Grant Agreement N° 215483

Title: *Validated set of adaptation and monitoring principles, techniques and methodologies considering context and HCI*

Authors: *CITY, FBK, POLIMI, SZTAKI, USTUTT, TUW, TILBURG, UNIDUE, UCBL*

Editor: *Gabor Kecskemeti (SZTAKI)*

Reviewers: *Harald Psaier (TUW)*
Claudia Di Napoli (CNR)
Annapaola Marconi (FBK)
Andreas Metzger (UNIDUE)

Identifier: *Deliverable # CD-JRA-1.2.7*

Type: *Deliverable*

Version: *1.0*

Date: *February 29, 2012*

Status: *Final*

Class: *External*

Management summary

This deliverable aims to present the research progress of the project partners in the project's last year. First, this deliverable puts special emphasis on the further refinements of the comprehensive adaptation and monitoring scenarios introduced in CD-JRA-1.2.5. Next, the research summarized in this document was focusing on the unexpected situations that could occur in cross-layer adaptation and monitoring techniques while they are maintaining context independent and HCI aware execution of large scale and heavily distributed service-based applications. The research results are presented through systematically listing and detailing the related research papers of the project partners. Finally, the deliverable offers an outlook on the future research directions in the field of adaptation and monitoring frameworks.

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)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero – The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany
University of Hamburg	Germany
Vrije Universiteit Amsterdam	Netherlands

Published S-Cube documents

All public S-Cube deliverables are available from the S-Cube Web Portal at the following URL:
http://www.s-cube-network.eu/results/deliverables/

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with todays limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-Cube materials are available from URL: `http://www.s-cube-network.eu/`

Foreword

This deliverable, CD-JRA-1.2.7 "Validated set of adaptation and monitoring principles, techniques and methodologies considering context and HCI" aims to address the following goals:

- to continue, refine and consolidate the monitoring and adaptation scenarios defined in CD-JRA-1.2.5, so they could encompass a contextual changes and HCI awareness while also reveal opportunities for handling unexpected situations in SBAs.
- through the summaries of the joint papers, present the research results and experiences on the realization and experimentation efforts while concretizing the previously defined scenarios.
- to identify future research directions derived from the research experiences in the current research domains of the different S-Cube partners that lead us towards the vision of the project.

Contents

Chapter 1

Introduction

In highly dynamic service environments, where the service availability, quality and reliability is dependent on several application-independent factors, the behavior and user experience about Service-Based Applications (SBAs) are highly dependent on the monitoring and adaptation techniques utilized during application execution. The different layers (e.g. BPM, SCC) of the SBAs however can be monitored and adapted with different approaches. These layers earlier were individually addressed, and with our previous deliverable we have shown several adaptation and monitoring scenarios that address the locality issues of the adaptation strategies (e.g., adaptation on the SCC layer could break BPM layer behavior of the SBA). Therefore the work package has explored different and orthogonal directions of integration of adaptation methodologies. These methodologies were first introduced in CD-JRA-1.2.3, then integrated in CD-JRA-1.2.4, finally adapted for a set of integration show cases in CD-JRA-1.2.5. In alignment with T-JRA-1.2.2 ("*Integrated Adaptation Principles, Techniques and Methodologies*"), the current deliverable validates, consolidates and, if needed, refines the show case scenarios with jointly written research papers and it provides an overview of the future research directions that lead towards sustainable research among the work package members even after the end of the project.

In several service-based applications, e.g., in the telecommunications domains, services need to interact heavily with humans, and it is important to adapt the interactions to different kinds of users and to different situations. This kind of adaptation should take into account different types of users (from developers to end-users), different levels of expertise, as well as different contexts of usage (e.g., connectivity, available devices, etc.). The contributed research papers in the deliverable provide the foundations for cross-layer, context-aware and user-driven monitoring, i.e., theories, principles, methodologies, and techniques that use contextual information and information about the user of monitoring results to drive monitoring activities. The deliverable – completing research task T-JRA-1.2.3 ("*Comprehensive, Context-Aware Monitoring*") – also reveals how the project members adapted the show case scenarios in order to handle the previously mentioned dynamically changing contexts and HCI environments.

This deliverable also aims at revealing the findings of the project partners about the principles, techniques and methodologies for proactive adaptation, i.e., to timely anticipate the need for changes across different functional layers. This deliverable reviews those joint scientific efforts that discuss the homogenization of proactive adaptation mechanisms in the different service-based application layers that experience different degrees of autonomy. The proactive adaptation capabilities discussed in the contributed research papers are also considering techniques to detect and handle unexpected situations, context awareness and user-driven adaptation taking into account different context factors, including those related to human errors or quality aspects of the operational environment of the service-based application (a strongly related research issue with work package JRA-1.3).

1.1 Relations with the Integrated Research Framework

1.1.1 Contribution to WP Challenges

The scope and the results of the deliverable directly contribute to the research challenges of the WP-JRA-1.2 and the S-Cube Integrated Research Framework. Specifically, this deliverable addresses the following challenges:

Comprehensive and integrated adaptation and monitoring principles, techniques, and methodologies. The research results focused on the scenario "*Quality-driven Multilayer SBA Monitoring and Adaptation*" are related to this challenge in sections 2.2.2, 2.2.4, 2.2.5, 2.2.7 and 2.2.9.

Proactive Adaptation and Predictive Monitoring. The scenario titled "*Assumption-based Proactive Monitoring and Adaptation*" focuses on this challenge, and it is detailed in several novel research papers that are detailed in sections 2.2.6 and 2.2.9.

Context- and HCI-aware SBA monitoring and adaptation. Those newly introduced research papers that are related to the scenario titled "*Context-based Adaptation and Monitoring*" address this challenge. See sections 2.2.1, 2.2.3 and 2.2.7.

