that the outcomes of the survey can give insights that contribute to a better understanding of the practical relevance of the issues addressed in this empirical study and ultimately can lead to more targeted research activities (as analyzed in Section 3).

4.2 Design, Execution and Data Collection

Concerning the design of our survey questionnaire, we specifically took into account the following findings from psychology as reported by Krosnick [3].

(i) Based on the observation that ranking can be very time consuming and that people enjoy ratings more than rankings, we asked participants to rate the various characteristics. We decided to determine the ranking based on the rates by applying a simple weighing function to the responses.

(ii) Concerning the answer choices, we did not offer a “no opinion” choice. This was based on the observation that offering a “no opinion” choice, can compromise data quality. Further, we used verbal scale labels, as numeric labels might convey the wrong meaning. Following a recent survey study presented by Narasimhan and Nichols [6], we used their five-point scale employing the following labels: unimportant, of little importance, somewhat important, important, very important. Finally, we started with the

negative options first, as studies have shown that people tend to select the first option that fits within their range of opinion. By doing so, our study will thus lead to more “conservative” results.

The questionnaire has been pretested by 10 researchers from our institute. This pretest lead to significant improvements for what concerns the understandability of questions and options. Specifically, questions that have been perceived as difficult to understand have been augmented with examples. In addition, the context in which the survey was carried out (the session at FIA) started with a short introductory presentation that explained the key terminology. Further, three different application domains have been presented to further illustrate adaptive FI Apps. Those included media, eHealth, as well as transport and logistics (the last one being described in Section 2 of this paper).

Each of the participants has been handed a survey questionnaire (see Section 4.2) during the course of the FIA session. Altogether we received responses from 19 of the 51 of the registered participants of the session\(^4\), i.e., we achieved a response rate of 38%. As we have distributed the questionnaire only to the participants of the FIA session on “adaptive services”, we can assume a general interest and knowledge about the topic and thus can expect good data quality. Further, as Table 1 shows the respondents demographic distribution closely matches that of the population, which implies that respondents constitute a representative sample.

\begin{table}
\centering
\begin{tabular}{lll|l}
\hline
Population (% of TOTAL) & Academia & Industry & Other & TOTAL \\
\hline
Registered Participants & 26 (51\%) & 24 (47\%) & 1 (2\%) & 51 \\
Survey Respondents & 10 (53\%) & 8 (42\%) & 1 (5\%) & 19 \\
\hline
\end{tabular}
\end{table}

4.3 Analysis, Findings and Validity Threats

The results of our survey are presented along the four questions that we asked the participants:

Q.1 How important are the following adaptation characteristics for Future Internet applications?
Q.2 How important are adaptation capabilities within the following Future Internet areas?
Q.3 How important are adaptation capabilities on the following layers?
Q.4 How important are adaptive FI applications for the following application domains?

The adaptation characteristics and the rating for each one of them are illustrated in Figure 3. The three most important characteristics are context-aware,
human-in-the-loop and cross-layer, which indicates the need to better understand the role of users for what concerns adaptation of FI Apps. Usage settings constitute a relevant context factor, human-in-the-loop adaptation relies on the critical decision making capabilities of humans, cross-layer adaptation involves the business and socio-economic layers, where human interaction starts to play an important role. Reactive and proactive adaptation capabilities have been rated roughly similarly, indicating that both characteristics need to be considered for FI Apps. Interestingly, distributed and cross-area adaptation have been ranked least important. Although the recent research literature points out some proposals towards distribution in service-oriented systems [4], it seems that this is not perceived yet as an important feature. We perceive the low rating of cross-area adaptation as a consequence of the fact that each one of the FI areas are not consolidated and well established (in terms of their specification foundations). Therefore, it is comprehensible that the need for this type of cross-area adaptation is not perceived as important.

![Fig. 3. Replies to Question Q1: Relevant Adaptation Characteristics](image)

The left side of the Figure 4 illustrates the importance rating of question “Q2”. Adaptation has been deemed to be most relevant for software services (IoS) and for security, privacy and trust (SPT) aspects, followed right behind by communication networks (NoF). Interestingly, adaptation in the IoT has been deemed least important, which might be attributed to the fact that so far, the IoT has been perceived a rather static entity, only now becoming increasingly dynamic (e.g., due to the increased use of nomadic devices).

The observations associated with question “Q3” are presented in the right side of Figure 4. The communication infrastructure layer has been deemed most important, followed by the application and the business layers. Although the high rank of the communication layer might contradict the findings that the IoS is the most important area, one needs to understand that the adaptation of services can be strongly impacted by network connectivity. Interestingly, adaptation at the computing infrastructure layer (such as cloud) ranks as less important, although one would expect that with increasing demands for cloud computing (such as provisioning of SaaS for multiple tenants), adaptation should be more relevant.

Figure 5 shows the results related to question “Q4”. In this survey study, eHealth with quite some difference from other domains, has been deemed the
application domain where adaptive capabilities will be most important. Smart cities and utilities, as well as mobility, transport and logistics follow right behind.

The analysis of validity threats to our survey is also an important issue to be discussed. We believe that the construct validity has been addressed by the carefully design of the survey questionnaire as discussed in Section 4.2. The questionnaire itself is accessible at http://www.s-cube-network.eu/fia. Another issue is the internal validity. In this survey study, one should be careful in interpreting the results for question Q.4, as three application domains, viz., eHealth, transport & logistics and media have been elaborated during the FIA session, thus possibly leading to biased answers. Finally, the we also analyze the external validity threat, and we believe that Section 4.3 demonstrated that the respondents of our session, during the FIA event, show a demographic distribution that is very close to the one of the whole population of registered session participants. Still, the participants in our survey study have not been selected randomly, which implies that generalization of results should be done with caution [3, 2].

Based on the aforementioned, therefore, we believe that the results of this survey study can contribute to understand where adaptation can play a key role for FI Apps. It also demonstrates which are the research fields where adaptation issues can to be further explored, like human-in-the-loop, cross-area adaptation.

Fig. 5. Replies to Question Q4: Relevant Application Domains
5 Conclusions

This paper has introduced and discussed key characteristics for service-oriented systems in the Future Internet (FI Apps). In addition to identifying such characteristics based on the state of the art, we have performed an empirical study to assess the importance of those characteristics. The survey has confirmed some of the typical expectations (e.g., the importance of adaptation for service-oriented systems). However, the survey also lead to unexpected outcomes. Specifically, distributed and cross-area adaptation capabilities have been deemed least important, although one would have expected those characteristics to become highly relevant in the Future Internet. We believe this deserves further investigation.

Acknowledgments: We thank the participants of our survey at FIA 2011, the speakers and panelists of the FIA session on “Adaptive Services”, our fellow co-organizers, as well as our Paluno members for participating in the pretest of the survey. We would like to thank the kind provisioning of demographic data about the FIA session participants by Anne-Marie Sassen from the EC. The research leading to these results has received funding from the European Commission’s 7th Framework Programme FP7/2007-2013 under grant agreements 215483 (S-Cube) and 285598 (Finest).

References

Adaptive Future Internet Applications: Opportunities and Challenges for Adaptive Web Services Technology

Clarissa Cassales Marquezan\textsuperscript{1}, Andreas Metzger\textsuperscript{1}, Klaus Pohl\textsuperscript{1}, Vegard Engen\textsuperscript{2}, Michael Boniface\textsuperscript{2}, Stephen C. Phillips\textsuperscript{2} and Zlatko Zlatev\textsuperscript{2}

\textsuperscript{1}Paluno (The Ruhr Institute for Software Technology) University of Duisburg-Essen, Essen, Germany\
\{clarissa.marquezan, andreas.metzger, klaus.pohl@paluno.uni-due.de\}
\textsuperscript{2}IT Innovation Centre, University of Southampton, Southampton, U.K.\
\{ve, mjb, scp, zdz@it-innovation.soton.ac.uk\}

Abstract
Adaptive capabilities are essential features to guarantee the proper execution of Web services and service-oriented applications once dynamic changes are not exceptions but the rule. In fact, the importance of adaptive services significantly increases in the context of Future Internet (FI) applications once they will be composed of a multitude of diverse types of services that offer flexible, remote access to software features, content and computing resources. Therefore, applications in this context, will have to autonomously adapt to changes on service provisioning, availability of things and content, computing resources, and network connectivity. Current proposals for adaptive Web services and adaptive service-based applications will be challenged in such context because they fall short to support essential characteristics in FI applications. This chapter will analyze and justify the need for the transition from adaptive Web services and service-based applications to adaptive FI applications. Based on two real-world use case from multimedia and transport and Logistics domains, we examine how current adaptive solutions need to be enhanced to properly address the adaptive needs of FI applications. Finally, we propose future challenges that need to be considered in adaptive FI applications.

1. Introduction
Adaptive capabilities are essential to guarantee the proper execution of Web services and service-oriented applications once dynamic changes are not exceptions but the rule. In fact, the importance of adaptive services significantly increases in the context of Future Internet (FI) applications. As it will be described in this book chapter, applications in this context will have to autonomously adapt to changes on service provisioning, availability of things and content, computing resources, and network connectivity. The unprecedented level of heterogeneity and dynamic changes of FI applications will demand a transition from adaptive Web Services and service-oriented systems to adaptive FI applications.
Three major pillars constitute the FI: the Internet of Services (IoS, e.g., software services based on third-party services), the Internet of Things (IoT, e.g., smart sensors and devices) and the Internet of Content (IoC, e.g., video streams and online games). These pillars will all converge into an integrated environment (Domingue et al., 2011). It is expected that – to a large extent – FI applications will thus be composed of third-party offerings deployed on federated service delivery platforms through different cloud delivery models, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) (Armbrust et al., 2010).

Ultimately, this means that loosely-coupled Internet services will form a comprehensive base for developing value-added applications in an agile way (Metzger & Di Nitto, 2012 (to be published)). This is unlike traditional application development, which uses computing resources and software components under local administrative control. To maintain their quality of service, FI applications therefore need to dynamically and autonomously adapt to an unprecedented level of changes that may occur during runtime.

**Problem Statement.** Over the past decade a wealth of technologies for engineering adaptive Web-based services and service-oriented systems has emerged. Those technologies offer significant advancements for what concerns furnishing applications with self-adaptive capabilities. Still, those solutions focus on isolated pillars of the Future Internet only. For example, many solutions consider software services (i.e., IoS) but fall short of integrating things (i.e., IoT) which leads to different levels of heterogeneity and unprecedented level of change on available resources and data.

**Contribution.** In this chapter, we review trends and current solutions for adaptive Web services and service-oriented applications. Based on real-world use cases from multimedia applications, as well as transport & logistics, we examine the transition from adaptive Web services to technology supporting the engineering and operation of adaptive FI applications. We demonstrate that FI applications promise full integration and combination of real, physical world services, business objectives and ICT services. This means a shift from considering adaptation only in the ICT level (and business level, as some initiatives already show), i.e., such as service-oriented applications, but extending this concept to unprecedented levels. The first use case explores how adaptation of IoC and IoS should be considered in FI multimedia applications. The second use case demonstrates the importance of combining IoT, IoS, and business objectives for the success of FI transport & logistics applications. Finally, we conclude with relevant research challenges that remain to be addressed.

In the next two sections we describe the major facts and topics that are relevant for the purpose of this book chapter. We start with the introduction of how research and ideas related to Web-services, service-oriented applications, adaptation and Future Internet evolved over the past years and got connected to each other. After this, we present an overview of technical solutions specifically developed for adaptation of Web services and service-oriented applications. The fourth section presents the use cases and describes their scope and adaptation needs. Then, we discuss which are the challenges faced in those use cases to advance from adaptive service-oriented applications to adaptive FI applications.
2. The Road from Web Services to Future Internet Applications and the Need for Adaptation

The goal of this section is to show the evolution of ideas starting from the development of Web services and service-based applications, passing through the design of adaptive solutions, until the definition of FI applications and the discussion about the need for adaptive characteristics in FI applications. This section is organized in three parts which address, respectively, each one of the ideas mentioned before.

2.1. Web Services and Service-based Applications

Web services technology (e.g., standards such as WS-BPEL, WSDL, SOAP, WS-Transaction, WS-Coordination, WS-Policy) and the Service Oriented Architecture (SOA) are increasingly adopted by practitioners for building highly dynamic and distributed applications. Such service-based applications (also known as composed services, service-based systems, or Web services based applications) are typically realized by composing individual Web services. Fundamental changes related to how software is developed, deployed, and maintained (Di Nitto, Ghezzi, Metzger, Papazoglou, & Pohl, 2008) emerge with the use of Web services and service-orientation. Software services separate ownership, maintenance and operation from the use of the software. Service users do not need to acquire, deploy and run the individual piece of software, because they can access the functionality of that software remotely through the service’s interface, as illustrated in Figure 1. Ownership, maintenance and operation of the software remains with the service provider, and thus can be performed by third-parties (Di Nitto et al., 2008).

![Figure 1. Service-based applications and the involved stakeholders](image)

In a simplified way, it is possible to say that there are three major stakeholder roles involved in service-based applications using third-party services as illustrated in Figure 1. The first one is the service provider who owns the Web services. The second is the service integrator who is responsible for invoking and composing the third-party services. The third is the entity that consumes the composed services provided by the service integrator. Figure 1 also depicts the composed service (represented by a workflow) provided by the service integrator. In this example one can observe that the Web services from service providers “A”, “B” and “C” are composed and exposed to the consumer as atomic and self-contained applications. The resulting environment of service-based
applications is extremely heterogeneous, decentralized and subject to highly dynamic changes such as the availability of the third party services, changes on the code of the third-party services, and network connectivity among the stakeholders (Papazoglou, Pohl, Parkin, & Metzger, 2010) (Qureshi & Perini, 2010).

2.2. Need for Adaptation

Web service based applications cannot be specified and realized completely in advance (i.e., during design-time) due to incomplete knowledge about the interacting parties (e.g., third party service providers or actual end-users) and the infrastructure where it will be executed (e.g., Internet, cloud, pervasive environment), as well as the changes mentioned above. Therefore, compared to traditional software systems, service-based applications need to be furnished with adaptation capabilities in order to decide on whether and how to modify themselves during operation (i.e., during run-time). Adaptive capabilities are essential features to guarantee the proper execution of service-based applications, once dynamic changes are not exceptions but the normal behavior imposed on these applications. Over the past years, many efforts have been made towards adaptive Web services and service-based applications. Based on the analysis of the literature (Papazoglou et al., 2010) (Mancioppi, 2011), Figure 2 depicts an illustrative and simplified time-line summarizing some of the main groups of efforts especially related to service-oriented applications. This time-line is not an extensive and exhaustive description of the field: it aims at capturing and illustrating some keywords associated to adaptive Web Services and service-based applications with the focus more on the technical level rather than on the business level.

According to the literature, it is possible to map three waves of research efforts, which are described as follows.

i. The first wave of initiatives addressed adaptive Web services through the use of more flexible and adaptive composition engines (e.g., BPEL engines) (Jang, Choi, & Zhao, 2003) (Charfi & Mezini, 2004) (Patel, Supekar, & Lee, 2004) (Zhou, Tang, & He, 2005) (Siljee, Bosloper,

ii. The second wave of efforts identified the need for reacting to dynamic changes during the execution time (i.e., runtime) (Bai, Xu, Dai, Tsai, & Chen, 2007) (Bianculli & Ghezzi, 2008) (Fabra, Ivalés, Bares, & Ezpeleta, 2008), where self-healing and self-adapting solutions started to be proposed (Baresi, Guinea, & Pasquale, 2007) (Di Nitto et al., 2008) (Hielscher, Kazhamiakin, Metzger, & Pistore, 2008).

iii. In the third wave, instead of reacting to runtime and dynamic changes, the proposed solutions started to focus on preventive and proactive adaptation (Psaier, Juszczyk, Skopik, Schall, & Dustdar, 2010) (Dranidis, Metzger, & Kourtesis, 2010). Service Level Agreements (SLA) and its QoS regulate the relationship among the stakeholders, and the management of SLAs became an important research topic towards providing adaptive service-based applications (Di Modica, Tomarchio, & Vita, 2009) (Zheng & Lyu, 2010). In addition, the emergence of the Internet of Services (IoS) (Corradi, Lodolo, Monti, & Pasini, 2008) (Spillner, Winkler, Reichert, Cardoso, & Schill, 2009) broadened and intensified the need for adaptive service-based applications in a third-party environment (Qureshi & Perini, 2010)(Mos et al., 2009) (Brenner, Atkinson, Hummel, & Stoll, 2007).

Indeed, the need for adaptation will become significantly more important in the context of the FI. A clear trend indicates that future service-based applications will be increasingly composed of third party services that are accessible over the Internet (Domingue et al., 2011). FI applications context, nevertheless, also involve more elements that contribute to an even more dynamic and heterogeneous environment. The next section characterizes the environment in which FI applications will be designed and executed, which starts the discussion about the adaptation need in this context.

2.3. Future Internet Applications and Adaptation

One of the main characteristics of the Future Internet is the convergence among areas: IoS, IoT (e.g., sensors and smartphones), IoC (e.g., video streams and online games), and the Network of the Future (NoF, which involves cloud infrastructures and ubiquitous network connectivity). Future Internet Applications (FIApps), as we call, will be able to explore and combine those different areas and service technologies will play a key role on enabling the relationship among those areas. For example, the information associated with sensors of the IoT will be accessed using Web service technology, so that their functionalities can be discovered and accessed over the Internet (in fact, proposals in this direction have already been made (Spiess et al., 2009) (Guinard, Trifa, Karnouskos, Spiess, & Savio, 2010) (Shelby, 2010)). FIApps are expected to enable a higher degree of integration between the ICT and real-life environments and business domains.