The scenarios referred in the challenges listing were defined in CD-JRA-1.2.5 and are shortly summarized in Section 2.1.

1.1.2 Relations with other work-packages

As with previous research results, the crosscutting nature of the topic of the work package is represented in several joint research opportunities that were exploited during the last year of the project. The deliverable in several cases presents joint research papers that were collaboratively produced among the various work packages of the project. In the following, we list the most prominent connections of the research results presented in this deliverable:

Connections with JRA-1.1 Work package JRA-1.1 is focusing on the software engineering aspects of SBAs. The HCI and context aware automated adaptation of the monitoring system represents a new software engineering challenge that was collaboratively investigated between the JRA-1.1 and JRA-1.2. These collaborative efforts are reflected in section 2.2.1. The analysis and then the classification of the various BPEL extensions in section 2.2.8 reveals the effects on methodologies considering adaptation-aware business process design.

Connections with JRA-1.3. This work package aims at the quality characteristics of the SBAs. First, in section 2.2.5, we present how service quality aware autonomous behavior of federated cloud infrastructures supporting service-based applications could reduce SLA violations. Later on, in section 2.2.9, we propose the use of fuzzy rules in the field of proactive adaptation and monitoring in order to predict and reduce QoS dissatisfaction in service-based applications.

Connections with JRA-2.1. This work package targets the issues that occur in the Business Processes Layer during the execution of service based applications. In sections 2.2.2, 2.2.4 we investigate various aspects of the cross layer behavior of monitoring and adaptation frameworks. These works present A&M approaches that not only allow the BPM layer to get notifications from layers below, but we also present methodologies and techniques that interactively involve the BPM layer in selecting and enacting efficient adaptation strategies meeting all the requirements of the involved service-based application layers.

Connections with JRA-2.2. The work package JRA-2.2 circles around the research questions of service compositions and coordination. In this deliverable, several contributed research papers address context aware adaptation of service compositions, thus they were developed with strong collaboration with JRA-2.2 (e.g. see sections 2.2.1 or 2.2.3). In section 2.2.7, we propose service compositions deployed and dynamically managed over the Amazon EC2 cloud infrastructure, this work has involved also JRA-2.3 along with JRA-2.2.

Connections with JRA-2.3. Self-* service infrastructures and various discovery mechanisms are the main concerns of this work package. Thus, the topic of utilizing service-based applications on infrastructure as a service cloud computing systems has been discussed in two sections: 2.2.5 and 2.2.7. Building on top of these solutions the CLAM (cross layer adaptation and monitoring) framework is proposed (see sections 2.2.2 and 2.2.4) allowing the business and composition layers of service-based applications to consider infrastructure level events and adaptation options in unexpected situations.

1.2 Deliverable Structure

This document aims to present our achievements regarding the validated set of adaptation and monitoring principles, techniques and methodologies considering context and HCI. This deliverable is structured as follows:

- First, chapter 2 circles around the scientific papers we have written in collaboration aiming at the deliverable's target objectives. These papers are attached to the deliverable, and therefore this deliverable only provides a structured view on them. This chapter is subdivided into sections as follows:
  - Afterwards, in section 2.1, we define the way the research results of the S-Cube partners will be presented. Throughout the section, we shortly summarize the relevant content of the previous deliverables as a result, reading the current deliverable will not need an extensive knowledge about past results.
  - Next, in section 2.2, we provide a listing of the various research papers contributed to this deliverable. This listing not only provides the tables describing the individual contibutions, but it also provides brief descriptions of the research ideas that initiate the integration of monitoring and adaptation principles across service-based application layers. We remark that the results presented in this chapter are based on the materials presented in a set of papers that are referred from and attached to the current deliverable only.
- Then, in chapter 3, concludes the deliverable and the last year of work of the S-Cube project in the scope of adaptation and monitoring approaches.
- Finally, in the Appendix, the contributed papers are attached to the document and reported as separate chapters.

Chapter 2

Individual contributions

The goal of this chapter is to provide an overall view of the research carried out during the last year of the work package. The chapter presents the results of the work package through the latest research papers that present the improved and consolidated validation results and methodologies even considering unexpected situations. The chapter first provides a schematic overview about how each research paper will be presented (see section 2.1). Then, based on this schema, in section 2.2 we will present and align the novel approaches developed by the S-Cube partners that aim at addressing and supporting the context and HCI aware integrated cross-layer adaptation and monitoring.

2.1 Template for presenting the results

In order to provide a synthetic overview of the contributions of the partners and to relate them to the cross-layer adaptation & monitoring principles, techniques, scenarios and methodologies presented in CD-JRA-1.2.5, a synthetic template is proposed (Table 2.2). Through the template we aim at presenting the research contributions of the various S-Cube members to the overall research results and future research directions of the project in relation with work package JRA-1.2.

For each contributed research paper, the template in Table 2.2 is filled out considering four major areas.

- First, it describes the paper in general.
- Then, it puts the paper in context of the research executed in the work package.
- Afterwards, it dives into the details of the contributed research paper.
- Finally, the template table is followed by the extended abstract of the paper that describes the concepts, the research questions and highlights the proposed solutions.

In the general description, first, the *title* and the *authors* of the article is specified. The authors field not only lists the researchers contributing to the particular paper but it also highlights if the paper was jointly developed by two or more project members. Next, the authors are followed by the *type* and a *short description* of the research presented in the filled in template. The type can indicate if a new *methodology* is introduced for a particular adaptation and monitoring related domain. Also, the type can reveal if a new *model* is proposed to describe the A&M behavior inside service-based applications. Then, the research paper could also contribute with a new *technique* applicable in cross-layer A&M that is also highlighted as a new type. In the type field, the template could also highlight the inclusion of *experimental evaluation* in the described research paper. Finally, the *classification* type of contribution reveals that the particular research carried out has provided a systematic evaluation and provided a taxonomic discussion of the research field.

The work package related contextual description in the template starts with the identification of the targeted *integration scenario*. The integration scenarios were introduced in the previous deliverable of this work package (titled "*CD-JRA-1.2.5 – Comprehensive, integrated adaptation and monitoring*

principles, techniques and methodologies across functional SBA layers considering context and HCI") to provide a common reference for the adaptation and monitoring problem in hand, providing a common basis for the different research works developed within the project. In the following we provide a short overview of the three identified scenarios:

Quality-driven Multilayer SBA Monitoring and Adaptation. The SBA reference model in this scenario primarily targets applications implemented as long-running business processes and workflows. The primary adaptation and monitoring problem in this scenario considers the quality requirements expressed as KPI, PPM, SLA, and other metrics of the application, across its functional layers. The scenario exploits data mining techniques to perform the diagnosis of the problem that leads to the violation of the high-level quality requirements. Regarding the individual quality properties at different layers, the scenario associates different adaptation actions (also in that layer) that are expected to improve the specific quality factor and, therefore, contribute to an overall improvement in quality.

Assumption-based Proactive Monitoring and Adaptation. In this scenario we consider business processes that are realized on top of executable service compositions implemented in BPEL. The idea that distinguishes this approach in this integrated scenario is based on the use of assumptions. This scenario uses assumptions to relate the continuously monitored data to the SBA requirements. The scope of the scenario is to be able to anticipate the needs for adaptation and to provide the corresponding monitoring and adaptation support in order to enable proactive adaptation of service-based business processes in case of failures and requirement violations.

Context-based Adaptation and Monitoring. In this scenario we primarily focus on SBAs, in which the context of those SBA plays the key role in the various activities across the SBA life-cycle.

Title	the title of the research work
Authors	Authors of the contribution
Type	Type of contribution (Methodology / Model / Technique / Classification / Experimental Evaluation / ..)
Short description	Brief description of the contribution with respect to the research problem presented in the integration scenario
Targeted integration scenario	One or more scenarios from the previous deliverable, namely: (i) Quality-driven Multilayer SBA Monitoring and Adaptation, (ii) Assumption-based Proactive Monitoring and Adaptation, (iii) Context-based Adaptation and Monitoring.
Contribution to the adaptation problem	Specific adaptation problem addressed by the approach (if any)
Contribution to the monitoring problem	Specific monitoring problem addressed by the approach (if any)
Integrated SBA Layers for Monitoring	Any combination of monitoring across BPM, SCC, SI layers that the research work has provided a consolidated approach for.
Integrated SBA layers for adaptation	Any combination of adaptation solutions through BPM, SCC, SI layers that the contributed article has aimed at.
Cross-layer mechanisms	Mechanisms applied for interaction between SBA layers involved
Unexpected situations handled	List of unexpected situations identified and handled by the approach
Architecture elements	Specify and refine the relevant components of the architecture
Requirements/constraints	The important assumptions and limitations of the current approach. These might be referred in the future research directions (e.g. possible ways to reduce limitations)
Refined SBA Life Cycle activities	Which activities of the SBA lifecycle were affected and updated by the current research paper
Future research directions	List the the remaining research issues that needs to be addressed in the topic of the actual scientific result

Table 2.2: Contribution overview template

The distinguishing factor of the SBAs participating in this scenario refers to the fact that these SBAs operate in continuously changing environments. This integrated scenario relies on the use of templates that characterize the monitoring and adaptation activities in general settings, which are then instantiated in different way for the specific contexts.

The rest of the WP related contextual description is focused on the particular paper's relations and effects on the monitoring and adaptation problem: (i) *Contribution to the adaptation problem*, (ii) *Contribution to the monitoring problem*, (iii) *Integrated SBA Layers for Monitoring*, (iv) *Integrated SBA layers for adaptation*, (v) *Cross-layer mechanisms*.

Finally, the last four rows of the template detail the paper and underline its strongest contributions to the work package. Therefore, for each paper, we have described the *unexpected situations* that the research identified and proposed solutions for. In the following row, the template lists the *architecture elements* to provide insights to architectures that could handle the previously identified problems and unexpected behavior. Afterwards, we have listed those *requirements and/or constraints* that need to be met in order to allow the correct operation of the proposed approach. Next, the table summarizes the *SBA life-cycle elements and activities* (first defined by the work package JRA-1.1 in [3]) that were refined while carrying out the research described in the paper. In the last row, for every contributed paper, the table summarizes the future research directions that S-Cube researchers identified for sustainable and continued research in the area of monitoring and adaptation of service-based applications.

2.2 Summary of the individual contributions to the integrated cross-layer adaptation and monitoring principles

2.2.1 Identifying, Modifying, Creating, and Removing Monitor Rules for Service Oriented Computing

Title	Identifying, Modifying, Creating, and Removing Monitor Rules for Service Oriented Computing [4]
Authors	Ricardo Contreras (CITY), Andrea Zisman (CITY)
Type	Technique
Short description	Supports the adaptation of the monitor activity due to changes in the context of the user of the SBA (context HCI-aware)
Targeted integration scenario	Context-based Adaptation and Monitoring.
Contribution to the adaptation problem	The technique can be combined with techniques/approaches capable of dealing with SBA adaptation
Contribution to the monitoring problem	Automatic specification and deployment of monitor rules. This is performed based on the user context and the specification of the SBA
Integrated SBA Layers for Monitoring	Focus on SCC
Integrated SBA layers for adaptation	Focus on SCC
Cross-layer mechanisms	-
Unexpected situations handled	Optimisation: monitor rules repository kept to a minimum size
Architecture elements	**Rule Adaptor:** identifies, creates, modifies and removes monitor rules. **Path Identifier:** retrieves the part of the specification related to a user context type. **Rule Verifier:** whether a rule is valid for a SBA or not. Monitor: uses rules to verify the SBA

Requirements/constraints	(i) Uses event calculus (EC) for the specification of rules. (ii) Use of patterns (also specified in EC) for the different user context types. (iii) Assumes the existence of user context annotations to retrieve the part of the specification related to a particular context.
Refined SBA Life Cycle activities	**Operation and Management:** monitoring mechanisms for the identification of problems. **Requirements Engineering and Design:** context types present in the SBA (annotations)
Future research directions	To investigate new techniques dealing with the (semi-) automatic adaptation of the monitor as a reaction to changes/adaptation in the application.