Ultimately, FI Applications are built upon the FI Infrastructures (ISTAG, 2008) that are loosely-coupled Internet services forming a comprehensive base for developing value-added applications in an agile way (Tseltentis, Domingue, Galis, Gavras, & Hausheer, 2009) (Domingue et al., 2011). Unlike traditional application development, which uses computing resources and software components under local administrative control, FI applications will thus strongly depend on third-party services. To maintain their QoS, those applications therefore need to dynamically and autonomously adapt to an unprecedented level of changes that may occur during runtime.
This book chapter, therefore, intends to highlight the importance of properly engineering adaptive FI applications having in mind the new scenario and conditions present in FI Infrastructures. In the next section we present a brief overview of current adaptive solutions applied in Web services and service-based applications. Two case studies of adaptive FI Applications in different domains are discussed and we argue why the current adaptive solutions need to be extended and enhanced in order to properly provide adaptation characteristics for such case studies.

3. Overview on Solutions for Adaptive Web Services and Service-based Applications

The literature associated with adaptive Web services and Service-based applications is very diverse. In some cases, the word adaptation is not employed; however, the effects of the actions proposed by these solutions are the same as the ones expected from adaptive solutions, i.e., healing, reconfiguration, and adjustment in face of changes in the Web services or service-based applications. The intention of this section is to present an overview of adaptive solutions for Web Services and service-based applications.

There are four major elements that characterize adaptive solutions: (i) monitoring properties; (ii) adaptation decision mechanisms; (iii) strategy decision mechanisms; and (iv) realization mechanisms (Papazoglou et al., 2010). The first one is vital in order to acquire information about events and status of the execution of Web services and service-based applications. Based on the monitored information, adaptation decision mechanisms can analyze which of the reported events, for instance, should demand an adaptation action. The strategy decision mechanisms are associated with the task of planning how an adaptation action should be executed. Finally, the realization mechanisms enforce the adaptation plans.

The literature shows that some proposals focus in the combination of all those elements (Mannava & Ramesh, 2011) (Sheng, Benatallah, Maamar, & Ngu, 2009), the combination of few of them (Brogi & Popescu, 2006) (Mosincat & Binder, 2011), or in one element in specific (Bianculli, Jurca, Binder, Ghezzi, & Faltings, 2007) (Ivanović, Treiber, Carro, & Dustdar, 2010). In addition, there are proposals that focus on particular perspectives such as time and layer when the adaptation activities are executed and also how centralized of distributed are those proposals. In this section we present an overview on works addressing these different foci on adaptive Web services and service-based applications.

Solutions proposed in (Sheng et al., 2009), (Mannava & Ramesh, 2011) (Psaier et al., 2010) (Erradi & Maheshwari, 2008) (Gui, De Florio, Sun, & Blondia, 2011) (Pernici & Rosati, 2007) share, in a high level, the same adaptive characteristics, i.e., these proposals are able to monitor, analyze, plan and execute actions related to adaptation of Web services or service-based applications. The scope of the adaptive solution can be similar or vary significantly from one proposal to another. For example, proposals use context information as an asset of the adaptive solution. Nevertheless, the same proposals focus on different abstraction levels of adaptation, i.e., for instance, targets the adaptation of the workflow model (Sheng et al., 2009) while in (Erradi & Maheshwari, 2008) focuses on adapting the running instance of a composition. Different from previously mentioned, the proposals in (Psaier et al., 2010) and (Pernici & Rosati, 2007) do not use context information but both are related to repair and model the “misbehavior” in Web services compositions.

There are also proposals that focus on different adaptation activities. For instance, the work presented in (Mosincat & Binder, 2011) is mostly focused on monitoring and analysis phases of adaptation, providing monitoring of process and service performance and ensuring maintenance of
process performance through automated detection of service failures and SLA violations, diagnosis, and repair. In (Hepner, Baird, & Gamble, 2009) the focus was on formally specifying reconfiguration of BPEL workflows based on dynamic web service changes. The other work in (Ivanović et al., 2010) focuses on the analysis phase of adaptation. Such a solution addresses the automatic derivation of dynamic, continuous-time models of behavior of service orchestrations to assure that the specified QoS levels are met.

Examples of proposals focused on the time perspective of adaptation actions are: (Dranidis et al., 2010) that describes a novel approach for just-in-time testing of the behavior of conversational services which allows potential failures to be detected shortly before the execution of services, thus enabling the service compositions to be adapted pro-actively. In (Leitner et al., 2010), SLA violations on service-based applications are predicted using regression models which have been trained based on monitoring information from past process instances. In (Ivanovic, Carro, & Hermenegildo, 2010), solutions were proposed for quality assurance of service-based applications execution and quality prediction techniques to support proactive adaptation.

In addition, there are some trends exploring the cross-layer approach for executing adaptation activities. For instance, Baresi et al. (Baresi, Guinea, Pistore, & Trainotti, 2009) proposed a monitoring framework that enables the integration of a wide range of events into more complex properties. The authors created an extended recursive model of monitored properties and capabilities to correlate and aggregate a variety of events from independent sources which enabled cross-layer service-based applications monitoring. In (Wetzstein et al., 2009), it was also proposed a cross-layer monitoring solution. The focus of this work is to define an integrated framework for runtime monitoring and analysis of the performance of WS-BPEL processes in a cross-layer setting based on machine learning. In (Popescu, Staikopoulos, Liu, Brogi, & Clarke, 2010) it was proposed a cross-layer adaptation methodology which attempts to enhance and semi-automate the adaptation in multi-layer applications by combining templates (which are specially target to deal with behavior adaptation issues) and taxonomies of adaptation issues (which are intended to support the semi-automated discovery and selection of cross-layer adaptation templates). Finally, (Kertesz, Kecskemeti, & Brandic, 2009) addressed the problem of renegotiating SLAs across the SOA layers in order to adapt to changes that might lead to SLA violations.

Despite of the fact that Web services and service-based applications are technologies by nature associated with distributed environments, we cannot take for granted that solutions developed for those technologies are decentralized. For instance, in (Yau, Huang, & Zhu, 2007) it was defined a virtual machine-based architecture for the execution, monitoring and control of workflows in service-based systems. The model adopted by Yau et al. includes centralized entities reasoning based on of distributed information. The work described in (El Falou, Bouzid, Mouaddib, & Vidal, 2010) does not address directly adaption, but it focuses on the planning phase of services compositions. The authors proposed an iterative and hierarchical solution for planning Web services composition, which helps to reduce the computational load for defining a composition plan. In (Tang & Xu, 2006), the authors create an adaptive model of service composition which is based on policy driven and multi-agent negotiation. Although it is not explicitly claim by the authors, in theory, the model proposed by the authors would allow their services to interact in a total decentralized fashion. The decentralized model used by the authors considers decentralized decision based on distributed information.

As briefly discussed above, there are many different perspectives and solutions proposed for adaptive Web services and service-oriented applications. In summary, the main focus of the current
solutions addressing adaptation are still related to the ICT aspects of the services, some integration between business objectives and technical level (mainly in respect to monitoring and predicting), and some integration with the underlying computational infrastructure supporting the ICT services. To move from adaptive service-oriented applications to FI applications the current techniques will have to be extended in order to accommodate the upcoming requirements of the applications.

4. FI Application Domains

In this section we introduce two use case scenarios from two different application domains to motivate the need for extending and integrating existing service adaptation techniques in order to properly design adaptive FI applications. The first use case is related to multimedia applications and the second is related to the transport & logistics domain.

4.1 Multimedia Applications

Multimedia applications are typically highly interactive and especially challenging to provision so that their performance is stable and the user experience is adequate. These challenges are due to the complexities in estimating the software performance on a given hardware infrastructure and also due to the varied behavior of the users resulting in varied and difficult to assess workloads. The SaaS paradigm gives the capability to multimedia application providers to reach an increasingly large number of users and also enables users to use the applications as a utility rather than having to unnecessarily invest in infrastructure and software. In addition, such a paradigm is able to better deal with provisioning challenges of applications once auto-scaling techniques can be used by the providers in order to compensate the changes on the application demands.

These challenges of multimedia application can be addressed by the means of adaptive environments used to operate the applications provided. Research in the IRMOS (IRMOS, 2011) and BonFIRE (BonFIRE, 2011) projects address these challenges. For instance, the IRMOS project addresses the challenges in an IoC scenario that explores an interactive real-time Application as a Service. In this example, SaaS providers encounter challenges in providing soft real-time multimedia applications with guaranteed (probabilistic) QoS. Examples of such applications include interactive and collaborative film post-production, virtual and augmented reality within the engineering design process, and interactive online eLearning environments. Consider the post-production scenario, for example, in which resources are needed on short notice to host a session with many users located in different countries to work on a film. Compute, storage and networking resources need to be selected to ensure the QoS to the users, who will be streamed a video and will interact with it by, for example, pausing, rewinding, or editing frames. One Key Performance Indicator (KPI) of such an application is whether frames are dropped, which should have certain QoS guarantees.

The IRMOS & BonFIRE Solutions for the IoC Scenario.

The IRMOS project has developed a toolbox of techniques that allow applications with soft real-time requirements to be planned and executed on virtualized service oriented infrastructure operated by third-party service providers. In the case of the IoC scenario, there is a need for well-defined and SLAs that have guaranteed QoS. The IRMOS toolbox provides an adaptive environment of tools for negotiating, monitoring and managing SLAs and applications.

One of the main components in the IRMOS toolbox is the Performance Estimation Service (PES), as seen in Figure 3, which encapsulates a methodology for the SaaS provisioning planning. The PES is used for planning the deployment of the SaaS application in terms of resources that
need to be reserved, as well as during the operation of the application to adapt its provisioning in response to critical events. A critical event could be, for example, observing KPI deterioration below the agreed level, or observing a deviation from the expected application workload that may compromise the agreed QoS.

During SLA negotiation, the PES estimates the required resources for a particular application based on the predicted performance of an application model on some given resources (according to availability by an IaaS provider). Details of an example model for the e-Learning application mentioned above can be found in (Cucinotta et al., 2010). The predicted performance is evaluated against the QoS terms set in the SLA with an objective function that encapsulates the business objectives of the SaaS provider. This is typically a function that evaluates the maximum profit for the SaaS provider, based on the expected income for running the application, minus the infrastructure costs and any penalties for not fulfilling the QoS. The IRMOS toolbox includes global and local optimization algorithms to determine the best resources according to the defined objective function, supported by a framework of caching execution results of application models and objective functions.

*Figure 3. IRMOS solution for IoC scenario in multimedia application domain*
Once the SLA is agreed, the application is deployed and run. To guarantee the agreed QoS, a performance feedback loop is implemented that facilitates reactive and proactive types of application provisioning adaptation via a Performance Feedback Service (PFS). During the application operation, KPI metrics are logged and fed back to the PFS. These observed metrics are compared against the agreed ones and in the events of deviation the resources are scaled according to pre-defined rules.

The work on predicting application performance in IRMOS has been continued in the BonFIRE project (BonFIRE, 2011), which offers a multi-site testbed of heterogeneous cloud resources for experimentation on the FI. One of the main focus of the work continued in BonFIRE is to address the challenges of predicting application performance for QoS estimation based on the descriptions of resources offered by IaaS providers.

Today, IaaS QoS offerings are expressed in low level terms (i.e., machine level, CPU speed, disk space, etc), whilst their customers, typically application users, are often interested in application-level parameters because the application is the thing that gives the customer the value (e.g. CFD simulation or video rendering). Therefore, the gap between the terms the Infrastructure provider offers and what the users really want is large which results in a complex relationship between application performance and resource parameters. The complexity of this relationship is increased for applications deployed across federated clouds where even low-level resource descriptions may differ due to lack of standardization.

IaaS resources should ideally be described in a uniform and descriptive manner, which can be fruitful for predicting application performance. The Dwarf taxonomy introduced in (Cavallo, Di Penta, & Canfora, 2010) is one alternative, which currently comprises 13 classes of computational benchmarks (Asanovic et al., 2009). A Dwarf benchmark is defined as “an algorithmic method that captures a pattern of computation and communication” (Asanovic et al., 2006). Ultimately we could imagine each IaaS provider describing the performance of their resources in terms of a standard set of benchmark scores (such as Dwarfs) and even agreeing SLAs in those terms. Alternatively, a PaaS provider may measure the performance of many IaaS providers, adding to one of many possible services that could be offered.

Initial findings in (Phillips, Engen, & Papay, 2011) indicates that scores on Dwarf benchmarks shows promise as a means of describing computing resources in the cloud, demonstrating that they can discriminate between different compute resources, even when the IaaS provider labels them the same. Furthermore, the initial findings in (Phillips et al., 2011) show different Dwarfs correlate more strongly with different applications, indicating that they can be useful in predicting application performance. Moreover, expression of IaaS parameters in this way can simplify the creation of application-level QoS that can be easily understood by users, as well as improving robustness and adaptability to QoS changes by making it easier to determine suitable resources to deploy from different IaaS providers.

4.2 Transport & Logistics Cloud Services

The Transport & Logistics (T&L) domain is a global industry which represented 13.8% of the global GDP and 10-15 % of the final product costs according to the communication of the European Commission in 2006 (Commission of the European Communities, 2006). In addition to its economical relevance, T&L domain produces environmental impacts and, in 2005, this industry was responsible for 14.3% of the world’s greenhouse gas emissions (World Resources Institute,
Despite of the achieved improvements in this domain, there are still major obstacles to be overcome in order to optimize the execution of T&L processes and enable a more efficient and sustainable industry. Examples of such obstacles are: (i) limited visibility on the T&L processes and critical events, i.e., during the transportation of the goods the involved partners have only a partial knowledge of what is actually happening; (ii) closed logistics supply chains which reduces the inter-organizational information exchange and collaboration; and (iii) the T&L processes are still highly dependent on manual intervention (Metzger & Marquezan, 2011). Different projects tackled different angles of these problems (eFreight, 2010) (FREIGHTWISE, 2006). One example is the Flnest project that investigates the employment of FI areas in order to design and develop T&L FIApps.

Figure 4 illustrate the solutions envisioned by Flnest. The front end layer of the Flnest platform provides users with role specific, secure, ubiquitous access from different devices to information concerning the operation of the transport and logistics network. The back end layer of the Flnest platform provides access to, and integration with, legacy systems, third party services and any IoT devices that may provide information during the transport lifecycle. Legacy system integration is facilitated by service-oriented technology, e.g., by exposing features of legacy systems as services, or by offering access to legacy systems via the SaaS (Mietzner, Metzger, Leymann, & Pohl, 2009). The core layer of the Flnest platform is composed of independent transport and logistics service modules (eContracting, Proactive Event Monitoring, Transport Re-Planning) integrated through the Business Collaboration Module. The independent service modules are cloud-based applications that provide essential domain services for the shipment of goods.

The Flnest services are realized by leveraging FI services based on the IoT and IoS cur-
rently under development through the European Union’s Future Internet Public Private Partnership (FI PPP) program. The following list describes the FI technologies of primary importance for implementing the FInest Platform. Most of these are addressed within the FI-W ARE project, which develops the FI Core Platform within the FI PPP program (www.fi-ware.eu):

- Infrastructure, methodology, and tools for cloud-based platform and application development, including an infrastructure for deploying the FInest Platform and its components on public or private clouds along with methodology and tool support for developing additional end-user services for individual transport and logistics stakeholders;
- Language and tool support for the IoS, including a service description language that covers both technical and business requirements along with integrated tool support for the provisioning, management, and consumption of services; this shall be used for realizing the back-end layer of the FInest platform and for managing the interaction of the FInest core modules;
- Access to real-world data from the IoT, enabling the integration and technical handling of real-world data obtained from sensor networks for real-time monitoring and tracking during the execution phase of transport and logistics processes;
- Facilities for data and event processing in the Internet of Contents, allowing to process huge amounts of data to retrieve insights into relevant scenarios, as well as analyzing real-time event data to quickly determine relevant situations and instantly trigger actions.

Adaptation Needs of FI applications for T&L.

Three scenarios were identified by the FInest project and one is related to the process of exporting goods from Turkey to the United Kingdom (UK). In this scenario, a supply chain composed of material supplier, truck carrier, sea carrier, warehouse, and material assembly, are formed. The proper support for the end-to-end visibility of such supply chain is provided by FInest platform through the use of IoT and IoS. In this section, we present what are the adaptation requirements that need to be considered in such scenario.

For example, suppose that the vessel of the sea carrier has sensors and cameras for monitoring the status of the transported material. For the end-to-end visibility of the supply chain it is necessary to keep the availability of information about the material status. In the situation, of damages on some sensors the information received by the application is not complete, i.e., part of the monitored area is not covered any more. The option to adapt to this change on the environment would be to change the source of the information from sensor to camera in the monitored area. In a first instance, one can think that the service substitution technique traditionally employed in Web services and service-oriented application would suffice. For example, instead of using a service that provide information coming from sensors the system discover and substitute the “current failed service” for another. But how to detect that the service associated with the sensor information has failed? The service itself is available and sending data, but the problem is that the data is not the one expected by the application.

The notion of service failure might also be reevaluated and might open opportunities for exploring the cross-cutting aspects of adaptation. For example, the service composition layer might not be able to recognize a failure, but this might also require the intervention of the business layer in the monitoring process. Current monitoring cross-layer solutions could be the start point. The combination of information coming from the sensors, services compositions, and business layer might advance the design of cross-cutting aspects adaptation support.

Another issue in this scenario is the need for a decentralized monitoring process. Current
decentralized monitoring techniques, as discussed in the previous section, could help, for instance, to support the distributed information monitoring need in this scenario. Nevertheless, the current decentralized solutions for adapting service-based applications would not provide the required support in that scenario, because they are not able to operate in a federated third-party environment. Suppose that each player in the supply chain of this scenario is an ICT service integrator itself. For example, the truck carrier would compose its own services and offer it to the supply chain as a unique service. The federation of the parties services would form the supply chain. In this case, a centralized entity integrating the services of each one of the parties would not be able to direct execute decentralized adaptations along the infrastructure of the parties, because it would not have access to the infrastructure of each party. Thus decentralized negotiations must be considered in order to adapt to changes in the services of this scenario.