Extended abstract

Monitoring of service-based systems is considered an important activity to support service-oriented computing. Monitoring can be used to verify the behaviour of a service-based system, and the quality and contextual aspects of the services participating in the system. Existing approaches for monitoring service-based systems assume that monitor rules are pre-defined and known in advance, which is not always the case. We have created a pattern-based HCI-aware monitor adaptation framework to support identification, modification, creation, and removal of monitor rules. In the framework, changes in the monitor rules are based on users interaction with a service-based system and different types of user context.

2.2.2 CLAM (Cross-layer Adaptation manager)

Title	CLAM (Cross-layer Adaptation manager) [5]
Authors	Annapaola Marconi (FBK), Marco Pistore (FBK), Asli Zengin (FBK), Luciano Baresi (Polimi)
Type	Methodology / Model / Technique
Short description	CLAM (i) analyzes the effects and consequences of an adaptation trigger for the whole service-based system, (ii) addresses the negative influences on the system through a gradual construction of adaptation strategies.
Targeted integration scenario	Quality-driven Multilayer SBA Monitoring and Adaptation
Contribution to the adaptation problem	Handling of cross-layer adaptations
Contribution to the monitoring problem	–
Integrated SBA Layers for Monitoring	–
Integrated SBA layers for adaptation	BPM, SCC, SI
Cross-layer mechanisms	State-of-the-art analysis and adaptation techniques are integrated at the CLAM platform. CLAM provides a cross-layer managing mechanism to coordinate these techniques.
Unexpected situations handled	Searching for existing approaches and finding the running adaptation tools is not trivial. We try to handle this issue by searching through what S-Cube partners can provide and by directly contacting the authors / developers of the tools.

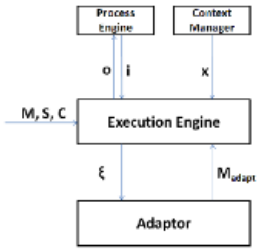
Architecture elements	Rule engine, model updater, tree constructor, ranker: • The core part of CLAM is the rule engine. It reasons on its predefined rules and contacts the appropriate tool at each step of its analysis. • The model updater is responsible for creating and keeping the updated SBS configurations each time an adaptation is proposed in the process. • The tree constructor keeps the proposed adaptation alternatives in a tree-form structure where each tree branch corresponds to an alternative adaptation strategy. • The ranker gets the complete tree and select the best path based on some pre-defined selection criteria.
Requirements/constraints	(i) High level cross-layer dependency model of the SBS should be given as input to the CLAM. (ii) Each time a new tool is plugged in the platform, cross-layer rules should be updated. (iii) The performance of CLAM depends on the integrated tools. The more tools CLAM has, the more comprehensive will be the analysis, but it will take more time as well.
Refined SBA Life Cycle activities	All parts of the SBA life cycle activities
Future research directions	In the current implementation we showed the CLAM approach on a case study. Future research will comprise the enhancement and evaluation of the approach through the trial of various case studies from different application scenarios. The use of CLAM in different adaptation cases and in different domains and possibly with different sets of tools will demonstrate the applicability, extensibility and flexibility of the approach.

Extended abstract

Adaptation is a cross- and multi-layer problem, and at each layer it could target different, possibly conflicting, system aspects. For example, when reorganizing the composition, one could privilege the application's price, its speed, or the compliance with some external regulations. Many existing solutions have addressed adaptation in a "local" way by only considering one system aspect at one layer; in contrast CLAM fosters a comprehensive approach able to address different layers and aspects concurrently, reason on the dependencies and consequences among them, and identify global solutions. These solutions must harmonize layers and system aspects, and provide an integrated adaptation plan based on local activities. CLAM relies on a comprehensive high-level model of the application and of the layers behind it. Each model element is associated with a set of analyzers to understand the problem, solvers, to identify possible solutions, and enactors, to apply them on the element. The coordinated operation of analyzers, solvers, and enactors is governed by predefined rules that identify the dependencies, and consequences, between the elements of the model and run the different tools. For each adaptation need, CLAM produces a tree of alternative adaptations, identifies the most convenient one, and applies it.

2.2.3 Adaptation of Service-based Business Processes by Context-Aware Replanning

Title	Adaptation of Service-based Business Processes by Context-Aware Replanning [6]
Authors	Antonio Bucchiarone (FBK), Raman Kazhamiakin (SayService), Marco Pistore (FBK), Heorhi Raik (FBK)
Type	Methodology / Technique
Short description	Adaptation of business processes to exogenous context changes and negative operation outcomes which are not handled by the process by the run-time and context-aware planning of the adaptation activities
Targeted integration scenario	Context-based Adaptation and Monitoring