4.3 Remarks

In contrast to current service-based applications, FI applications will (i) continue to increase the scale and heterogeneity of involved parties, and (ii) provide a much stronger connection between real-life elements and the business connected by the application. Thus, the environment of FI applications tends to be much more dynamic and heterogeneous than the ones which service-based applications are subjected to. The techniques and models of adaptive service-based applications may be considered as the start-up of adaptive FI applications; nevertheless, enhancements are needed to properly engineer and execute adaptation in the hostile FI applications environment.

5. Requirements and Challenges to Enable Truly Adaptive FI Applications

One key enabler for adaptive FI applications, such as the ones above, will be the ability to have a seamless and consistent way of monitoring, detecting and predicting critical events for the different areas of the FI and thus for the types of services offered. As an example, consider that an IoT-based RFID-sensor that would track packages within a warehouse fails, seamless monitoring would allow switching to video data and video analysis to track those packages. Further, due to the very large scale of FI applications, this requires significant progress towards decentralized and highly dispersed facilities for monitoring, detecting and predicting critical events. On the other hand, the high volume of data from different sources (such as services, things or media items) may provide refined approaches towards detecting and predicting critical events; e.g., by correlating trends from those various sources or applying more powerful complex event processing facilities.

5.1. Detection

As indicated in Section 4.1, many practical applications will have “soft” QoS requirements. This means that the decision whether QoS expectations have been violated is not a clear cut one, but needs to rely on objective functions (such as the one introduced from the point of view of the provider in IRMOS). In addition, utility functions that assess the “severity” of the violation from the point of view of the end-user need to be taken into account for assessing whether a monitoring event indicates a critical event. This means that the notion of Quality of Experience (QoE) needs to be considered during detection. When talking about QoE, the role of the context in which an application executes will have an additional impact; for instance, a user traveling in an airplane
might not necessarily expect to have broadband access to his/her services and thus would be satisfied with much less connectivity.

5.2 Prediction

The prediction of critical events (such as deviations in the QoS of constituent services or SLA violations of service-oriented systems or multimedia applications) will always have a margin for error. Thus, in order to decide whether a proactive adaptation decision should be based on the prediction of a critical event, metrics and tools need to be in place in order to assess the accuracy of such a prediction. Moreover, the availability of a wider range of data sources in the FI will provide an added level of accuracy, as predictions can be based on more and more frequent information. For example, one could correlate trends from those various sources or apply more powerful complex event processing facilities. However, how to exploit this in an integrated and coordinated way is still an open question.

In addition, the highly dynamic setting of FI applications will make it difficult to select the best and most suitable prediction technique during design time. As discussed in Section 4.1, benchmarking application behavior to better predict infrastructure properties is quite challenging particularly since there is currently no uniform way of describing resource offerings in the cloud. This is a challenge that is currently investigated in the BonFIRE project, but is only one piece of the puzzle for achieving scalable and dynamic prediction of application performance. For example, dynamic approaches to adapting the prediction models based on operative data could be an interesting research direction.

Finally, predicted problems may be "contextualized" in order to better understand the criticality of the forecasted event. Here, again QoE considerations may play an important role. In addition, cost models become important in order to balance the costs of not taking an adaptation vs. the cost of doing so and thereby being able to quantify the risk involved in both these decisions. As an example, the cost of the predicted violation might be smaller than the penalty to be paid for an SLA violation. Such a critical event could then be safely ignored.

5.3 Cross-cutting Aspects

The capacity of dealing with cross-cutting aspects that cut across different areas and layers of the FI is a characteristic that is not entirely covered by current strategies. Existing solutions, such as the ones developed in the EU Network of Excellence S-Cube (Papazoglou et al., 2010), focus on cross-layer aspects of the service-oriented architecture only (i.e., on the service infrastructure, service composition, and business process layers). However, applications are envisioned to encompass all areas of the FI (e.g., IoS, IoC) in an integrated fashion so that IT systems and business can be tightly integrated with the physical world (Domingue et al., 2011).

One question related to cross-cutting is: where does cross-layer/area adaptation start? The idea of different areas of FI is not mature enough. In fact, there are many discussions about what exactly the FI is. However, there is at least a common sense that in the future there will be a convergence among the many areas on current isolated development, such as IoS and IoT. The convergence of different types of networks already started, and will be the basis for the NoF. Adaptation strategies, for instance, are being developed in each one of the fields. But when it comes to the FI context, how to draw the limits among cross-layer and cross-area adaptation actions? For instance: Is the disruption of network connectivity a problem to be solved only by the NoF in the FI Platform layer, or should the FI application be able to decide which kind of connectivity adaptation would
be more suitable for the application scope? This simple question opens up many opportunities for investigating revolutionary designs and methodologies devoted to create adaptive FI applications.

5.4 Decentralization

Existing solutions typically employ a central entity for gathering information from distributed sources and for taking decisions. This leads to at least two critical problems: (i) in a third-party service provider environment, the central entity will only have limited access to information about the external service provider (for example, nothing can be known about the invocation load of the external service); (ii) a central entity will present classical problems like single point of failure, lack of scalability, and bottlenecks which will not be acceptable once the scale of the FI includes not only traditional devices (e.g., PCs, notebooks, smartphones), but also sensors, RFIDs, or any kind of device that can be endowed with network communication features. There are solutions that employ hierarchical strategies to gather and analyze the information, which reduces the impact of centralization. Nevertheless, these solutions tend to ignore the restrictions imposed by the third-party environment when executing the actions associated to adaptation (i.e., monitoring, analyzing, planning, and executing). Nevertheless, how much decentralized adaptive decision-making process can we really use? This question is also very much related to the role of humans in the design of decentralized adaptive FI applications. Maybe autonomic decisions might be designed based on decentralized models, but human-in-the-loop decisions might not be designed solely based on fully decentralized models.

5.5 Boundaries

Given the strong interaction of FI applications with the real / physical world, where do we draw the boundary between normal application logic and adaptation? One of the important changes that will be introduced by FI applications is the strong connection with the physical world through the use of IoT technology. The ICT services will be merged with real world services. For example, one type of logistics services is the door-to-door shipment of goods. When a person contracts a shipment through a logistics FI application, the ICT services will help the business owner to execute such logistics services, in a very similar way as described in Section 4.2 (combining IoS and IoT). This merge of ICT systems and services involves the real world, which will create boundary issues related to adaptation. For instance: Might we consider the change on a dispatching truck executed by a transport & logistics IT system already an adaptation? What is the actual system (entity) that is being adapted? If only the business process changes (but not the underlying IT System), is this adaptation? These are questions that emerge from this deep relationship of ICT systems and real world and that need to be better investigated.

5.6 Remarks

The discussion above tried to capture, based on concrete scenario problems, questions that need to be understood and answered in order to engineer truly adaptive FI Applications. The challenges and requirements discussed in this section are not an exhaustive and closed list. We discussed some aspects of adaptation, i.e., detection, prediction, cross-cutting and decentralization, and identified the gaps that current solutions proposed to address those aspects fail to properly tackle.
6. Conclusion

There are many questions to be answered, and for each question new ones emerge. Despite all the uncertainties surrounding FI applications, there is at least one certain and incontestable fact: FI applications will have to be engineered explicitly considering adaptation aspects. In this book chapter, we discussed the need of thinking and designing FI applications considering aspects beyond the ones considered by current adaptive Web services and service-based applications. To enable adaptive FI applications, it is clear that a seamless and consistent way of monitoring, detecting and predicting critical events, dealing with cross-cutting aspects, decentralization, and boundaries between application logic and adaptation needs is required. Besides other challenges, applications need to be able to dynamically adapt to an unprecedented level of changes that can occur during runtime.

Acknowledgment

The research leading to these results has received funding from the European Community’s 7th Framework Programme FP7/2007-2013 under grant agreements 214777 (IRMOS), 215483 (SCube), 257386 (BonFIRE), 285248 (FI-WARE) and 285598 (FiNest).

References


BonFIRE. (2011). EC FP7-ICT BonFIRE Project. (Available at http://www.bonfire-project.eu/.)


Mancioppi, M. (Ed.). (2011, November). Consolidated and updated state of the art report on Service-Based Applications. (Deliverable CD-IA-1.1.7 of S-Cube Project.)


SLAs for Cross-layer Adaptation and Monitoring of Service-Based Applications: A Case Study

Konstantinos Bratanis
South-East European Research Centre,
CITY College,
International Faculty of the University of Sheffield,
24 Proxenou Koromila Street, 54622, Thessaloniki, Greece
kbratanis@seerc.org

Dimitris Dranidis
Department of Computer Science
CITY College,
International Faculty of the University of Sheffield,
3 Leontos Sofou Street, 54626, Thessaloniki, Greece
dranidis@city.academic.gr

Anthony J.H. Simons
Department of Computer Science
University of Sheffield
Regent Court
211 Portobello Street
Sheffield S1 4DP, UK
a.j.simons@sheffield.ac.uk

ABSTRACT
Cross-layer adaptation and monitoring (CLAM) is an approach to the run-time quality assurance of service-based applications (SBAs). The aim of CLAM is to monitor the different layers of an SBA and correlate the monitoring results, such that in the event that a problem occurs an effective adaptation strategy is inferred for enacting a coordinated adaptation across all layers of the SBA. An important aspect of CLAM is the definition of the appropriate Service-Level Agreements (SLAs) for third party services utilised in the different layers of the SBAs. In this paper, we present insights into how to define SLAs for CLAM, by analysing SBAs in order to differentiate the third party business, software and infrastructure services utilised by the SBA. As a case study, we apply the analytical approach to an existing platform-as-a-service framework, which has been developed as an SBA and could benefit from CLAM. The analysis reveals the different third party services and their characteristics, as a precursor to defining SLAs. The case study successfully demonstrates how distinct SLAs for business, software and infrastructure services may be applied respectively in the BPM, SCC and SI layers of an SBA, to provide a flexible monitoring and adaptation response across layers.

Categories and Subject Descriptors
D.2.11 [Software Engineering]: Software Architectures—Service-oriented architecture (SOA); D.2.9 [Software Engineering]: Management—Software quality assurance (SQA)

General Terms
Reliability, Verification, Design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

QASBA ’11, September 14, 2011, Lugano, Switzerland
Copyright 2011 ACM 978-1-4503-0826-7/11/09 ...$10.00.

Keywords
Service-Based Application, Quality Assurance, Monitoring, Adaptation, Service-Level Agreements, Case study

1. INTRODUCTION
One of the main barriers to the adoption of service-based applications (SBAs) is the concern raised over the trustworthiness and reliability of third party services utilised in an SBA. The third party software services are often implemented as Web services that realise business activities, such as paying with a credit card or shipping purchased goods, and are beyond the control of the SBA provider. The problem of reliability becomes more complex when third party cloud computing services are utilised as the underlying infrastructure for provisioning the SBA. Given that the SBA provider does not have control over the quality of the third party services, unreliable third party services could threaten the quality of the SBA and result in lower business performance, software faults, and performance degradation that could consequently lead to the total collapse of the SBA. Therefore the dependability of the third party business, software, and infrastructure services utilised in an SBA becomes a principal concern for the SBA provider, who will require to adopt mechanisms within the SBA for quality assurance during run-time.

An approach to the run-time quality assurance of SBAs is the cross-layer adaptation and monitoring (CLAM), which aims on detecting problems early in the SBA layers and coordinating effective corrective actions across the SBA layers, such that problems are compensated for, or even prevented from occurring [9]. The functional layers of an SBA have been introduced in [6] and comprise the business process management (BPM), service composition and coordination (SCC), and the service infrastructure layers (SI). Based on the aforementioned separation of the SBA layers, in this paper we suggest that each layer concerns a different type of services. For example, the BPM layer concerns business services, the SCC layer concerns software services, and the SI layer concerns infrastructure service. It is necessary to perform analysis of the SBA to identify the business, software, and infrastructure services and their characteristics, in order to define appropriate Service-Level Agreements (SLAs) for such services. We present ideas for defining SLAs, used in CLAM approaches, by performing analysis of SBAs in
order to identify the third party services and their characteristics utilised in each SBA layer. We do not focus on a concrete reference architecture for CLAM. We describe a new case study for applying CLAM related to a platform-as-a-service (PaaS) offering, which has been implemented as an SBA. We analyse the PaaS offering, in order to identify the third party business, software, and infrastructure services and their characteristics for the definition of SLAs.

The rest of the paper is organised as follows. In Section 2 we present related work on CLAM. In Section 3 we describe insights for defining SLAs in SBAs. In Section 4 we present a new case study for CLAM related to a PaaS offering and we perform analysis for defining SLAs. Finally, in Section 5 we discuss conclusions and we provide an outlook for future research.

2. RELATED WORK

Recent research into run-time quality assurance has focused on implementing CLAM techniques for SBAs [9], by integrating the existing fragmented work in the field of adaptation and monitoring of service-based systems. Gjørv et al. [5] introduce a middleware for supporting the implementation of cross-layer self-adaptation of SBAs. Kazhamiakin et al. [6] describe a conceptual framework comprising the definition of the SBA layers and a set of requirements needed to be addressed by mechanisms and techniques for CLAM of SBAs. Popescu et al. [10] present a methodology for cross-layer adaptation using adaptation templates. Latest research has focused on SLAs for CLAM. More particularly, Fugini et al. [4] describe an SLA contract that comprises parameters from user goals, business service and IT infrastructure for CLAM of SBAs. Schnieders et al. [11] propose the combination of SLA prediction, which uses assumptions about the characteristics of the execution context, and cross-layer adaptation mechanisms for preventing SLA violations.

SLAs for third party services utilised in each SBA layer are an important element in such approaches, since SLAs specify the expected characteristics of each third-party service, named Service-Level Objectives (SLOs), to be monitored, and possibly adaptation strategies for compensating or even proactively preventing violations of SLOs. We support the research directions towards the run-time quality assurance of SBAs using CLAM techniques, and we believe that such techniques could greatly benefit from an analysis approach of SBAs for identifying third party services and their characteristics across the SBA layers for the definition SLAs.

To the best of our knowledge, we are aware only of one recent work that is related to the definition of SLAs for CLAM [4]. Although this work describes a methodology for creating SLAs, it focuses on the dependencies between the characteristics of services, and it does not follow the SBA layers as they have been defined in [9, 6]. The authors focus more on how KPIs and IT infrastructure metrics impact the goals of a service user, and they introduce a new indicator named Key Goal Indicator. Our work is different since we present ideas for defining SLAs, by performing analysis of SBAs in order to identify the third party services and their characteristics utilised in each SBA layer.

In the next section, we present insights into how to define SLAs for CLAM by analysing SBAs, in order to differentiate the third party business, software and infrastructure services utilised by an SBA.

3. SBA ANALYSIS FOR DEFINING SLAS

In the context of Service-Oriented Computing, an SLA specifies the exact functionality and the desired quality of service to be delivered by a software service [7]. An SLA includes a set of metrics and a behavioural specification that could be used to determine whether the service provider is delivering the service as agreed. An SLA could also include compensation actions in the event that the agreement was violated. Machine-readable SLAs for the third party services, used in an SBA, are utilised in CLAM. CLAM approaches monitor the third party services for detecting violations of the agreed service characteristics, in order to perform compensation actions across all layers of an SBA.

The three functional layers of an SBA are defined in [6, 9]. The top layer of an SBA is the business process management (BPM) layer and it concerns the business level aspects of an SBA, such as process workflows, service networks, key performance indicators, and process performance metrics. The BPM layer focuses mostly on monitoring business activities and manages the performance of the business. The middle layer of an SBA is the service composition and coordination layer (SCC), which concerns the composition of individual services into new services, the functional (e.g. service behaviour) and non-functional quality of service (QoS) (e.g. responsiveness and availability) characteristics of the individual services or the composed services. The SCC layer focuses mostly on both run-time verification and testing of the service behaviour, and monitoring the QoS of the individual or the composed services. The bottom layer of an SBA is the service infrastructure (SI) layer and concerns the software (e.g. service middleware, service registry) and the hardware (e.g. compute, storage, bandwidth) resources utilised in an SBA.

Based on the description of the SBA layers, we suggest that each layer concerns different types of services. The BPM layer concerns business services or business activities realised through software services. For instance, a shipping provider exposes a Web services API for shipping goods. The shipment of goods is a business activity provided through a Web service. The SCC layer concerns software services that implement a specific functionality or a business activity. For instance in the case of the shipping provider the Web service is the software service. The SI layer concerns the infrastructure services used by an SBA. For instance, an SBA could be running on a third party Cloud Computing infrastructure and rely on shared computing, storage, and networking resources. Based on the aforementioned suggestions, we argue that an SBA is a software application that outsources business activities, consumes software services, and uses infrastructure services.

Due the fact that BPM, SCC, and SI layers concern the business, the software, and the infrastructure services respectively, and considering that such services in each layer could be provided by third parties, it is necessary to have separate SLAs for all services in each layer. The SLAs in each of the three layers are required for monitoring the conformance of the services to the agreements. Given the possibility that there are dependencies between business, software, and infrastructure services, it is necessary for the monitoring activity to correlate the monitoring results from all services in each layer, such that the real cause of a problem is diagnosed, in order to conclude and enact an effective adaptation strategy.
In the following section, we present a new case study showing how the analysis of different types of service supports the introduction of CLAM in an existing platform-as-a-service offering, which has been developed as an SBA. The platform is analysed in order to identify the business, software, and infrastructure services and their characteristics for the definition of SLAs for such services in the three SBA layers.

4. CASE STUDY

In this section, we present a new case study showing the analysis of an SBA in preparation for CLAM. We provide a brief description of the SBA used in the case study. We continue by identifying the distinct business, software and infrastructure services and their characteristics as a precursor to defining SLAs for these services.

4.1 The CAST Platform

The case study concerns a platform-as-a-service (PaaS) offering for enabling the customisation of software-as-a-service (SaaS) applications by third parties, developed during the European project CAST\(^1\). The platform was developed using Java, OSGI, Web services, and other service-oriented technologies. A brief summary of the CAST Platform follows based on the work [8, 1, 2] carried out during the CAST Project.

The CAST platform offers the foundations for developing software ecosystems of domain specific solutions, which comprise apps and external Web services. A developer is able to create his own solution by implementing new apps or by extending the functionality of existing community apps, which are made available by other developers. In the context of the CAST Platform, a solution is a collection of multiple apps that are the building blocks of the platform for implementing a particular functionality. Figure 2 depicts the case of a Customer Relationship Management (CRM) solution in the CAST Platform. The CRM solution comprises an address app for managing customer addresses, a document app for managing documents, a translation app for translating documents, and a postcard app for sending postcards to customers. An app can interact with external Web services, which have been registered to the Governance Registry & Repository system of the CAST Platform. The main function of the registry is the lifecycle management and quality assurance of solutions, apps, and external Web services. Two examples of apps that use external Web services are the translator app, which uses the TranslationShop Web service for automated or human-expert translation of text, and the postcard app, which uses the LetterShop Web service for sending digital and paper postcards to customers.

Due to the high costs involved for creating a private owned infrastructure, the platform provider has decided to use a cloud provider, such as Amazon Web Services, for the provisioning of the CAST Platform. The fact that the CAST Platform uses computational, storage, and networking resources provided by a third party raises concerns about the reliability and performance, but also the reputation and the business value of the platform. Thus, before a Web service becomes available to be used in an app, it has to be registered in the Governance Registry & Repository system. During the registration of a new Web service, the WSDL file of the service and a machine-readable SLA for the service are stored. The SLA is an agreement between the provider of the external service and the platform provider and it specifies the expected response time and availability of the provided Web service.

Due to the high costs involved for creating a private owned infrastructure, the platform provider has decided to use a cloud provider, such as Amazon Web Services, for the provisioning of the CAST Platform. The fact that the CAST Platform uses computational, storage, and networking resources provided by a third party raises concerns about the reliability and performance, but also the reputation and the business value of the platform. Thus, before a Web service becomes available to be used in an app, it has to be registered in the Governance Registry & Repository system. During the registration of a new Web service, the WSDL file of the service and a machine-readable SLA for the service are stored. The SLA is an agreement between the provider of the external service and the platform provider and it specifies the expected response time and availability of the provided Web service.

4.2 SLAs in the CAST Platform

The CAST platform could clearly benefit by being enhanced with CLAM. This would support more appropriate, and more timely checking of SLA violations, and the triggering of more suitable and better-coordinated adaptations in the BPM, SCC, and SI layers, whose effects would therefore be much less likely to interfere with each other across the layers. The platform provider will need to identify the external business, software, and infrastructure services and their characteristics, which could affect the quality of the platform. The characteristics could be used for defining SLOs to be included in SLAs for these services. Table 1 summarises the business, software, and infrastructure services that have been identified in the CAST Platform. The table shows the also the identified characteristics of each service that could affect the quality of the platform.

\(^1\)CAST project website - http://www.cast-project.eu/
Table 1: The results of the analysis show the identified business, software, and infrastructure services and their characteristics identified in the CAST Platform.

<table>
<thead>
<tr>
<th>SBA Layer</th>
<th>Service Type</th>
<th>Services</th>
<th>Service Characteristics</th>
<th>Service-Level Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPM</td>
<td>Business</td>
<td>automated translation of text</td>
<td>automated translations processed per day, average time for an automated translation</td>
<td>automated translations processed per day &gt; 50, average time for an automated translation &lt;= 5min</td>
</tr>
<tr>
<td></td>
<td></td>
<td>manual translation of text</td>
<td>manual translations processed per day, average time for a manual translation</td>
<td>manual translations processed per day &gt; 10, average time for an manual translation &lt;= 1 hour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>send digital postcard</td>
<td>digital postcards sent per day, average delivery time of digital postcards</td>
<td>digital postcards sent per day &gt;= 300, average delivery time of digital postcards &lt;= 15min</td>
</tr>
<tr>
<td></td>
<td></td>
<td>send paper postcard</td>
<td>paper postcards sent per day, average delivery time of paper postcards</td>
<td>paper postcards sent per day &gt;= 50, average delivery time of paper postcards &lt;= 3 days</td>
</tr>
<tr>
<td>SCC</td>
<td>Software</td>
<td>TranslationShop Web service</td>
<td>behavioural conformance, average response time, hourly availability, average error rate per hour</td>
<td>average response time &lt;= 350m, hourly availability &lt;= 98%, average error rate per hour &lt;= 0.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LetterShop Web service</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td>Infrastructure</td>
<td>Amazon EC2/S3</td>
<td>average time to provision a resource, hourly availability of resources, average storage I/O per minute, average storage error rate per minute</td>
<td>average time to provision a resource &lt;= 5min, hourly availability of resources &gt;= 99%, average storage I/O per minute &gt;= 10 million, average storage error rate per minute &lt;= 0.02</td>
</tr>
</tbody>
</table>

In the BPM layer, two business services were identified. Two separate SLAs are required between the platform provider and the two service providers of the TranslationShop and LetterShop services. These services realise automated activities such as machine translation or emailing digital postcards, and manual activities that require a human to perform a task, for example, human-expert translation or posting of paper postcards. Therefore, the SLAs should contain agreements on characteristics of both automated and manual activities. The two SLAs will include agreements on the key performance indicators (KPIs) and process performance metrics (PPM) of the two services. A KPI for both services could be the customer satisfaction, which could be calculated from the ratings provided by end users of the two services. The PPMs for the TranslationShop service could comprise the number of automated and manual translations processed per day, and the average time to complete a machine or manual translation. The PPMs for the LetterShop service could comprise the number of digital or paper postcards sent per day, and the average delivery time for digital or paper postcards.

In the SCC layer, two software services were identified. Two separate SLAs are required between the platform provider and the two service providers of the TranslationShop and LetterShop services. The two SLAs will contain agreements on the technical characteristics of the TranslationShop and LetterShop services. The technical characteristics comprise some functional and non-functional elements of the two external services. The functional characteristics could comprise the behavioural specification of each service, while the non-functional characteristics could comprise the average response time, the hourly availability, and the average error rate per hour.

In the SI layer, one infrastructure service was identified. Only one SLA is required between the platform provider and the Amazon Web Services. This SLA will contain agreements for the characteristics of the infrastructure services provisioned by Amazon. The characteristics of the infrastructure could comprise the average time to provision a resource (e.g. a virtual machine or more storage space), the hourly availability of resources, the average storage I/O per minute, and the average storage error rate per minute.

The identified SLAs for the third party business, software, and infrastructure services utilised in the CAST Platform will be used for implementing CLAM. Existing SLA frameworks for Web services, such as WSLA\(^2\) or WS-Agreement\(^3\), could be employed for representing machine-readable SLAs. Each SLA could include a subset of an adaptation strategy to be used during the generation of the cross-layer adaptation strategy. For instance, a subset of an adaptation strategy for the TranslationShop service in the SLA of the BPM layer could comprise that in the event of a dramatic increase of the average time of manual translation, the manual translation business process in the Translation App will have to be adapted, such that it will use the automated translation function of the TranslationShop service, in order to provide a quick low quality translation, while waiting for the manual translation of better quality to arrive at a latter time. CLAM approaches similar to the efforts in [11], will be re-

\(^2\)http://www.research.ibm.com/wsla/
\(^3\)http://www.gridforum.org/documents/GFD.107.pdf
quired for correlating the observations of the monitoring activities, deciding effective adaptation strategies, which will take into consideration the subsets of adaptation strategies provided in the SLAs, for enacting effective adaptations in the CAST Platform for preventing or responding to SLA violations.

5. CONCLUSION

In this paper, we have suggested that each SBA layer concerns different types of services used in an SBA. Our view is that the business process management (BPM) layer is concerned with business services, the service composition and coordination (SCC) layer is concerned with software services, and the service infrastructure (SI) layer is concerned with infrastructure services. We demonstrated that a CLAM approach requires a clean separation of these types of service, such that different kinds of SLA may be drawn up with different providers in the service chain. We have focused on the definition of SLAs in the BPM, SCC, and SI layers, for cross-layer adaptation and monitoring of SBAs. We have presented insights into how an SBA should be analysed, in order to identify and separate the distinct business, software and infrastructure services. We applied this technique to a new case study, based on an existing platform-as-a-service offering, which was chosen as an example for CLAM. Each of the services identified in the study was then analysed to reveal its individual characteristics, prior to drawing up appropriate SLAs. The study clearly demonstrates the utility of separating the run-time quality assurance concerns at each layer of the SBA.

We are currently investigating existing methods for representing SLAs for the business, software, and infrastructure services. As future work, we plan to extend previous work related to the implementation of an extensible monitoring architecture for Web services, in order to support the development of a CLAM framework for SBAs that will use multiple SLAs for business, software, and infrastructure services.

6. ACKNOWLEDGMENTS

We cordially thank Dr. Iraklis Paraskevakis and Dimitrios Kourtessis from the South-East European Research Centre for providing information concerning the CAST Platform.

We would like to thank the anonymous reviewers for their constructive and detailed comments on improving the paper.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

7. REFERENCES


Negotiation towards Service Level Agreements: A Life Cycle Based Approach

Sajid Ibrahim Hashmi¹, Rafiqul Haque¹, Eric Schmieders³, and Ita Richardson¹
Lero – The Irish Software Engineering Research Centre, University of Limerick, Ireland¹
{sajid.hashmi, Rafiqul.haque, ita.richardson}@lero.ie¹
Paluno - The Ruhr Institute for Software Technology University of Duisburg-Essen²
Gerlingstraße 1645127, Essen, Germany²
eric.schmieders@paluno.uni-due.de²

Abstract—Service Based Systems (SBSs) are composed of loosely coupled services. Different stakeholders in these systems, e.g. service providers, service consumers, and business decision makers, have different types of concerns which may be dissimilar or inconsistent. Service Level Agreements (SLAs) play a major role in ensuring the quality of SBSs. They stipulate the availability, reliability, and quality levels required for an effective interaction between service providers and consumers. It has been noticed that because of having conflicting priorities and concerns, conflicts arise between service providers and service consumers while negotiating over the functionality of potential services. Since these stakeholders are involved with different phases the life cycle, it is really important to take into consideration these life cycle phases for proposing any kind of SLA negotiation methodology.

In this research, we propose a stakeholder negotiation strategy for Service Level Agreements, which is based on prioritizing stakeholder concerns based on their frequency at each phase of the SBS development life cycle. We make use of a Collaxa BPEL Orchestration Server Loan service example to demonstrate the applicability of the proposed approach. In addition, we simulate the negotiation priority values to predict their potential impact on the cost of the SLA negotiation.

Keywords- Service Based Systems (SBS ); Stakeholders; Roles; Development Life Cycle; Service Level Agreement (SLA); Quality of Service (QoS); Loan Flow example

I. INTRODUCTION

Service Oriented Architecture (SOA) is having a major impact on the development of software systems because of its potential for increased business agility, adaptability of applications, interoperability between systems, and reuse of legacy assets [1]. SBSs involve different activities, e.g., requirements management, adaptation and monitoring, rules, policies, testing, and management. Generally, there are three key types of stakeholders involved in SOA [17]: the customer, the business or sales organization, and the operations or delivery organization.

These stakeholders are assigned different roles, for example creation and maintenance of development activities and work products is the responsibility of different internal stakeholders, e.g., analysts, architects, developers, testers, and managers [2]. Due to the distributed and loosely coupled nature of the constituent services, external stakeholders also come into play directly in the form of service consumers, as they are the potential users after the services have been developed and deployed. Therefore, an SBS life cycle has to support the involvement and presence of these different types of stakeholders, and, the development life cycle must incorporate stakeholders’ concern [3]. Normally, an SLA is used to develop a formal contract between the service provider and the service consumer. It is usually formed either through adoption of agreement from the provider, or by negotiation between the two. Their main purpose is to determine whether predefined characteristics and quality attributes of services are met [20]. Negotiation is carried out between the service provider and the consumer before any kind of agreements can be met. This negotiation is likely to raise conflicts because of difference in Quality of Service (QoS) priorities. It is really important to mitigate these conflicts so that SBS stakeholders, who contribute towards the business value, can mutually agree upon an SLA.

Opposing concerns of stakeholders on the provider as well as on the consumer side, across different phases of the life cycle, may raise conflicts on negotiating QoS capabilities (such as response time). This raises a number of questions, which specific stakeholders are involved in this conflicting situation? What life cycle phases and activities are involved in potential QoS capabilities whose negotiation leads to the occurrence of these conflicts? How can we methodically resolve these conflicts by assigning priority to a certain group of stakeholders who are working towards SLAs? How can we observe the potential effect of assigned priority on QoS capabilities of the potential service, e.g. response time?

Recent research in the area has not focused on stakeholders at different phases of the life cycle and their potential role in the negotiation process which eventually leads to SLAs. As they are the most common mechanism used to establish agreements on the quality of service between the service provider and the service consumer, the importance of negotiation cannot be underestimated [8]. In addition, it is important to take the stakeholders into account considering that SBSs are developed, owned, and used by different stakeholders with different perspectives, i.e. developer and provider, broker and composer, and consumer and end user respectively.
In this paper, we identify stakeholders and roles associated with them based on their key responsibilities at each phase of the SBS development life cycle. Then, we identify how conflicts may occur between the service provider and the consumer while negotiating towards SLA. An example scenario is presented using the Collaxa BPEL Loan Flow Service to demonstrate the potential conflicts which can occur between the stakeholders. We prioritize the stakeholders in the negotiation process based on the frequency of their involvement at each phase of the life cycle. We use a simulator to predict stakeholders’ impact on cost of the SLA negotiation by means of assigning them priority rotating.

The remainder of the paper is organized as follows: Section 2 describes our Research Methodology. Section 3 consists of related work and Section 4 contains background information on our research project and Service Level Agreements. Section 5 describes our contribution in terms of identification of stakeholders, the study design with an example scenario, mapping stakeholder roles at each phase of the SBS life cycle, and the simulator. Finally we discuss Conclusions and Future Work in Section 6.

II. RELATED WORK

In order to conduct this research, our literature review had to focus on three topics: the interaction of stakeholders in accordance with life cycle phases, negotiation on SLAs, and the role of development life cycle and corresponding stakeholders in this negotiation process.

Conflict negotiation has been addressed for both conventional [23] as well as for service based systems [24][6]. In addition, SLAs have been a focus of researchers. Some negotiation based conflict resolution methodologies have been proposed [5][6][7][8][9][10], but in general, their focus has been on run time adaptation and composition of services only. Studies involving stakeholders and their roles have mostly focused on the requirements engineering phase of the traditional software development life cycle [4][5][6][7]. There are some methodologies for identifying the total number of stakeholders which make it is easier to identify the project critical stakeholders [25][26]. In addition, there has been some work on the interaction among stakeholders [10][11][12][13]. But since these related works do not take into account involvement of different life cycle phases, they cannot be applied to the case of SBSs. [21] has proposed a method for conflict resolution but the context is limited to conventional software systems only.

In terms of web services, there is some research on SLA management. For example, [14] has proposed a methodology on SLA negotiation but it is among anonymous service providers and the consumers, and does not take into account the situation in which a provider and consumer are already tied into a business link. In addition, a contract Net Protocol tool [10] has been developed which sends the contract request information to potential services. The corresponding services bid on those options. The user then selects a specific supplier, rejecting all other offers. The scope of our research is somehow different. We focus on a Business-to-Business (B2B) scenario in which a provider-consumer relation already exists between the two and the goal is to customize the negotiation over a potential service provision. In short, in our review we have not identified methodology addressing the stakeholder negotiation by incorporating the SBS life cycle phases, before Service Level Agreements can be made.

Our literature review suggests that the use of stakeholder roles, taking into account life cycle phases, and SLA negotiation has been mutually exclusive in the existing research. Moreover, researchers to date have not studied the potential impact of their proposed SLA negotiation methodologies on the cost associated with Service Level Agreements. Therefore this paper addressed this gap.

III. RESEARCH METHODOLOGY

Figure 1 represents an overview of the research methodology. Our research question is How can we make use of the phases in an SBS development life cycle phases and stakeholders involved in them to facilitate negotiation towards SLAs. The plan is to thoroughly investigate the steps and stakeholders involved in development as well as adaptation of web services in service based systems. We target negotiation between two types of stakeholders, service providers and service consumers.

We carried out a literature review to understand the phases involved in the development of SBS. We then mapped them to the S-Cube life cycle [17] which ensured Development as well as Adaptation phases of the SBS development. A quantitative stakeholder identification template is used to identify relevant stakeholders of the project, and the corresponding life cycle phases where their activities resided. Each participant was advised to answer the questionnaire according to his interest and the best of knowledge considering in mind the generic activities performed at each phase of the life cycle. The questions were selected specifically after the literature review to facilitate the proof of the research question. The template included 10, mostly close-ended, questions. It was distributed among the 15 project partner universities across Europe to identify the potential stakeholders. We mapped these stakeholders to the S-Cube life-cycle (Figure 2). Once stakeholders’ types and count were identified, we distributed them into two different roles, such that:

Total number of Stakeholder Roles (\(T_R\)) = \(C_R + P_R\)

Where  \(C_R = \sum\) (All stakeholder roles which are involved in consuming services, or in making use of them, e.g., application clients and end users. \(P_R = \sum\) (All stakeholders roles which are responsible for providing these services, e.g. application builders and service composers. We used a Loan Flow service example to identify a scenario which may lead to conflicts on QoS between service provider and the service consumer. The identified conflicting node
A reference lifecycle for service based systems has been developed by S-Cube project researchers (Figure 2). It is composed of two cycles. The evolution cycle depicts classical application design while the adaptation cycle reflects the adaptation of the SBSs. These systems need to accommodate many changes at run time. The service life cycle model envisioned by the S-Cube framework captures an iterative and continuous method for developing, implementing, and maintaining services, in which the feedback is continuously cycled to and from phases in iterative steps of refinement and adaptations of all three layers of the technology stack [4]. The method accommodates continuous modifications of service based systems and its quality (e.g. QoS) at all layers. Once service based systems (or parts thereof) have been adapted, they will be redeployed and re-provisioned and put into adaptation.