Contribution to the adaptation problem	Run-time adaptation of business processes to exogenous context changes and negative operation outcomes unhandled by the process
Contribution to the monitoring problem	–
Integrated SBA Layers for Monitoring	–
Integrated SBA layers for adaptation	SCC
Cross-layer mechanisms	–
Unexpected situations handled	exogenous context changes and negative operation outcomes that are not handled by the process
Architecture elements	The architecture contains four elements communicating to each other: 1. Execution engine coordinates the work of other elements and tracks the execution of the process; 2. Process engine provides the execution of process activities; 3. Context manager tracks exogenous context changes; 4. Adaptor, given an adaptation problem, uses a planner to derive an adaptation procedure; 
Requirements/constraints	(i) Only one adaptation strategy is used (local adaptation that tries to change the problem locally, so that the process can be executed from the point where it has failed); (ii) The approach is supposed to be used as short-term adaptation, i.e. instance based adaptation versus process model evolution; (iii) The adaptation process optimality criterion is the just minimal number of execution steps;
Refined SBA Life Cycle activities	–
Future research directions	The approach can benefit from using different adaptation strategies (for example, jump forward or roll back and compensate strategies). The approach can be successfully used to refine abstract activities at run time. In this case some most volatile activities remain abstract and are refined only immediately before the execution, taking into account the current context. The adaptation history can be successfully used to bring corresponding changes to the process model (process evolution). While deriving the adaptation activities, different optimality criteria can be used.

Extended abstract

Service-based business processes are typically used by organizations to achieve business goals through the coordinated execution of a set of activities implemented as services and service compositions. Since they are executed in dynamic, open and non-deterministic environments, business processes often need to be adapted to exogenous context changes and execution problems. In this paper we provide an adaptation approach that can automatically adapt business processes to run-time context changes that impede achievement of a business goal. We define a formal framework that adopts planning techniques to automatically derive necessary adaptation activities on demand. The adaptation consists in identifying recovery activities that guarantee that the execution of a business process can be successfully resumed and, as a consequence, the business goals are achieved. The solution proposed is evaluated on a real-world

scenario from the logistics domain.

2.2.4 Multi-layer Monitoring and Adaptation

Title	Multi-layer Monitoring and Adaptation [7]
Authors	Sam Gunea (Polimi), Gabor Kecskemeti (SZTAKI), Annapaola Marconi (FBK), and Branimir Wetzstein (USTUTT)
Type	Methodology / Technique
Short description	We propose a framework that integrates layer specific monitoring and adaptation techniques, and enables multi-layered control loops in service-based systems. The proposed approach is evaluated on a medical imaging procedure for Computed Tomography (CT) Scans, an e- Health scenario characterized by strong dependencies between the software layer and infrastructural resources.
Targeted integration scenario	Quality-driven Multilayer SBA Monitoring and Adaptation
Contribution to the adaptation problem	Multi-layer adaptation of service based systems: identification and enactment of holistic adaptation strategies coordinating layer-specific adaptation mechanisms.
Contribution to the monitoring problem	Multi-layer monitoring and analysis of service-based systems: correlation (as general and domain-specific metrics) of monitoring events captured by layer-specific sensors, and holistic identification of the adaptation need through the analysis of the aggregated data.
Integrated SBA Layers for Monitoring	BPM, SCC, SI
Integrated SBA layers for adaptation	BPM, SCC, SI
Cross-layer mechanisms	Cross-layer correlation of monitoring events, cross-layer analysis of adaptation needs, cross-layer identification of adaptation strategies, cross-layer adaptation enactment.
Unexpected situations handled	Integration of heterogeneous, layer-specific monitoring and adaptation techniques.
Architecture elements	The main architectural elements of the framework are: **Monitoring and Correlation Component:** obtains low-level data/events from the process or from the context of execution using Dynamo, or from the infrastructure using Laysi; aggregates the monitoring events using the event correlation capabilities provided by EcoWare. **Adaptation Needs Analysis Component:** the Influential Factor Analysis component identifies the relations between the set of metrics (potential influential factors) and the KPI category based on historical process instances; the Adaptation Needs Analysis component uses this information to identify the adaptation needs, i.e., what is to be adapted in order to improve the KPI. **Cross Layer Adaptation Manager Component:** identifies the application components that are affected by the adaptation actions, and computes an adaptation strategy that properly coordinates the layer-specific adaptation capabilities. **Adaptation Enactment component:** enacts the adaptation strategy properly coordinating software adaptations through DyBPEL, and infrastructure adaptations through Laysi.
Requirements/constraints	Business layer not fully covered (lack of monitoring/adaptation capabilities to be integrated in the framework).
Refined SBA Life Cycle activities	Operation & Management, Identify adaptation need, Identify adaptation strategy, Enact adaptation.

Future research directions	We will continue to evaluate the approach through new application scenarios, and through the addition of new adaptation capabilities and adaptation enacting techniques. We will also integrate additional kinds of layers, such as platforms, typically seen in cloud computing setups, and business layers. This will also require the development of new specialized monitors and adaptations. Finally, we will study the feasibility of managing different kinds of KPI constraints.

Extended abstract

We propose a framework that integrates software and infrastructure specific monitoring and adaptation techniques developed within S-Cube, enabling multi-layered control loops in service-based systems. All the steps in the control loop acknowledge the multi-faceted nature of the system, ensuring that we always reason holistically, and adapt the system in a coordinated fashion. In our prototype we have focused on the monitoring and adaptation of BPEL processes that are deployed onto a dynamic infrastructure.

Building upon our past experiences we have integrated process and infrastructure level monitoring with a correlation technique that makes use of complex event processing. The correlated data, combined with machine-learning techniques, allow us to pinpoint where the problems lie in the multi-layered system, and where it would be more convenient to adapt. We then build a complex adaptation strategy that may involve the software and/or the infrastructure layer, and enact it through appropriate effectors.

In the Monitoring and Correlation step, sensors deployed throughout the system capture run-time data about its software and infrastructural elements. The collected data are then aggregated and manipulated to produce higher-level correlated data under the form of general and domain-specific metrics. The main goal is to reveal correlations between what is being observed at the software and at the infrastructure layer to enable global system reasoning.

In the Analysis of Adaptation Needs step, the framework uses the correlated data to identify anomalous situations, and to pinpoint and formalize where it needs to adapt. It may be sufficient to adapt at the software or at the infrastructure layer, or we may have to adapt at both.

In the Identification of Multi-layer Adaptation Strategies step, the framework is aware of the adaptation capabilities that exist within the system. It uses this knowledge to define a multi-layer adaptation strategy as a set of software and/or infrastructure adaptation actions to enact. A strategy determines both the order of these actions and the data they need to exchange to accomplish their goals.

In the Adaptation Enactment step, different adaptation engines, both at the software and the infrastructure layer, enact their corresponding parts of the multi-layer strategy. Each engine typically contains a number of specific modules targeting different atomic adaptation capabilities.

The proposed approach is evaluated on a medical imaging procedure for Computed Tomography (CT) Scans, an e-Health scenario characterized by strong dependencies between the software layer and infrastructural resources.

2.2.5 Facilitating self-adaptable Inter-Cloud management

Title	Facilitating self-adaptable Inter-Cloud management [8]
Authors	Gabor Kecskemeti (SZTAKI), Michael Maurer (TUW), Ivona Brandic (TUW), Attila Kertesz (SZTAKI), Zsolt Nemeth (SZTAKI), Schahram Dustdar (TUW)
Type	Methodology
Short description	This contribution introduces a methodology to autonomously operate cloud federations by controlling their behavior with the help of knowledge management systems. Such systems do not only suggest reactive actions to comply with established Service Level Agreements (SLA) between provider and consumer, but they also find a balance between the fulfillment of established SLAs and efficient energy consumption.

Targeted integration scenario	Quality-driven Multilayer SBA Monitoring and Adaptation
Contribution to the adaptation problem	This contribution offers three options to incorporate the concepts of knowledge management systems into the Federated Cloud Management architecture to perform adaptation: (i) local integration is applied on a per deployed component basis, e.g. every CloudBroker utilizes a separate KM system for its internal purposes; (ii) global integration is based on a single KM system that controls the autonomous behavior of the architectural components considering the available information from the entire cloud federation; and (iii) a hybrid KM system combining both global and local integration options.
Contribution to the monitoring problem	Within analyzing the various autonomous actions that the KM system can exercise, the authors investigated the monitoring system of the FCM architecture and the possible metrics it can collect to allow the identification of those cases when the architecture encounters unsatisfactory behavior. In the current system, they monitor and analyze the behavior of CloudBrokers, the FCM repository and individual service instances.
Integrated SBA Layers for Monitoring	SI.
Integrated SBA layers for adaptation	SI.
Cross-layer mechanisms	–
Unexpected situations handled	The proposed approach for Inter-Cloud management by FCM is capable of handling various unexpected situations based on its knowledge management solution with extendible adaptation rules. In this contribution, the following actions can be triggered for unexpected situations: Reschedule of service calls, Rearrange VM queues, Extend/Shrink VM Queue, Rearrange VA storage , Self-Instantiated Deployment.
Architecture elements	**Monitors:** Various metrics have been defined in the KM system that are used in the rules for triggering actions. **Adaptation strategy engine:** The actions of the KM represent the strategies in this contribution. **Adaptation enactment engine:** the Global Autonomous Manager incorporating the Knowledge Management System is responsible for executing the actions. **Adaptation capabilities:** These capabilities are predefined in the extensible set of rules within the KM system.
Requirements/constraints	The proposed Self-adaptable Inter-Cloud management solution adopting rule-based techniques for its knowledge management to federate clouds of multiple infrastructures. In the future additional rules will be defined to cover a wider range of system failures or malfunctions.
Refined SBA Life Cycle activities	**Construction:** Developers may provide additional adaptation actions to the Knowledge Management System of the Global Autonomous Manager **Operation, management and QA:** The Global Autonomous Manager continuously execute the predefined rules to perform autonomous control **Identify adaptation requirements:** The rules of the KM system specify this process. **Identify adaptation strategy:** The actions to be triggered according to the monitored events are predefined in the rules of the KM system **Deployment and provisioning:** Service deployments of the Self-adaptable Inter-Cloud management architecture are managed by the FCM components following the predefined rules of the KM system. The paper describes its operation by detailing the rule for removing VA from native repository of a specific Cloud due to high global costs.

Future research directions	In this contribution various metrics have been defined to indicate possible SLA violations in federations, and rules have been developed to trigger adaptation actions in the case of predicted violations. Regarding future works, the authors plan to investigate more the green aspects in the autonomous behavior of cloud federations. They will also aim at defining new rules for advanced action triggers and evaluate the applicability of case based reasoning. An experimental system will also be set up to investigate the effects of the autonomous behavior on the overall performance of the cloud federation.

Extended abstract

Cloud Computing represent a novel computing paradigm where computing resources are provided on demand following the rules established in form of Service Level Agreements (SLAs). SLAs represent the popular formats for the establishment of electronic contract between consumer and provider stating the terms of use, objectives and penalties to be paid in case objectives are violated. Thus, appropriate management of Cloud Computing infrastructures (such as Amazon, Rackspace, Eucalyptus, Opennebula) is the key issue for the success of Cloud Computing as the next generation ICT infrastructure. Thereby, the interaction of the system with humans should be minimized while established SLAs with the customers should not be violated. Since Cloud Computing infrastructures represent mega scale infrastructures comprising up to thousands of physical hosts, there is a high potential of energy waste by overprovisioning resources in order to keep the violation level of SLAs as low as possible. Federated cloud management systems offer a simplified use of these infrastructures by hiding their proprietary solutions. As the infrastructure becomes more complex underneath these systems, the situations (like system failures, handling of load peaks and slopes) that users cannot easily handle, occur more and more frequently. Therefore, federations need to manage these situations autonomously without user interactions.