B. Service Level Agreement (SLA)

An SLA addresses an agreement between a service provider and its consumer, can be between two parties as well as among multiple ones [20][15], and becomes valid after it has been signed by the contracting parties [5]. No customization occurs in these contracts, the possibilities for service requirement-capability matching are severely limited and negotiation is carried out through meetings between the client, the service and legal aid. Since the negotiation process is continuous, it emphasizes the importance of involvement of stakeholders at different phases of the life cycle. Normally, SLAs are defined using predefined templates in plain text, which makes them open to inconsistencies and conflicts. An SLA may have the following information [12]: pre specified elements, fixed Information, negotiable elements, and their Choices (choices of parties, or, choice of the elements).

These are the negotiable elements which are likely to give rise to conflicts as opinions of service providers and service consumers may differ. An SLA is authored once an agreement has been made between the Service Consumer and the service Provider. This authoring process can be
offline, where the information is exchanged between the parties via e-mail or other human communication mechanisms.

The SLA is defined via WSLA (Web Service Level Agreement) language. For flexibility, certain terms of the contract can be negotiated [7][15]. Importance of negotiation in even increased because any type of WSLA authoring tool uses an SLA template to present graphically various input fields and choices to be made by an author [7], but information on this template is not finalized until both parties agree on QoS parameters of the potential service. The provision of the service depends on SLA and negotiation plays an important role in the formation of these agreements because the life cycle of an SLA starts with its negotiation [7].

Conflicts which arise during the SLA negotiation are likely to be overcome either by going for an alternative service provider, or by renegotiation among stakeholders. The former one may not be a good idea as it could involve more overheads in terms of looking for a new provider and finalizing the agreement with it. Besides, negotiating with existing provider regarding newly evolved requirements is a non-trivial task because both the service provider and the service consumer can take advantage of the existing business link between them, which would preclude the service consumer changing its provider. This is exactly the situation we are targeting in this research.

V. STUDY DESIGN

A. Identification of Stakeholder Roles

Two major categories of stakeholders were identified during previous research conducted in the S-Cube project [17]:

- Consumers and users of service based applications including experience and inexperienced end users
- Service composers and users involved in the system design, such as software engineers, system integrators and architects, business experts

SEI’s Capability Maturity Model Integration (CMMI) [18] specifically proposes that stakeholders be selected among customers, final users, developers, producers, test staff, suppliers, marketing staff, maintenance staff, and anyone who may affect or be affected by the software process and the final product. The stakeholder identification from S-Cube fulfills these criteria. Another reason for using the term stakeholder is that it is more generic and can involve different types of users within it [17]. Figure 3 shows the stakeholder identification template [17] we used to identify stakeholders at each phase of the SBS life cycle. The Service Providers category is the one who is the owner of the service(s) and develops and publishes them. Service composer composes existing services for achieving certain business goals. The application builder integrates services into systems which fulfills user requirements. The application client is an end user who uses the service based applications to achieve certain goals. Supporting roles refers to a category of stakeholders who is not directly involved with the service life cycle (or we can say that it is one of the internal stakeholders), e.g. project managers or service legal advisers [17]. These different types of stakeholders are involved at each phase of the life cycle in order to perform their respective activities. Each type of these stakeholders has a value and dependency associated with each phase of the life cycle.

B. Example Conflicting Scenario

The example scenario we used to elaborate our conflict example is a Loan Flow service was provided by Collaxa [9]. The functionality of the system requires that a user enters a social security number as an input into the system. The system returns an integer number as credit rating by means of Check Credit Rating service. One loan service LoanApp is responsible for receiving loan application documents. The other service StartLoan is responsible for returning the loan offer documents.

As part of our example scenario, we assume that a service consumer is negotiating with a service provider over the provision of an SBS (Figure 4). In addition, the figure shows the conflicted node using shaded rounded rectangle box. The conflicting situation triggers when the service consumer demands the system to check for the user’s credit rating in 5 seconds but the provider cannot make that available in less than 10 seconds. It is important to identify and negotiate these conflicts as they are likely to propagate among stakeholders, across different phases. For instance, the conflict between the service end-user and the service broker influences negatively the agreement between service broker and the credit rating service.
Figure 4. Example Conflict in the Loan Flow Example

Figure 5 highlights the QoS requirements constraints filled in by the service consumer, in terms of efficiency and availability. It wants the credit rating to be calculated in no more than 30 seconds, response time should be 10 seconds, and service must be available on June 1st 2011 from 08:00 to 20:00 to December 31st 2011 (Figure 5). The service provider agrees with these QoS constraints except the service response time.

As part of our solution, we investigate the involvement of types and numbers of stakeholders on this conflicting node Check Credit Rating. In order to do so, we consider the life cycle phases in the development of the SBS system depicted in Figure 4, which consequently would give us the types and number of stakeholders involved at each of its phases. Table I shows the relevant information in this regard. In the third column from left, we have listed the potential phases from the SBS life cycle, for the development and adaptation of the Check Credit Rating service. The notable thing is that couple of phases like Deployment & Provisioning and Identify Adaptation Strategy are not listed. The reason is that these two phases does not seem to be specifically involved in the service development and adaptation of the service, as service deployment and adaptation strategy is not confined solely to a single service only. Also, these phases do not seem to have anything to do with the negotiation on service QoS requirements and its functionality.

The next column lists the types of stakeholder roles involved in the conflicting Node. These stakeholders have been categorized into two main roles, service provider and service consumer. The corresponding number of each type of stakeholder role is counted and categorized into their respective category.

This stakeholder frequency information is used to help us prioritize and weigh the stakes of corresponding stakeholders in the SLA negotiation process. Using probability theory we can provide a quantitative description of the likely occurrence of a particular event, and is conventionally expressed on a scale from 0 to 1. We can also calculate the probability of an event by calculating its relevant frequency [14] (which is obtained by dividing the number of times an event occurs by the total number of times an experiment occurs). This means that the relative frequency of a stakeholder role can be calculated by dividing its individual occurrences by the total number of occurrences in that phase. We have considered relevant frequency as the probability of occurrence of a stakeholder role in each phase of the life cycle. These relevant frequency values are considered as a priority mechanism in the negotiation process for that specific stakeholder role; i.e. if relevant frequency value is greater than its counterpart, that specific stakeholder role is assumed to have more preference in the negotiation process. Table II shows the probabilistic values of each stakeholder role based on its relevant frequency at each phase.

TABLE I. INVOLVEMENT OF PHASES & STAKEHOLDERS IN LOAN FLOW EXAMPLE

<table>
<thead>
<tr>
<th>Reason of Conflict</th>
<th>SBS Life Cycle phases involved</th>
<th>Stakeholder Roles Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Credit Rating</td>
<td>Non-functional requirement (Service Response time)</td>
<td>Requirements Engineering and Design, Construction, Operation and Management, Identify Adaptation needs, Enact Adaptation</td>
</tr>
</tbody>
</table>

TABLE II. RELEVANT FREQUENCY OF THE STAKEHOLDER ROLES AT EACH PHASE OF THE SBS LIFE CYCLE

<table>
<thead>
<tr>
<th>Phases</th>
<th>Stakeholder Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Application Builders</td>
</tr>
<tr>
<td>Requirement Engineering &amp; Design</td>
<td>0.61</td>
</tr>
<tr>
<td>Construction</td>
<td>0.38</td>
</tr>
<tr>
<td>Operation and Management</td>
<td>0.2</td>
</tr>
<tr>
<td>Deployment &amp; Provisioning</td>
<td>0.36</td>
</tr>
<tr>
<td>Identify Adaptation Needs</td>
<td>0.45</td>
</tr>
</tbody>
</table>

SLA_C = (30s, 10s, (20110601080000),(20111231200000))

SLA_P = (30s, 30s, (20110601080000),(20111231200000))
These relevant frequency values are calculated using the total frequency of the different roles each type has in that specific life cycle phase. These values are further summed up into two main categories, i.e., service provider and service consumer (Table III). We consider Application Builders and Service Composer roles to reinforce service provision. Application Clients seem to be the only consumer role involved at each phase, but we count on Supporting Roles towards the service consumers because they may serve as managers as well as technology consultants and legal officers [13]. The involvement of each stakeholder is influenced by the collective sum of its corresponding roles at each phase of the life cycle, thus providing the relevant frequency.

<table>
<thead>
<tr>
<th>Phases</th>
<th>Stakeholder Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement Engineering &amp; Design</td>
<td>CP</td>
</tr>
<tr>
<td>Construction</td>
<td>0.77</td>
</tr>
<tr>
<td>Operation and Management</td>
<td>0.88</td>
</tr>
<tr>
<td>Deployment and Provisioning</td>
<td>0.7</td>
</tr>
<tr>
<td>Identify Adaptation Needs</td>
<td>0.81</td>
</tr>
</tbody>
</table>

C. Simulator

Cost is one of the measurable qualities of SLAs as defined in [19]. We have built a simulator to validate our approach by observing the trend in this variable in response to the assigning priorities to both types of stakeholder roles, one by one. Studies show that one way of measuring cost is to quantify it in terms of stakeholders’ fulfillment [14]. So in our case, it is measured as dissatisfaction between the two types of stakeholder roles, i.e., the provider and the consumer, such that high cost between both types of stakeholder roles would imply lower satisfaction, and the vice versa. Table IV shows equations of our simulation model.

While building our model, we follow a systematic approach as outlined by [22]. Also, our formulation of the problem is quite clear, as we want to observe the trend in SLA cost by investigating the effect it undergoes in response to assigning priority to both stakeholder roles in turn.

TABLE IV. COST EQUATIONS FOR SIMULATION

<table>
<thead>
<tr>
<th>Total Cost of all stakeholder roles(Cp)</th>
<th>Total Cost of service provider roles (Cp) when assigned priority Prcp</th>
<th>Total Cost of service provider roles (Cp) when not assigned priority Prcp</th>
<th>Total Cost of service consumer roles (Cc) when assigned priority Prcp</th>
<th>Total Cost of service consumer roles (Cc) when not assigned priority Prcp</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
<td>(Rc x PrCP) / Rcp</td>
<td>(Rc x PrCP) / Rcp</td>
<td>(Rp x PrCP) / Rp</td>
<td>(Rp x PrCP) / Rp</td>
</tr>
</tbody>
</table>

Where

- \( CP \) : total cost of all roles (consumer and provider)
- \( C_C \) : cost of service consumer roles
- \( C_P \) : cost of service provider roles
- \( Pr_c \) : priority of service consumer roles
- \( Pr_{cp} \) : priority of service provider roles
- \( R_f \) : total relative frequency at each phase of the life cycle
- \( R_p \) : relative frequency of provider roles at each phase
- \( R_c \) : relative frequency of consumer roles at each phase
- \( R_{fp} \) : relevant frequency of a provider role at the \( i \)th life cycle phase
- \( R_{fp} \) : relevant frequency of a consumer role at the \( i \)th life cycle phase

As far as the availability of quantitative data is concerned, it has already been made available (Tables II & III). Finally, the results are represented in the form of graph plots to make the findings more conducive.

By analyzing Table IV, equations for assigning and negating priority to a stakeholder role appear to be the same. But their outcomes are different as different priority values for the same stakeholder role, results into different outcomes. We use the priority scheme used by [14] to associate high and low priority values with both cases. Table V contains these value settings, in the form of priority values for both cases. First, when priority is assigned to service provider roles; second, when priority is assigned to service consumer roles. The order of life cycle phases is same as shown in Table III.

TABLE V. PRIORITY VALUES FOR BOTH HIGH & LOW CASES

<table>
<thead>
<tr>
<th>Stakeholder Role</th>
<th>High Priority</th>
<th>Low Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Provider</td>
<td>( Pr_p = 10 )</td>
<td>( Pr_p = 5 )</td>
</tr>
<tr>
<td>Service Consumer</td>
<td>( Pr_c = 10 )</td>
<td>( Pr_c = 5 )</td>
</tr>
</tbody>
</table>

The values in Table VI are obtained by inserting the relative frequency values into the corresponding simulation cost equations, using both high and low priority values (Table V) for each type of stakeholder roles.

One way of validating a simulation model could be the demonstration of difference between previously existing research works and the proposed approach [14], but currently no life cycle based SLA negotiation approach exists. So, our validation of the model consists of the investigation of both types of cost results; that is, effect on cost when each one of service providers and service consumers roles are given preference in the negotiation process, turn by turn. Also, we demonstrate both types of results using graphs, which is another form of credible validation approach for a simulation model [22].
Also an output data which closely resembles to the potential actual outcome should be considered as another validity test for the model [22]. For example, in our case, assigning preference to the service provider roles is likely to reduce the cost, and the graphical demonstration of the output of our model confirms this likely result.

**First Case:** Figure 6 shows the comparison of SLA cost with higher preference given to the service provider. Using simulation cost equations, when priority is assigned to service provider roles, which is the likely case (on the basis of higher relevant frequency values, Table III), the cost of SLA negotiation for the case is far less and stable as compared to service consumers'. Also, the graphical curve of the latter is quite abrupt and unstable, which denotes that cost of assigning priority to service consumer roles will be far higher and inconsistent across different phases of the life cycle.

![Figure 6. Cost curves with high preference on service provider roles](image)

Second Case: Figure 7 shows the comparison of SLA cost with higher preference given to the service consumer. Using simulation cost equations, when priority is assigned to service consumer roles, (which is the unlikely case on the basis of lower relevant frequency values, Table III), the maximum cost of SLA negotiation for the case is even higher, with somehow similar abruptness and instability as identified in the previous case. But in comparison, the cost of SLA negotiation is still far less stable than the service provider roles. This trend in the graph testifies to the fact that trying to assign priority to service consumer roles will not serve the purpose, considering the fact that their relevant frequency is too small.

![Figure 7. Cost curves with high preference on service consumer roles](image)

**Discussing threats to validity:** Our simulation comprises a small set of input and output variables. We described the straight forward algorithm exploiting stakeholder frequencies and stakeholder priorities in order to compute the output values, i.e. the cost-values. Based on the clear relation between the input and output variables we can exclude side effects and unfolded dependencies, thus excluding threats to the internal validity of the experimental results. When discussing the external validity it must be noted, that a simulation model can only be an approximation to the actual system, as a model, an abstraction of a potential real world situation. Moreover, validation of the simulation model itself remains as another challenge but as discussed by [22], a few factors help to establish credibility for a model. One of them is understanding and agreeing with the model’s assumptions, while the other one is ownership and involvement with the task. We fulfilled both of these criteria while building our simulation model. Furthermore the acceptance is influenced by the applicability of the presented approach to similar settings. Forecasting based on different proportions of the stakeholder involvements could be useful for similar scenarios which fall within its boundaries. To support this, our work uses an external, commonly used service for the simulation, by applying common standards. As our work is reproducible and applicable to a broader area of similar problems, threats to external validity are considerably lower. Finally, to work against threats to construct validity, the simulation model was continuously tested during constitution. Additionally, the simulation results were checked against independently computed results.

**VI. CONCLUSIONS AND FUTURE WORK**

It is important to understand and implement a good negotiation process as it leads towards a formal agreement between the two parties in the form of the SLA. We have proposed a life cycle based methodology for negotiation between service providers and service consumers. The first phase of this negotiation scheme is to identify the type and number of stakeholder roles involved in the SBS life cycle. This information helped us to calculate their relevant frequencies at each phase, which in fact was used as a corresponding ratio of their participation in the negotiation process. Based on this information, an example conflicting situation was analyzed using the potential number and types of the involved stakeholders. Finally, the identified stakeholder information was simulated to successfully predict the SLA cost associated with assigning priorities to both types of stakeholders; service providers and service consumers.

The distribution of our number of stakeholder roles may look partial, for example, the service provider seem to have more roles associated with, thus increasing its priority in the negotiation process. Using the stakeholder role information we gathered across different SBS development life cycle phases is shown in the example.
The information may differ across different environments but the basic theme for assigning priority to any of the stakeholder involved in the negotiation process remains the same. It should be noted that we did not include service consumers as application users as it is rather impossible to predict the exact number of the potential service users. Also, exclusion of the stakeholder role information from the two phases Deployment & Provisioning and Identify Adaptation Strategy would have changed the results slightly, but the main purpose of this research is to demonstrate the usefulness of the approach.

In our future work, we plan to implement the proposed approach with automated contract negotiation. This will allow us to measure cost as well as other quality attributes associated with electronic contract negotiation. Furthermore, we are planning to expand the present set of equations used to compute the output values. The equations could be refined based on case studies. This should increase the approximation of the simulation to the simulated system thus increasing the fine grainness of the simulation results.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). It was supported, in part, by Science Foundation Ireland grant 03/CE2/I303.1 to Lero – the Irish Software Engineering Research Centre (www.lero.ie), and a Higher Education Authority grant PRTLI 4 to the Lero Graduate School in Software Engineering.

REFERENCES


Abstract—Run-time adaptability is a key feature of dynamic business environments, where the processes need to be constantly refined and restructured to deal with exceptional situations and changing requirements. The execution of such a system results in a set of adapted process variants instantiated on the same process model but dynamically restructured to handle specific contexts. Process evolution exploits the information on process variants to identify the best performing recurrent adaptations and adopt them as general solutions in the process model. However, process variants are strictly related to specific execution contexts and cannot be adopted as general solutions. We propose a framework supporting context-aware evolution of business processes based on process instance execution and adaptation history. Instead of looking for recurring adaptations, we propose to look for recurring adaptation needs (i.e., process instances with the same context constraint violation and system configuration). Based on the analysis of adapted instances, we automatically construct and rank corrective evolution variants which can handle the problematic context. At the same time, we try to identify preventive evolution variants by constructing process variants which can prevent the adaptation need. We demonstrate the benefits of our approach using a car logistics scenario.

Keywords—Business Processes, Evolution, Context-awareness

I. INTRODUCTION

Adaptability is recognized as a key feature of dynamic business environments, where operational excellence requires to continuously re-structure business processes to adapt to changes in the execution context [1]. These adaptation needs may be triggered by specific cases to be handled, unexpected situations depending on environmental conditions, or changing requirements.