This paper introduces a methodology to autonomously operate cloud federations by controlling their behavior with the help of knowledge management systems. Such systems do not only suggest reactive actions to comply with established Service Level Agreements (SLA) between provider and consumer, but they also find a balance between the fulfillment of established SLAs and efficient energy consumption. The paper adopts rule-based techniques as its knowledge management solution and provides an implementation of federated clouds on top of multiple simulated infrastructures. Using the FCM architecture as the basis of the investigations, the authors analyzed different approaches to integrate the knowledge management system within this architecture, and found a hybrid approach that incorporates fine-grained local adaptation operations with options for high-level override. This research carried out has pinpointed the adaptation actions and their possible effects on cloud federations. Finally, metrics have been developed that could indicate possible SLA violations in federations, and defined rules that could trigger adaptation actions in the case of predicted violations.

2.2.6 Web Service Interaction Adaptation using Complex Event Processing Patterns

Title	Web Service Interaction Adaptation using Complex Event Processing Patterns [9]
Authors	Yehia Taher (TILBURG), Michael Parkin (TILBURG), Mike P. Papazoglou (TILBURG), Willem-Jan van den Heuvel (TILBURG)
Type	Technique

Short description	Differences in Web Service interfaces can be classified into two types, signature and protocol incompatibilities, and techniques exist to resolve one or the other of these issues but rarely both. This paper describes an approach based on complex event processing to resolve both signature and protocol incompatibility problems that may exist between Web Service protocols. Our approach uses a small set of operators that can be applied to incoming messages individually or in combination to modify the structure, type and number of messages sent to the destination. The paper describes how CEP-based adapters, which are deployable in CEP engines, can be automatically generated from automata representations of the operators through a standard process and presents a proof-of-concept implementation.
Targeted integration scenario	Assumption-based Proactive Monitoring and Adaptation
Contribution to the adaptation problem	Resolution of both signature and protocol incompatibility problems that may exist between Web Service protocols.
Contribution to the monitoring problem	–
Integrated SBA Layers for Monitoring	–
Integrated SBA layers for adaptation	–
Cross-layer mechanisms	–
Unexpected situations handled	Situations regarding the structural and behavioural incompatibilities between web services.
Architecture elements	The architecture is composed of two integrated environments. (i) The Design Time Environment is used to instantiate the template operators. This phase models the adapter using operator automata through the use of an incompatibility detection process to produce a platform independent model. (ii) The run-time environment, on the other hand, contains a CEP platform, in which the transformation phase takes the platform independent model to produce the adapter as a CCQ (continuous computation query) for a CEP engine, i.e, a platform specific model. It consists of a continuous query engine and a set of SOAP message integration layers that allow the environment to send and receive messages to and from Web Services.
Requirements/constraints	The approach assumes services to be modeled using Automata and it does not involve incompatibilities regarding semantics or deadlocks, but handles those related only to structural and behavioral properties.
Refined SBA Life Cycle activities	The design-time activities relates to the construction, and deployment & provisioning phases of the SBA life-cycle; and the run-time activities defined in this study relates to all adaptation phases including the identification of adaptation needs and strategies, and their enactment as well as operation & management.
Future research directions	Ongoing work aims at extending the proposed solution toward two directions: (i) comparing our similarity measures to others and testing detection algorithm on real services; and (ii) assisting business process designers in determining how to address incompatibilities.

Extended abstract

Web services provide a solution to the integration of distributed software through the standardization of data format, interface definition language, transport mechanism and other interoperability aspects such as security and quality of service. The Web Service Description Language (WSDL) defines a Web Service interface as a document in XML format and a service as a set of endpoints that operate on messages containing either document-oriented or procedure-oriented information. The interface document provides a contract between the provider of a service and its users and allows some flexibility for the service provider as it hides the details of the implementation of the service from those using it.

Web Service interfaces (i.e., WSDL, BPEL, etc.) define the messages and protocol that should be used to communicate with the service. However, if two services wish to interact successfully, they must both support the same messages and protocol through the implementation of compatible WSDL and BPEL documents. Unfortunately, this is difficult to achieve in practice; Web Services are often developed independently and follow different standards or approaches in constructing their interfaces and Web Service compositions will often use of services in ways that were not foreseen in their original design and construction. Therefore, it is likely that most Web Services will be incompatible since many services will not support the same interface.

To solve this problem, one needs to generate adapters that can make two Web services collaborate even if they were not designed in that a way. The generation of adapters requires the elicitation of mismatches between services. The study describes an approach that makes use of complex event processing (CEP) to resolve both signature and protocol incompatibility problems that may exist between Web Service interfaces. The approach is oriented towards the use of a set of operators that can be applied to incoming messages individually or in combination to modify the structure, type and number of messages sent to the destination. By using a continuous query engine running within a CEP platform, we demonstrate how adapters can be automatically generated for a CEP engine and how signature and protocol adaptation between Web Services can be achieved practically in a proof-of-concept implementation. The adapters are capable of intercepting messages sent between services and can adapt the structure, type and number of incoming messages into the desired output message or messages.