This need for continuous adaptation results in a system characterized by a huge set of process executions that, although instantiated on the same process model, strongly differ in terms of process structure. In such a dynamic environment, the process models cannot remain unchanged; the short-term adaptations applied to process instances should be used to derive long-term changes in the process models. Providing support for process model evolution is becoming one of the main requirements for managing the lifecycle of dynamic processes [2]. In particular, the set of adapted process instances together with the information concerning their execution should be used as training cases for evolution mechanisms in order to progressively improve process models that are then used to instantiate future process instances.

Most existing approaches addressing this problem (e.g., [3], [4]) derive model-level changes by analyzing frequently occurring changes at the instance-level. In other words, if an instance-level change/adaptation occurs more frequently then a predefined threshold, the change will be propagated at the model-level. These evolution approaches present two major drawbacks. First, an instance-level adaptation variant is not good "in general", it is good for a specific context/situation, and thus cannot simply be propagated to the process model without taking into account the adaptation need it was devised for. Moreover, plugging-in adaptation variants in the original process model is not always a good solution, since it may result in embedding fault-handling activities rather than trying to solve the problem that required runtime adaptation.

To overcome these limitations we present a context-aware evolution framework that, instead of searching for recurring process changes, searches for recurring adaptation needs. At run-time, we may determine that a certain context which at design-time was assumed to occur rarely, actually occurs for a high percentage of the process instances. In this situation, we analyze the instance-level adaptions that have been used for handling the unexpected context. Based on these adaptations, we determine the changes that must be performed on the process model, in order to handle the new context. On the basis of the analysis results, the framework automatically proposes process variants that either embed general corrective solutions derived from context-specific adaptations, or restructure the original model to prevent the recurring adaptation need from occurring.

The enactment of context-aware evolution requires a process modeling notation that on the one hand allows to model contextual information that are relevant to the process execution, and on the other hand allows to easily embed adaptation variants within the process structure. As a reference model for adaptive pervasive systems, we adopt the approach described in the ALLOW project [1], which focuses on modeling and building adaptable pervasive systems. In particular, we model processes as Adaptable Pervasive Flows (APFs) [5], [6], [7], an extension of traditional workflow concepts [8] which make them more suited for adaptation and execution in dynamic environments.

The paper is structured as follows. We describe the mod-

1ALLOW Project, http://www.allow-project.eu/.
eling artifacts in our framework using a car logistics scenario in Section II. In Section III we introduce the evolution framework, focusing on the three main phases of the evolution lifecycle: execution, analysis, and evolution. For each lifecycle phase, we use our car logistics scenario to illustrate the new concepts. We present the related work in Section IV, followed by conclusions and future work in Section V.

II. BACKGROUND

In this section, we describe the modeling artifacts of our evolution framework, as well as their lifecycle (Figure 1).

To illustrate the role of these artifacts, we will use a car logistics scenario. We consider the concrete case of the automobile terminal of Bremerhaven. In our scenario, cars arrive at the seaport and are initially added on stock and stored. Once a delivery order for a certain car is received, the car is removed from stock and prepared according to a treatment list specified in the delivery order. The treatment list may include treatments such as installation of special equipment, paintwork, washing. When the treatment is complete, the car is delivered to the corresponding dealer or directly to the customer.

In a dynamic business environment, the role of the context is fundamental in realizing the adaptation activities [9] as it enables identifying when the process adaptation needed and what should be done. At design-time, along with the definition of a business process the business context is modeled by the Domain Expert as a set of context properties. A context property represents some important characteristic of the environment that can change over time. The Domain Expert model the evolution of a context property with a context property diagram, which is a state transition system. Here states correspond to possible configuration of a property and transitions stand for possible property evolution. Each transition is labeled with an event that characterizes the changes. It is important to note that a context property may evolve as an effect of the business process execution, which corresponds to the "normal" behavior of the domain, but also as a result of volatile - "unexpected" - changes. For this reason we decide to model and manage the context as a separate concern respect to the business processes. For a complete formal notation for context properties please refer to [10]. From the vast domain knowledge in the car logistics domain, we will focus on the handling of cars and their context properties. We consider a subset of these properties: the health status of the surface of the car, the location of the car on the seaport, and whether the car is covered or not. These context properties are displayed in Figure 2. The transitions on events can be guarded. For example, in the carSurfaceHealth the transitions on minorDamage are possible only if the carStatus is in configuration uncovered.

Here, we assume that business processes are modeled using Adaptable Pervasive Flows (APFs) [5], [6], [7]. APFs are business processes, consisting of activities and an execution order specified using control elements. The advantage of using APFs is that they include special modeling elements from the pervasive domain, such as context events and human interaction activities. Further, activities in an APF can be related to context properties, a relation which is realized through preconditions, effects, and context constraints. By using preconditions, we can require context properties to be in certain specified configurations. Context constraints are similar to preconditions, except that they can be defined on a scope consisting of more than one activity. Finally, by using effects, we can specify that the execution of an activity changes certain context properties.

We implement the sequence of steps that the car undergoes on the terminal as a business process (Fig. 3) inspired by [11]. Activities in the process model include unloading the car from the ship, inspecting it for possible transport damages, storing it, applying treatment procedures, and delivering it to the customer.

The activities in the process model are related to the car context properties through a context constraint. Here, we use the notation $s^*(p)$ to denote the fact that the context property $p$ is in configuration $s$, and $e^*(p)$ the fact that event $e$ has occurred for property $p$. The constraint then requires that the carSurfaceHealth is in state ok from the moment it is inspected and until it is delivered to the customer. In case the car surface gets damaged while executing one of the activities in the scope of the context constraint, the process instance will be adapted dynamically.

Our process models have associated goals. The goal associated to the car process model is to have the car delivered to the client in perfect condition. If this is not possible because the surface of the car got damaged and we cannot repair it on site, we at least want the car to be brought to a disposal area, from which it can be returned to the manufacturer or disposed:

- primary goal: $ok^*(sh) \land atDestination^*(loc)$
- recovery goal: $unrepairableLocally^*(sh) \land disposalArea^*(loc)$

This intuitive description can be formally modeled using the language for expressing goals with preferences introduced in [10].

A third artifact in our application is a list of Key Performance Indicators (KPIs), which are defined at design-time by a Business Analyst. These KPIs can be used to evaluate the performance of process instances.

In our scenario, we consider the following list of KPIs:

- **KPI: Percentage of cars delivered on time**
  - Measurement: We consider all the car process instances for which the execution log contains an expected delivery time (if there are multiple values, then we consider the last), as well as the time the execution of the process instance was finished. On these process instances, we compute the percentage for which the finishing time was before or at the expected delivery time.
  - Target: Above 95%
- **KPI: Waiting time at technical treatment stations**
  - Measurement: Total time spent by a vehicle in queues, parking spaces, or service car ramps, while waiting to be
processed.
Target: Reduce the waiting time at technical treatment stations to at most 10 hours per vehicle.
- KPI: Average number of on-site surface damages
  Measurement: For each car process instance, we count the number of times the events minorDamage and severeDamage occur in the associated carSurfaceHealth property. The damages are considered to be on-site only if they occur after executing the Check Car activity. Target: Below 0.003
- KPI: Percentage of repaired car surface damages
  Measurement: We consider all the process instances for which carSurfaceHealth was logged during the execution of the instance as being in one of the damaged configurations. We measure the percentage for which the carSurfaceHealth was in configuration ok upon completion of the instance execution (i.e., a car surface is considered repaired only if it was ok at delivery time). Target: Above 95%

At run-time, both process models and context properties are instantiated. Process instances are then executed and possibly adapted. The decision to adapt a process instance is based on contextual information provided by the instantiated context properties, which are continuously monitored. For example, process adaptation can be initiated when the observed values of the context properties violate some constraint.

The set of adapted process instances together with the information concerning their success and their execution context will be used as training cases for the Performance Analysis in order to progressively improve the process models with respect to the KPIs. The results of this analysis can be used to generate a set of evolution variants which optimize the existing process model with respect to the KPIs. The evolution variants are then proposed to the process designer. In case the process designer decides to evolve the process model, all new process instances will be based on the evolved model.

III. EVOLUTION FRAMEWORK

While in previous Section we introduced the modeling artifacts for evolvable process-based applications, in the following we present in details the proposed framework for context-aware evolution. Figure 4 shows an architectural overview of the framework presenting the relations between the different components and positioning them with respect to the three main phases of the evolution lifecycle, namely (1) execution phase, (2) analysis phase, and (3) evolution phase.

During the first phase, the framework is responsible for managing the execution and adaptation of the system and for logging all the information that may be useful to the other phases (e.g., execution traces, adaptation needs, adaptation variants, execution performances). During the analysis phase, the framework controls and evaluates the quality of execution...
of the processes with respect to the KPIs, decides the need for evolution for a certain process model, and, on the basis of the execution history, identifies the contextual evolution problem in terms of recurring system configuration that required adaptation. Finally, in the evolution phase, the framework uses the information obtained from the analysis phase to compute process model variants that either embed the best performing (with respect to KPIs) adaptation variants or prevent the violation of the context constraint. These evolution variants are then presented to the process designer, who decides whether they should be adopted for future executions. The process designer obtains the evolved process model using a set of supporting tools.

In the following, we present in detail the different phases in the evolution lifecycle.

A. Execution phase

During the execution phase, process models are instantiated and the resulting process instances are executed on the Process Engine. The Execution Manager is responsible for keeping the system configuration up to date and for consistently aligning the status of context properties to the execution of the processes and to the context events received by the Context Manager.

The system configuration is used by the Execution Manager to monitor context constraints associated to running process instances and to trigger adaptation in case of violation. The Adaptation Manager receives in input the description of the adaptation problem (i.e., process instance to be adapted, system configuration, and constraint violation) and computes an adapted process instance that deals with the constraint violation (i.e., bringing the process to a “stable” state that does not violate the constraint, skipping/replacing some original process activities to deal with the context, compensating and terminating the process in case there is no means to deal with the constraint violation).

The information about the system execution and adaptation is recorded in the Execution Log. Examples of stored information are the traces of process instances and of context properties execution and the adaptation history in terms of adaptation problems and adopted adaptation variant.

Consider for example that while executing instances of the process model in Figure 3, the context constraint ok\textsuperscript{sh}(sh) can be violated for some of them. For each of these violations the context configuration is given by the state of the context property diagrams, for example: carSurfaceHealth is in state lightlyDamaged, carLocation in state storageArea, and carStatus in state uncovered. For each problematic process instance, the Adaptation Manager, considering the precise context configuration, can provide different adaptation solutions as the following:

- **Eager repair** (Figure 5a). In this case, the runtime adaptation is to first block the execution of the Receive delivery order activity. We bring the car to the treatment area and repair the damage (problem fixing part), and then return to the storage area (redo part). Finally, we resume waiting for the delivery order.
- **Lazy repair** (Figure 6a). In this case, the adaptation is to complete the execution of the Receive delivery order and add surface damage repair to the list of treatments.
to be performed. We can then proceed to the treatment area and perform the treatments on the list. Here, note the different context constraint. The idea is that until the car is repaired, we need to allow the carSurfaceHealth to be also in the state lightlyDamaged, but we also need to ensure that it is not unrepairableLocally.

B. Analysis phase

The Process Analyzer is responsible for detecting the need for evolution and for identifying the contextual evolution problem.

In particular, the evolution need is triggered by a problem in the system performance with respect to the KPIs of the process models. The evolution need is identified by the Process Analyzer through the analysis of the information stored in the Execution Log. According to the specific techniques used by the Analyzer, we can distinguish between proactive and reactive analysis. In proactive analysis, the Analyzer predicts future KPI violations (see [12]) and evolution techniques can be applied to prevent these violations. Reactive analysis detects the fact that the KPI has already been violated and evolution techniques can be used to avoid future violations.

Given the specific evolution need (KPI violation for a certain process model), the Analyzer considers all the process instances that contributed to the KPI violation and looks for recurring adaptation needs examining their adaptation history in the Execution Log. The retrieved information is used to identify the specific evolution problem which is characterized by the process model to be evolved, the critical KPI, the recurring adaptation need (system configuration and constraint violation), and the history of all adapted process instances.

Consider for example the following evolution need: the violation of the KPI “Average number of on-site damages” for the car process model. The process instances which contributed to the KPI violation are all the instances for which the corresponding carSurfaceHealth reached one of the configurations lightlyDamaged or unrepairableLocally during the execution of the scope of the context constraint ok\(^*(sh)\).

The recurring adaptation need consists of the violation of the context constraint \(ok^*(sh)\) and the following context configuration: carSurfaceHealth in lightlyDamaged, carLocation in storageArea, and carStatus in uncovered. The evolution problem will consist of the car process model, the KPI “Average number of on-site damages”, the recurring adaptation need, and a history of all adapted process instances which had the same adaptation need. In our example, this is a subset of the process instances which contributed to the KPI violation. In general, the adapted instances included in the evolution problem can be also instances which do not contribute to the KPI violation. Such instances are particularly useful in the evolution, since they address the adaptation need and also meet the target for the KPI.

C. Evolution phase

The aim of this phase is to propose a set of process variants that deal with the identified evolution problem and to support

---

**Fig. 4. Evolution Framework**

---
For what concerns the proposal of evolution variants, we distinguish two different approaches: corrective evolution and preventive evolution.

The aim of corrective evolution is to propose a set of adaptation variants that, according to the execution history, can deal with the recurring adaptation need and exhibit good performances. The Evolution Manager looks for adaptation variants that solve the same adaptation problem and that have good performances with respect to the KPI. The identified adaptation variants are then ranked according to execution performances (e.g., performances with respect to the other KPIs and confidence) and proposed as evolution variants to be plugged-in in the original process model.

Figures 5 and 6 show the transformation from an adaptation variant to an evolution variant for the two instance-level adaptations introduced in Section III-A. For both evolution variants, we have used one of the built-in adaptation tools, namely the Context Handler, to handle the critical context. In both cases, if the context constraint $ok^*(sh)$ is violated and the context condition $lightlyDamaged^*(sh) \land storageArea^*(loc)$ is satisfied, the execution of the main process will be blocked, and we proceed by executing the fragment specified in the Context Handler. Upon completion, the control is given back to the main process.

As it can be seen also from the examples, the correspondence between instance-level adaptation variants and evolution variants is not immediate. The challenge here is to understand which of the built-in adaptation tools can be used and how it can be used in order to capture the effects of the instance-level adaptation, and limit these effects to the problematic context.

The adaptation variants can have different performance results. For example, it can be the case that the “Eager repair” strategy performs better than the “Lazy repair” with respect to the KPI “Percentage of cars delivered on time”, since it avoids the delay caused by the extra treatment. However, the “Lazy repair” may perform better with respect to the KPI “Waiting time at technical treatment stations”, since the treatment tasks can be rescheduled considering the availability of the treatment stations. We use these performance results of the adaptation variants, as well as a confidence parameter (the number of instances), to rank the evolution variants.

Preventive evolution proposes evolution variants that prevent the recurring adaptation need from being triggered. This approach is more disruptive since it requires a restructuring of the original process model such that its executions guarantee that the system configuration requiring adaptation is avoided. The idea here is to apply AI planning techniques similar to that of [13], which, given the process model, the context properties, the critical system configuration, and the process goal (in terms of control flow requirements on system configurations) look for a re-planned process model whose traces never reach the critical configuration and yet guarantee to satisfy the process goal. The evolution variant resulting from preventive evolution cannot be analyzed in terms of performances since it consists of a new process model for which no information is available in the logs.

In our example, we can have an activity Apply Cover which requires as precondition: $uncovered^*(cs) \land storageArea^*(loc)$ (the car covers are available only in the storage area), and triggers the effect: $cover^*(cs)$. Similarly, we have also an activity Remove Cover. Then, by replanning the
process model, we will find that the process model displayed in Figure 7 satisfies the process goal and prevents the adaptation need.

The Evolution Enactment module is responsible for obtaining the evolved process model by choosing the evolution variants to be adopted for future executions and for embedding them within the existing process model.

In most application domains thinking of completely automating this phase is unrealistic. As a matter of fact, modifying the process models is often critical and understanding the goodness and impact of the process variants on the system requires a comprehensive knowledge of the domain.

In our framework we adopt a man-in-the-loop approach, where the Evolution Enactment module presents the evolution variants to the process designer and provides her with a set of tools that support the definition of the evolved process. In particular, we exploit the built-in adaptation tools defined in [7], which allow us to embed context-specific adaptation variants within a business process.

In our example, the process designer may find that switching completely to the preventive evolution variant is too costly, since it involves more human resources than are currently available. For regular cars, the best performing correcting evolution variant is sufficient. However, changing to the preventive evolution variant is reasonable in the case of luxury cars, since these are comparably few, and the costs for repairing even minor damages is very high. The process designer can then use the Contextual If built-in adaptation tool to text for luxury cars and integrate the two evolution variants.

IV. RELATED WORK

The problem of supporting the evolution of business processes models has been addressed by several works. Some of them are able to capture a precise sets of contexts and to use for each of them a predefined process variants [14], [15], others support the evolution of processes by analysing previous executions and adaptations [3], [4], [16], [17], [18].

Most existing approaches focus on the problem of extracting useful information from the adaptation logs. The approaches differ both in what they log and in the techniques that they use to analyze the logs (e.g., [3], [4], [17], [18], [19]). These approaches may be used in the analysis phase of the proposed evolution framework.

One direction is to use process mining techniques, as in [4], considering large collections of structurally different process variants created from the same process model. The authors use a heuristic search to find a new process model such that the weighted average distance between the new model and the variants is minimal.

Also using mining techniques to analyze change logs is the approach in [17]. The evolution result of this approach is an abstract change process consisting of change operations and causal relations between them. These change processes can be used as an analysis tool to understand when and why changes were necessary.

A different direction is that of [3]. Here, concepts and methods from case-based reasoning (CBR) are used in order to log, together with the change operations, also the reasons for and context of each change. Change information is stored as cases in a case-base specific to the process model. The case-bases are used to support process actors in reusing information about
similar ad-hoc changes, and are also continuously monitored to automatically derive suggestions for process model changes.

Change analysis and reuse is also relevant for loosely specified process models. [18] facilitates change reuse by providing a search interface for the repository of process variants. For declarative processes, [19] supports users through recommendations, which are generated based on similar past executions and considering certain optimization goals.

The approaches in [3], [4] are closest to our work, since they also generate new process models based on the information from the adaptation logs. A limitation of the approach in [4] is that it does not consider the context requiring adaptation. An instance-level adaptation may be useful only for a particular context, and therefore should not be included in the process model even if it occurs relatively often. Although [3] considers also the context of changes, this is specified as natural language question-answer facts. These are useful for supporting process actors in identifying the existing cases with the same context. They are also useful for the process engineer, who can manually determine the context for a new process model change. However, these tasks cannot be done automatically.

Further, in both [3], [4] the decision whether to integrate a change operation in the process model is determined by the frequency of the change operation. However, it can be the case that the need for adaptation was a fault in the process, and the adaptation contains fault handling and compensating activities. In these cases, rather than including the adaptation in the process model, we may be interested in avoiding the adaptation need altogether.

In contrast to [3], [4], we use the context as the main driver for evolution. This allows us to determine if an instance-level adaptation is useful for a particular context, and it allows us also to search for an alternative process model which avoids a problematic context.

The importance of context for improving process models is recognized also in [16]. Here, the authors propose a context-aware process management cycle, with context-awareness spanning all the stages of the process lifecycle. While [16] remains at a very general and abstract level, we are taking the approach one step further, and provide concrete ideas for implementing the context-aware process management cycle.

V. CONCLUSIONS AND FUTURE WORK

We have presented a framework for evolving process models based on a history of process instance executions and adaptations. Our approach is context-driven. If the need to evolve the process model is detected, we analyze the relevant adapted process instances and look for recurring adaptation needs (i.e., the same constraint violation and system configuration). This allows us to construct and rank evolution variants which can handle the problematic context (corrective evolution). It also allows us to construct evolution variants which can prevent the adaptation need (preventive evolution).

In our future work, we will develop concrete solutions for corrective and preventive evolution. For corrective evolution we plan to develop techniques to automatically transform instance-level adaptation variants into evolution variants using the built-in adaptation tools. For preventive evolution we will apply AI planning techniques to re-plan the process model in order to avoid the critical configurations. We will implement and evaluate our solutions on realistic scenarios, such as the car logistic scenario introduced in this work.

Acknowledgements.

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube) and from the FP7 EU-FET project 213339 ALLOW.

REFERENCES


Constraint-Based Runtime Prediction of SLA Violations in Service Orchestrations

Dragan Ivanović,¹ Manuel Carro,¹,² and Manuel Hermenegildo ¹,²

¹ School of Computer Science, T. University of Madrid (UPM), Spain
{idragan@clip.dia.fi.upm.es, {mcarro, herme}@fi.upm.es}
² IMDEA Software Institute, Spain

Abstract. Service compositions put together loosely-coupled component services to perform more complex, higher level, or cross-organizational tasks in a platform-independent manner. Quality-of-Service (QoS) properties, such as execution time, availability, or cost, are critical for their usability, and permissible boundaries for their values are defined in Service Level Agreements (SLAs). We propose a method whereby constraints that model SLA conformance and violation are derived at any given point of the execution of a service composition. These constraints are generated using the structure of the composition and properties of the component services, which can be either known or empirically measured. Violation of these constraints means that the corresponding scenario is unfeasible, while satisfaction gives values for the constrained variables (start / end times for activities, or number of loop iterations) which make the scenario possible. These results can be used to perform optimized service matching or trigger preventive adaptation or healing.

Keywords: Service Orchestrations, Quality of Service, Service Level Agreements, Monitoring, Prediction, Constraints.

1 Introduction

Service-Oriented Computing is a paradigm that has been increasingly gaining ground as the basis for development of highly flexible, dynamic, and distributed service-based applications (SBAs). Key to the development of SBAs are service compositions, that allow the application designer to put together several loosely-coupled specialized component services, often provided and controlled by third parties, to perform more complex, higher-level, and/or cross-organizational tasks [7]. Trends in service-oriented application design indicate increased reliance on third-party services available on Internet [19].

In that context, quality of service (QoS) properties of individual services and their compositions are critical for overall usability. For externally offered services, service-level agreements (SLAs) define boundaries of permissible values for QoS attributes, such as execution time, availability, or cost. Potential and actual SLA violations can be avoided or mitigated using some form of adaptation (e.g., rebinding or changing

* The research leading to these results has received funding from the European Community’s Seventh Framework Programme under the Network of Excellence S-Cube (Grant Agreement n° 215483). The authors were also partially supported by Spanish MEC project 2008-05624/TIN DOVES and CM project P2009/TIC/1465 (PROMETIDOS).
the service selection preferences), or triggering structural changes both in the design and the running instance [7, 10]. For structurally constrained compositions of non-cyclic shape, flexible provisioning techniques have also been proposed [18].

Therefore, the task of analyzing and predicting QoS metrics for service compositions, both at design time and at the level of an executing instance, is of great theoretical and practical importance. Among the recently proposed approaches we can cite the application of statistical reasoning based on historical data (e.g., data mining) to predict likely SLA violations and their probable causes [15, 23], or to apply techniques related to model checking and online testing [10, 8].

In this paper, we take a different approach based on generating a constraint model for QoS metrics of an executing composition based on its structure, the semantics of its building blocks, and its current state of execution at a given moment. Previous works [4, 3, 13] also used the composition structure as the basis to derive properties thereof. In terms of results, instead of trying to find the most likely SLA conformance or violation scenario, we identify the possible cases of SLA conformance and violation at a given point of execution and infer conditions under which these may occur.

We consider service orchestrations, which are compositions with a centralized control flow. They may involve a wide range of workflow patterns [22] — including parallel flows, different splits/joins, loops, branches, etc. — and are usually expressed using some dedicated notation, such as BPMN [16], BPEL [14], Yawl [20] or DecSerFlow [21], or other adequate formalism. In this paper, we use abstract (but executable) notation for orchestrations from which we formulate a constraint satisfaction problem (CSP) [6, 1] that models the situation of SLA conformance or violation.

The rest of the paper proceeds as follows: Section 2 presents a motivating example. Section 3 then describes how the CSP can be automatically formulated on the basis of an orchestration continuation, to take into account the known assumptions about third-party components, as well as to include internal structural parameters of branches and loops. In Section 4 we present an experimental evaluation, Section 5 gives some implementation notes, and finally Section 6 presents conclusions.

2 Motivation

Consider a scenario where a provider of multimedia content (text, audio and video) needs to periodically update and reconfigure program streams offered to individual
clients (users), based on their historical usage patterns. That may require choosing between different mixtures of available streams (such as news, sport, entertainment, etc.) presented to a user, genres within them, and type of multimedia materials. The choice may depend on the frequency of use (casual vs. frequent users), user interests, and bandwidth adequate to serve different types of content (e.g. low quality vs. HD video). In such a scenario, the provider would run the reconfiguration process from time to time when serving user requests, although typically not for each access. Reconfiguration depends on other (usually back-end) administrative and analytic services, and should not cause noticeable glitches in content delivery. The SLA for the content delivery service does provide some window for running the reconfiguration process on top of it, but it is normally very restricted. Therefore, the running time of the reconfiguration process and its availability are of the utmost importance.

Fig. 1 depicts an example orchestration implementing the reconfiguration process, using BPMN notation [16]. It starts with the reception of user ID (activity $a_0$), which spawns in parallel ($a_1$) the retrieval of the users’ account record ($a_2$) and the user’s usage patterns ($a_3$). If the usage pattern is stable ($a_4$), the user’s current content profile is reused ($a_5$). Otherwise, a new content profile is generated ($a_6$) based on the account record and the current usage patterns. For efficiency, first minor variations in content profile parameters are attempted; if these are not likely to fit the usage pattern ($a_7$), more radical changes are attempted, and so on. Finally, the content profile (either the current one or a new one) is written to the configuration database ($a_8$).

In this example, the configuration process may affect responsiveness of the main multimedia content delivery service, and therefore we want to continuously monitor and predict reconfiguration running time, having in mind the overall SLA. At any point in the execution of the reconfiguration orchestration, including its start, and within that particular context, there are a number of interesting objectives to aim at:

**Predicting Certain SLA Violations:** If we are able to predict that the orchestration cannot possibly meet the SLA constraints, then we can either abort it (effectively postponing the reconfiguration), or adapt it by switching to a simpler and/or more robust version. Conversely, if we are reasonably sure that the execution will be SLA-conformant, we can plan to use the potential slack in a productive way.

**Predicting Possible SLA Violations:** If we can predict that SLA violations may occur, but not necessarily so, and we can identify potential points of failure, then we can prepare, ahead of time, adequate adaptation and healing mechanisms, and/or try to decrease the risk of violation by using fail-safe component services.

**Inferring the Necessary Preconditions:** If we not only predict, but understand why an SLA violation may or must happen, we can use that information to identify bottlenecks, to develop criteria for selection of components, and to drive either runtime or design-time adaptation.

In this paper we present a unified constraint-based approach and analysis framework that makes it possible to perform runtime prediction of SLA violation / conformance for service orchestrations, based on monitoring information and a constraint model of an abstract semantics of the orchestration structure. Predictions are based
on and expressed in a form that describes the circumstances under which SLA violations and conformant executions of an orchestration may take place, which can be used to reason about the orchestration and its components.

3 Constraint-Based QoS Prediction

3.1 The General Prediction Framework

An SLA typically defines, among other things, which QoS attributes are relevant in the context of the provider-client contract, and what values of these QoS attributes are acceptable. For QoS attributes expressed as numbers on a measurement scale, QoS constraints given by an SLA are often expressed as ranges of permissible values for each attribute. More complex relationships between SLA attributes — such as trade-offs between cost and speed — can be devised, but in our analysis we will assume that the QoS constraints are given as lower and upper bounds on appropriate QoS metrics.

Furthermore, we will focus on an important subset of QoS metrics that are monotonic and cumulative in the sense that they express an amount of a physical or logical resource consumed by each activity in an orchestration, so that the amounts from subsequent activities add together into an aggregate metrics. Running time is an obvious example of a cumulative metrics, because consumed time is never recovered. In this paper we will assume, for simplicity, that metrics are accumulated by through addition (which is a fairly common case). Note that some metrics whose natural aggregation function is not addition can be easily mapped into additive metrics. For instance, the aggregation function for the availability (the probability of successful access) $p$ of $n$ subsequent operations can be calculated as $\prod_{i=1}^{n} p_i$, where $0 < p_i \leq 1$ is the availability of the $i$-th component. Using the transformation $\lambda = -\log p$, we can transform the original multiplicative metric of $p$ into the additive $\lambda = \sum_i \lambda_i$.

An important feature of a cumulative QoS metrics is that, at any point in execution of an orchestration, its value can be calculated as a cumulative function (such as addition) of two components: the previously accumulated metrics and an estimate of the pending metrics for the remainder of the execution of the orchestration, until it finishes. For some metrics, their accumulated value needs to be be measured taking into account the history of the actual execution up to the current execution point (e.g. elapsed time from the start of execution), while for other metrics the current value at any execution point does not depend on the previous history. For example, in the case of availability the current metrics always represents “availability so far”. Since it is being measured at some execution point which has obviously been reached, the probability $p$ of being available up to the point of measure is 1 (and then $\lambda = 0$).

Let us present intuitively how accumulated metric values and a prediction for the rest of execution can be applied to predict SLA violations. We will use Fig. 2, taken from [12]. Points $\alpha$-$\beta$ on the $x$-axis stand for the start, finish and two intermediate points in time during the execution of an orchestration. Let us assume that at the initial point $\alpha$ we have a prediction (solid line) for the QoS metrics for the rest of the execution. According to this prediction, the QoS at the finish falls under the limit $Max$ given by some SLA. However, at point $\beta$ we notice that some deviations
have occurred up to that moment (the dashed line). Therefore, we adjust our prediction, which now seems to indicate borderline SLA compliance. At point $C$, further measured deviations lead to another adjustment of the QoS prediction, this time indicating a likely violation of the SLA.

An important aspect of such prediction scheme is the existence of a time horizon between the detection of the possibility of an SLA violation and its actual occurrence. In our example, it is the period between $B$ and the point of failure which lies somewhere between $C$ and $D$. This “window” makes it possible to warn about (potential) future SLA violations ahead of time. A prediction technique also needs to identify conditions that increase or decrease likelihood of an SLA violation, in order to filter false positives from true positives and thus increase the reliability of prediction. These conditions can be related to internal parameters of the orchestrations, such as the truth value of branching conditions or the number iterations in a loop. For our constraint-based approach, this will be illustrated in Section 3.5.

### 3.2 QoS Prediction Architecture

The architecture of the constraint-based QoS prediction framework is shown in Fig. 3. A process engine executes service orchestrations and interacts with external services by exchanging messages. In the process, it publishes lifecycle events such as signaling the start or end of a process, invocation of a component service, and reception of a reply. Also, from time to time, the process engine publishes the current point of execution of a running orchestration in the form of a continuation (explained in the following subsection). That is typically not done at each step, but at specific milestones such as service invocations, loop iterations and branches. Deciding how to determine the optimal granularity for publishing points is a matter for future work.

The events published by the process engine are sent via an event bus. The constraint-based QoS predictor can be connected to that bus and listen to lifecycle events (or a subset of events of interest). When a continuation is published, it is pushed by the event bus to the predictor. The predictor performs the analysis, and publishes QoS predictions back to the event bus, together with QoS metric bounds inferred by the analysis. That information can be accessed by an adaptation mechanism, which
can use the published predictions and the QoS metrics to prepare adequate adaptation actions on the orchestration definition, an executing instance, or both. Such adaptation actions may include, among other things, selection of components to minimize the risk of failure, changes in the structure of the process, or intervention on the orchestration data.

### 3.3 Representing Orchestrations and their Continuations

In order to estimate how much the remainder of the execution can contribute to a given QoS metrics, we need to have some knowledge about where in the execution we are placed — or, more precisely, what remains to be executed: it is the orchestration activities yet to be executed which need to be taken into account to predict the remainder of the metric value. In our case we represent this still-not-executed part of the orchestration explicitly, in the form of a *continuation*. A continuation [17] is an abstract object (such as a set of data structures or a function) that represents the control state of a computation — i.e., the precise execution point of a program (including the associated data) and whatever remains to be executed.

In our case we are interested in continuations of running instances of orchestrations. A continuation is always implicit in the state of a process engine, even when the chosen programming language does not make it accessible as such: it is determined, for example, by the activity being executed, the representation of the orchestration and the data in the orchestration. In our approach, we rely on keeping available at all moments an explicit representation of the continuation, inspect its structure (which in general becomes progressively simpler as execution proceeds) and use it to generate constraints which model the conditions under which the execution can meet / not meet the QoS stated in the SLA.

The (simplified) abstract syntax we will use is shown in Fig. 4. It is based on the concrete syntax used by a prototype orchestration engine which we developed as ex-

---

**Fig. 3.** Architecture of the QoS Analysis Framework.
continuation ::= a.
a, a1, a2 ::= {elementary operation} (elementary operation)
| a1, a2 (sequence)
| ([cond] → a1 ; a2) (if-then-else)
| a1 ∧ a2 (and-join)
| a1 ∨ a2 (or-join)
| while([cond], a) (while loop)
| foreach(x, list, a) (list comprehension)
| invoke(partner, out, in) (invoke a service)
| reply(out) (send a reply)
| relax (do nothing)
| stop (finish)

Fig. 4. Abstract syntax for orchestrations.

performed base for this paper and that uses Prolog as the language to express branch and loop conditions and elementary operations. A simple activity represents a basic unit of work, such as a calculation or assignment. Similarly, cond encodes a logical condition that is used for if-then-else branching or while loop iteration. List comprehension is simplified using foreach. Communication with the environment is done using invoke and reply. Besides sequences, both parallel OR and AND splits/joins are supported. Most BPMN constructs can be translated straightforwardly. A translation of the example process from Fig. 1 (with some low-level details omitted) is shown on the left of Fig. 5.

The continuation at every point of the execution of Fig. 5 is not explicit in the orchestration representation, but is rather kept by the interpreter which executes it (which we do not have space to describe in detail in this paper). This continuation represents what is left to execute after every computation step, and is updated every time a step is taken. For instance, after taking the else branch in the orchestration from Fig. 5 (left), the continuation is a sequence of activities in lines 6-9, 11 and 12.

3.4 Deriving QoS Constraints from Continuations

A constraint is a relation that restricts values of variables that, in our case, represent values of QoS metrics associated with the constructs in the orchestration and their

Fig. 5. Orchestration for Fig. 1 (left) and its associated running time constraints (right).
basic components. The particular relations which are generated depend both on the QoS metric that is to be captured and on the structure of the continuation. In our approach, after deriving the constraints from the structure of the given continuation, constraint solving techniques (see Section 3.6) are used to infer admissible ranges for variables that lead to either SLA satisfaction or violation.

We require that these constraints lead to a conservative prediction of QoS fulfillment: under the assumption that our knowledge about the QoS characteristics of the basic orchestration components (i.e., atomic activities or external services) is correct, we want that any prediction we make about the conformance of an execution w.r.t. the stated SLA is also correct. In this direction, we make no assumptions on the (in)dependence of behavior of individual components. I.e., if the behavior of two external services seems to be strongly linked (because of e.g. past history), we do not take this apparent correlation into account for the sake of prediction safety. Such information, if available, could be added to try to make predictions more precise: for example, given that some service took less time than expected to answer, we might assume that the same is going to happen to some other service which is apparently historically related. While this seems to help in making predictions more accurate, it also makes them potentially unsafe.

We illustrate constraint derivation with two metrics: running time and availability. For a continuation consisting of a (complex) activity \( a \) representing the remainder of the execution, the total running time of the orchestration is a sum of the elapsed time since the start \( T_a \) and the pending time \( T(a) \). The total availability is equal to the pending availability \( \lambda(a) \), as explained before. We derive \( T(a) \) and \( \lambda(a) \) structurally, and then constrain them against the SLA limit: \( T_{\text{max}} \) for the maximal allowed execution time by and \( \lambda_{\text{max}} \) for the negative logarithm of the minimal allowed availability (see Section 3.1). The resulting constraints:

**For SLA conformance:** \( T_a + T(a) \leq T_{\text{max}} \) and \( \lambda(a) \leq \lambda_{\text{max}} \).

**For SLA violation:** \( T_a + T(a) > T_{\text{max}} \) and \( \lambda(a) > \lambda_{\text{max}} \).

are solved to obtain the (approximate, but safe) ranges for \( T(a) \) and \( \lambda(a) \), and thus for the total QoS, for the two cases of conformance and violation, respectively.

We generate the above constraints by formulating a constraint for each simple activity contained inside \( a \) (usually relating the value of the QoS metric for the activity with its expected bounds) and combining these constraints (using disjunctions and conjunctions according to the structure of \( a \)) into a larger constraint which provides bounds for \( T(a) \). The right hand side of Fig. 5 shows the set of constraints corresponding to the process on the left. We will now detail how constraints for simple and complex activities are generated.

**Simple activities.** For a simple activity \( a \) — a simple operation, \texttt{relax} or \texttt{stop} — and simple operations (in curly braces), the assumption is that they include only elementary constructs and do not entail complex computations. A lower bound for

---

3 Note that in reality this knowledge is always inexact and subject to dynamic changes. However, we are putting ourselves in the situation that this knowledge is exact, and we want to ensure that, at least in this optimistic situation, the constraints we generate meet safeness requirements.
this is always $T(a) \geq 0$, and an upper bound depends on the execution environment (computer clock, CPU, etc.). It is usually on the order of microseconds, and should be experimentally determined for each architecture. In the example we have put some reasonable limits, which do not necessarily reflect a real situation. As for the availability, since no external components are involved, in this case we have $\lambda(a) = 0$.

**Sequences.** Since we are considering cumulative metrics, the metric values are accumulated for the case of sequences: for sequence $a \equiv a_1, a_2$ we have $T(a) = T(a_1) + T(a_2)$ and $\lambda(a) = \lambda(a_1) + \lambda(a_2)$.

**Service invocations.** For an activity $a$ that is an invoke to an external service, for both the running time $T(a)$ and the availability $\lambda(a)$ the analyzer needs to rely on empirically or analytically derived estimates, which include the local message handling and network delivery. In our approach, we deal with the ranges of possible values, rather than with probable or expected values. That means that in absence of any information, we simply have $T(a) \geq 0$ and $\lambda(a) \geq 0$, but the upper bounds on $T(a)$ and $\lambda(a)$, if known, must be safe, or else the prediction will be too optimistic and fail to detect some cases of possible SLA violations.

**Parallel flows.** In the case of a parallel flow $a \equiv a_1 \land a_2$, $T(a)$ must lay somewhere between $\max(T(a_1), T(a_2))$, when $a_1$ and $a_2$ run fully in parallel, and $T(a_1) + T(a_2)$, which is the worst, sequential case of execution. Therefore, it is safe to take

$$\max(T(a_1), T(a_2)) \leq T(a) \leq T(a_1) + T(a_2)$$

as a conservative approximation.

This approximation can however be too cautious and may lead to overly pessimistic estimates. If we have additional information about the semantics of the orchestration language and the implementation of the execution engine, we can refine the estimate for $T(a)$. For instance, if the execution of local activities is single threaded, while external services invocations are ensured to run in parallel, we can use the following scheme. Consider the case where $a_1$ and $a_2$ are sequences ending with an invoke activity, i.e., $a_1 \equiv a_{11}, a_{12}, \ldots, a_{1k}, a_1^*$ and we call $a_1^* \equiv a_{11}, a_{12}, \ldots, a_{1k}$ (respectively for $a_2$). We will assume that $a_1^*$ and $a_2^*$ are sequences of activities to be executed locally by a single thread, even if they appear in different branches of the flow, while $a_1^*$ and $a_2^*$ can be executed remotely in parallel. In this case, the total estimated time for the flow is

$$\max(T(a_1^*), T(a_1^*) + T(a_2^*) \leq T(a) \leq T(a_1^*), T(a_2^*) + \max(T(a_1^*), T(a_2^*))$$

If, say, $a_1^*$ is not an external invoke, but $a_2^*$ is, then $T(a_1^*)$ is part of $T(a_1^*)$. If neither $a_1^*$ nor $a_2^*$ are external invoke, then simply $T(a_1^*) = T(a_2^*) = 0$. This structural analysis can of course be easily extended to more than two parallel flows. The running time of an OR-parallel flow can be conservatively approximated using the case of AND-parallelism.

From the point of view of availability, parallel flows do not affect the total risk of failure, since the total availability depends on availability of all used components, regardless of their order of execution. Therefore, for $a \equiv a_1 \land a_2$ or $a \equiv a_1 \lor a_2$, we have $\lambda(a) = \lambda(a_1) + \lambda(a_2)$.

4 Or those that can be converted into a cumulative (e.g. additive) equivalents.
Conditionals. For a conditional \( a \equiv ([\text{cond}] \rightarrow a_1 ; a_2) \), where \( a_1 \) is the then part and \( a_2 \) is the else part, the metric depends on how the condition is evaluated. We introduce a Boolean variable \( b_{\text{cond}} \) to represent the result of the condition evaluation, so that we can state the following disjunctive constraint: either (1) \( b_{\text{cond}} = 1 \) and \( T(a) = T([\text{cond}]) + T(a_1), \lambda(a) = \lambda(a_1) \), or (2) \( b_{\text{cond}} = 0 \) and \( T(a) = T([\text{cond}]) + T(a_2), \lambda(a) = \lambda(a_2) \). The value of \( b_{\text{cond}} \) is generally unknown, but can be constrained to either 0 or 1 as the result of constraint solving. This makes it explicit that either the then or the else part can be taken, but not both.

Loops. In case of a loop \( a \) — while or foreach with body \( a_1 \) — we introduce an integer variable \( k_a \geq 0 \) that stands for the number of loop iterations. Then, we have \( T(a) = k_a \times (T([\text{cond}]) + T(a_1)) + T([\text{cond}]) \) and \( \lambda(a) = k_a \times \lambda(a_1) \). The actual value of \( k_a \) is generally unknown, but its inclusion into the constraints allows us to reason about the maximal or minimal number of loop iterations that lead to SLA compliance or violation.

3.5 Using Computational Cost Functions

To improve the precision of the predictions, the constraint-based predictor is able to use computational cost functions for service orchestrations [13], which, in this case, express lower and upper bounds of the number of loop iterations as a function of the input data to the orchestration. These computation cost functions may be automatically inferred at the start of an orchestration, statically determined at design time, or manually asserted for known cases. The inference of the computation cost functions depends on the semantics of the workflow constructs and the (sub-)language of conditions and elementary operations in which the orchestration is expressed.

If computation cost functions are available, the default constraint for the number of iterations of loop \( a \) (\( 0 \leq k_a \)) can be strengthened to \( \ell \leq k_a \leq u \wedge 0 \leq k_a \), where \( \ell \) and \( u \) are, respectively, lower and upper bounds on the number of iterations, which depend on the actual values of the input data. In the absence of one (or both) of the bounds, the corresponding constraint is simply not generated (as in Fig. 5, right).

3.6 Solving the Constraints

The constraints derived from the orchestration continuation relate the QoS metrics for the entire continuation with those of individual activities, component services, Boolean results of evaluating the conditions, the number of loop iterations, and the limits from the SLA. As such, they represent a constraint satisfaction problem [6] that can be solved for values of the constrained variables, which, in our case, include QoS metrics, Boolean conditions and loop iteration counters. Depending on the type of problem and the particular constraint solver used, solving the CSP may involve several iterations of constraint propagation and problem splitting [6, 1], which are used to reduce the equations in the original CSP to a series of simpler ones, before attempting to assign to the constrained variables values that satisfy the constraints.

In our case, we use the interval constraints (ic) solver from the ECL/PS® Constraint Logic Programming (CLP) system [2, 5]. The underlying Prolog subsystem of
ECLiPSe is used for constructing the constraints from a continuation, handling information on QoS metrics of component services, and reporting the results. The solver handles constrained variables over bound and unbound integer (discrete) and real number (dense) domains. The values of the constrained variables are represented as (possibly unbound) real or integer intervals. Integer variables with bounded domains are handled in a manner similar to finite domain solvers [6]. The solver directly supports disjunctive constraints (which we use for conditionals) and reified (Boolean valued) constraints.

The solver produces results given as bounds on values of the constrained variables, obtained from propagation of arithmetic constraints, or fails if the constraints cannot be satisfied. In our case, as mentioned before, we always solve two CSPs, one modeling SLA conformance and another one modeling SLA violation.

The constraint solver is complete, i.e., it does not discard feasible solutions. Therefore, upon constraint satisfaction, the answer intervals for the variables include all admissible values, and values outside these intervals cannot possibly satisfy the constraints. On the other hand, it may be that some combinations of values inside the answer intervals do not satisfy the constraints. Let us see an example: the constraint $0 \leq T(a_1) + T(a_2) \leq 100$ has as answer $T(a_1) \in [0..100] \land T(a_2) \in [0..100]$. This contains all feasible solutions (for example, $T(a_1) = 0 \land T(a_2) = 100$) but also combinations of values which do not satisfy the constraints (for example, $T(a_1) = 50 \land T(a_2) = 51$). Of course, if the latter values are fed into the constraint solver together with the initial constraint, the constraint solver will determine that the system is unsolvable.

4 Experimental Evaluations

Table 1 shows the results of running our QoS prediction framework applied to the orchestration in Fig. 5 (corresponding to the workflow in Fig. 1) and using execution time as QoS metric. The assumptions on ranges for the invocations of external services are shown at the bottom. These ranges would be updated by the QoS predictor based on the observation of invoke/reply events published by the process engine. Note that we are only concerned with the range of possible running times for each component, not the probability distributions within these ranges, and therefore we only need only to adjust the boundaries of the corresponding ranges.

The top part of Table 1 shows the results for the case of an unbound number of while loop iterations, which is the default if no additional information is provided. A series of successive assumed running time limits (500, 750, 1,500 and 3,000 ms) was considered, and both the SLA compliance (success) and violation results are shown. The meaning of the rest of the rows are as follows:

- **duration** shows the predicted running time ranges for the orchestration in ms.
- **cond(if)** is a Boolean value showing the possible evaluations of the condition in the conditional (1 for the “then” branch and 0 for the “else” branch).
- **iter(while)** shows the range of possible iteration counts in the while loop (corresponding to the repetition after testing the condition in the “else” branch).
- **E.C.D.T.** earliest certain detection times: the earliest time at which a certain violation or success can be detected.
Case 1: Unconstrained iterations

Successive running time SLA ranges

<table>
<thead>
<tr>
<th>Variable</th>
<th>Metrics</th>
<th>Success</th>
<th>Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration ms</td>
<td>0..500 ms</td>
<td>500..750 ms</td>
<td>750..1500 ms</td>
</tr>
<tr>
<td>cond(if) bool</td>
<td>0..1</td>
<td>1</td>
<td>0..1</td>
</tr>
<tr>
<td>iter(while) nat</td>
<td>0..∞</td>
<td>1..10</td>
<td>0..10</td>
</tr>
</tbody>
</table>

Case 2: Between 1 and 10 iterations

Successive running time SLA ranges

<table>
<thead>
<tr>
<th>Variable</th>
<th>Metrics</th>
<th>Success</th>
<th>Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration ms</td>
<td>0..500 ms</td>
<td>500..750 ms</td>
<td>750..1500 ms</td>
</tr>
<tr>
<td>cond(if) bool</td>
<td>0..1</td>
<td>1</td>
<td>0..1</td>
</tr>
<tr>
<td>iter(while) nat</td>
<td>0..10</td>
<td>1..10</td>
<td>0..10</td>
</tr>
</tbody>
</table>

Component running time assumptions

<table>
<thead>
<tr>
<th>Running time (ms)</th>
<th>local op</th>
<th>account svc</th>
<th>usage svc</th>
<th>reuse svc</th>
<th>gen svc</th>
<th>conf svc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ms .. 10 ms</td>
<td>500 ms .. 800 ms</td>
<td>200 ms .. 500 ms</td>
<td>100 ms .. 400 ms</td>
<td>200 ms .. 600 ms</td>
<td>100 ms .. 300 ms</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Sample QoS prediction results.

% E.C.D.T. percentage of the total (maximum) execution time which elapsed up to the E.C.D.T.

lead time between E.C.D.T. and the closest moment in which the orchestration can finish (i.e., the shortest time span to react in the worst case).

The results show that the lowest limit of 500 ms could not be met under the initial assumptions regarding execution times for atomic activities and external services. The 750 ms limit can be met, if the conditional evaluates to 1, meaning that the while loop is avoided. The 1500 ms limit can be met in both cases of the conditional, but can be violated only for the case of taking the “else” branch. Finally, for the range of running times between 1500 ms and 3000 ms, the prediction shows that, under the given assumptions, the only possible situation for both compliance and violation is taking the “else” branch, with the number of iterations in the range 0 .. 11 and 3 .. +∞, respectively. Note that for the latter limit, between 0 and 2 iterations guarantees compliance, and more than 11 iterations guarantees violation of the limit. An adaptation mechanism can, use these predictions to prepare and trigger adaptation actions that may prevent, minimize, or compensate for possible SLA violations ahead of time.

The earliest time at which a success or violation can be predicted depend on the particular execution. Let us look at an example: in Table 1, Case 1, columns “750 ms .. 1500 ms”, successes have been detected at 500 ms and SLA violations at 1200 ms. The reason that successes have been detected before violations is that these correspond to different executions: in the case of violation, the “else” branch (with the loop) has been taken, it is detected that there will be a violation after some iterations. On the other hand, if the “then” branch is taken, certainty of success is immediately detected, as there are no loops to be taken. With this interpretation in mind,
the constraint-based predictor is able to detect SLA violation with certainty up to between 300 and 500 ms in advance, while SLA conformance can be detected as early as after 500 or 700 ms of running time. In relative terms, SLA conformance has been detected in the experiments when only between a 23% and a 66% of the maximum execution time has elapsed, and SLA violations have been detected in some cases when only a 60% of the execution has elapsed.

The middle part of Table 1 shows a hypothetical case where, based on input data and computational cost functions, the predictor is able to infer that the actual number of loop iterations, in case the "else" branch is taken, must fall between 1 and 10. The results follow the same pattern as in the first case, but this time the predictor is able to infer that the duration of the orchestration under the assumptions must fall between 600 and 7,820 ms. This inferred running time range for the orchestration can be used by other parts of the runtime system (including predictors themselves) to update their QoS metrics assumptions on the deployed components. Note that the guarantee of at least one loop iteration increases the lead for the earliest certain detection of violations to 500 ms.

The average net time for performing one running time limit compliance/violation prediction depicted in Table 1 (not counting the time for sending and receiving data over the network), based on the average from 10,000 executions, was 0.574 ms on a small end-user non-dedicated machine.5

5 Implementation Notes

We have tested the approach using a prototype implementation of the architecture from Fig. 3, which includes the process engine, the QoS predictors, and the event bus, organized as a distributed and scalable system of components that communicate using reliable messaging. The tests included deployments on Linux and Mac OS X 32 and 64 bit platforms.

In our running prototype, the QoS predictors are implemented in ECLiPSe Constraint Logic Programming system, while the process engine (that executes orchestrations) is implemented in Ciao Prolog [9]. Both Prolog dialects support a variety of constraint logic programming techniques, but have, at the moment, slightly different orientation and strong points. ECLiPSe provides very robust, industrial-scale constraint solvers which can easily handle very complex problems involving thousands of constraints and variables, while Ciao is a flexible multi-paradigm programming environment with sophisticated support for concurrency. Fortunately the fact that they are both Prolog-based systems greatly facilitates interfacing and putting together the required architecture.

In our prototype, the language in Fig. 4 is used to define service orchestrations and to maintain instance control state throughout execution, so that there is no additional overhead in communicating continuations to QoS predictors, other than message transfer times. Also, any adaptation that changes the orchestration structure for

5 The tests were run on a 32-bit 2GHz Intel Core Duo notebook with 2GB of RAM, running Mac OS X 10.6.7 and ECLiPSe version 6.0_167.
a running instance can be simply implemented by replacing one continuation with another.

The messaging subsystem is implemented using ZeroMQ [11], which provides fast and reliable multi-part binary message exchange primitives on top of TCP networking and IPC subsystems, including request-reply, push-pull and publish-subscribe patterns. We have developed Prolog (Ciao and ECLiPSe) bindings to ZeroMQ with data (term) serialization that provide transparent higher-level data exchange primitives.

6 Conclusions

We have devised and implemented a method which makes it possible to predict possible situations of SLA conformance and violation, and to obtain information on the internal parameters of the orchestration (branch conditions, loop iterations) that may occur in these situations. The method is based on modeling QoS metrics of a service orchestration using constraints, based on assumptions on the behavior of the orchestration components. That analysis can, in principle, be applied at each step in an orchestration based on the current continuation. This allows periodic or continuous updating of the predicted bounds for QoS metrics for the orchestration and therefore a continuous assessing of conformance to SLA, which can be useful for proactive adaptation and self-healing. This approach can be combined with automatically inferred computational cost functions for service orchestrations, which can express the bounds of internal parameters (such as loop iterations) as functions of input data given to the orchestration instance, to provide a higher level of prediction precision. We have implemented the method in a prototype and reported some efficiency results.

Our future work will concentrate on making the implementation of all elements of the QoS prediction architecture laid out in this paper more complete and robust, including the process engine, beyond the prototype stage. We also plan to add support for different execution engines, targeting specifically those that have well-defined interfaces for event-listening plugins or can be adapted accordingly (e.g. because the implementation is open-source).

References