2.2.7 Design for Self-adaptation in Service-oriented Systems in the Cloud

Title	Design for Self-adaptation in Service-oriented Systems in the Cloud [10]
Authors	A. Bucchiarone (FBK), C. Cappiello (POLIMI), E. Di Nitto (POLIMI), S. Gorlatch (MUENSTER), D. Meilander (MUENSTER), A. Metzger(UNIDUE)
Type	Methodology
Short description	In this work we focus on two main aspects, that is, the kinds of changes that trigger self-adaptation in a service-oriented system and the strategies that can be adopted to deal with adaptation. We provide a preliminary contribution to the systematic understanding of adaptation across layers (Infrastructure and service composition layers).
Targeted integration scenario	(i) Quality-driven Multilayer SBA Monitoring and Adaptation (ii) Context-based Adaptation and Monitoring.
Contribution to the adaptation problem	Distribution handling for the dynamic adaptation of running application sessions by adding/removing Cloud resources on demand using particular adaptation strategies
Contribution to the monitoring problem	Monitoring of application-specific data, e.g., update rate, number of entities, etc.
Integrated SBA Layers for Monitoring	SCC, SI
Integrated SBA layers for adaptation	SCC, SI
Cross-layer mechanisms	This work will not scrutinizes the aspect of cross-layer adaptation
Unexpected situations handled	• Change in QoS, e.g., caused by unreliable hoster resources; • Change in the machine, e.g., caused by increasing user interactions, making computation of state updates more expensive; • Change in the business context, e.g., more users connect to the application due to changing user preferences.

Architecture elements	In order to support Real-Time Online Interactive Applications (ROIA) development and adaptation on Clouds, we develop the RTF-RMS resource management system [2] on top of the Real-Time Framework (RTF – [1]). RTF-RMS implements the following mechanisms for ROIA development on Clouds: (i) *Monitoring* of application-specific data, e.g., update rate, number of entities, etc. (ii) *Distribution handling* for the dynamic adaptation of running application sessions by adding/removing Cloud resources on demand using particular adaptation strategies. (iii) *Application profiles* that allow developers to specify application-specific adaptation triggers. (iv) High-level *development support* for communication handling and application state distribution.
Requirements/constraints	• In its current implementation, RTF-RMS only supports the Amazon Elastic Compute Cloud (EC2) interface. • For the definition of application profiles, the application developer has to manually find concrete values and suitable thresholds for his application by conducting experiments in a Cloud environment. • Continuous adaptation during runtime eventually leads to an inefficient application architecture.
Refined SBA Life Cycle activities	All core life-cycle activities
Future research directions	Our work on ROIA development along the S-Cube Lifecycle Model identified the demand for balancing run-time adaptation and application re-design. Particularly, the continuous adaptation on Cloud resources may lead to an inefficient application structure which requires application redesign. In our future research, we will address this task with particular regard to the adaptation on Cloud resources that implies additional challenges, like unknown resource locations, heterogeneous resource performance, etc. We will study how to incorporate new design-for-adaptation activities into the software development process using RTF-RMS and the S-Cube Lifecycle Model, e.g., how to define suitable adaptation triggers and strategies.

Extended abstract

Service-oriented systems are able to offer complex and flexible functionalities in widely distributed environments by composing different types of services. These systems have to be adaptable to unforeseen changes in the functionality offered by component services and to their unavailability or decreasing performances. Furthermore, when systems are made available to a high number of potential users, they should also be able to dynamically adapt to the current context of use as well as to specific requirements and needs of the specific users. In order to address these issues, mechanisms that enable adaptation should be introduced in the life-cycle of systems, both in the design and in the runtime phases.

In this work we will go through the life-cycle of a service-oriented system highlighting those activities that are needed to support adaptation. The adaptation activities can be performed at various layers of the service-oriented system. In particular, they can concern the layer where services are composed together or the layer of the executing infrastructure, typically, a cloud system. To exemplify the various steps and activities we use an example from the domain of real-time online interactive applications.

2.2.8 A Classification of BPEL Extensions

Title	A Classification of BPEL Extensions [11]
Authors	Oliver Kopp (USTUTT), Katharina Grlach (USTUTT), Dimka Karastoyanova (USTUTT), Frank Leymann (USTUTT), Michael Reiter (USTUTT), David Schumm (USTUTT), Mirko Sonntag (USTUTT), Steve Strauch (USTUTT), Tobias Unger (USTUTT), Matthias Wieland (USTUTT)

Type	Classification
Short description	In this paper, we provide (i) a classification of existing BPEL extensions and (ii) guidelines to develop extensions.
Targeted integration scenario	–
Contribution to the adaptation problem	Some of the classified BPEL extensions aim at improving the adaptability of BPEL by means e.g. of aspect-oriented programming and run-time injection of additional behaviour.
Contribution to the monitoring problem	Some of the classified BPEL extensions aim at enabling monitoring of BPEL process, e.g. by representing the state of activities as resources that can be accessed at run-time, and by exposing hooks that trigger the evaluation of business rules.
Integrated SBA Layers for Monitoring	SCC
Integrated SBA layers for adaptation	SCC
Cross-layer mechanisms	–
Unexpected situations handled	Several of the classified BPEL extensions aim at enabling recovery of activity failures and increased reliability of BPEL processes.
Architecture elements	Service composition, Monitoring engine, Adaptation engine
Requirements/constraints	Depends on the particular BPEL extension
Refined SBA Life Cycle activities	Depends on the particular BPEL extension
Future research directions	The findings presented in this work on the design and implementation of BPEL extensions remains valid in the context of the Business Process Model and Notation (BPMN) language. As part of our future work, we will classify BPMN extensions according to the presented classification framework.

Extended abstract

The Business Process Execution Language (BPEL) has been designed for the implementation of business processes using Web service technology. Nowadays, BPEL is used for implementing business processes in numerous different scenarios such as the automation of scientific simulations, the provisioning Software as a Service (SaaS) applications and as exchange format for business processes. With the growth over time of the adoption of BPEL, it has been applied to scenarios and use-cases that were not originally envisoned, and for which its constructs are not sufficient. For instance, the modelling of sub-processes is a functionality that the BPEL specification and, as a consequence, standard-conforming implementations do not cover.

As a result, BPEL is frequently extended for supporting desired functionality that is not available in standard BPEL. Depending on the particular purpose, an extension may improve efficiency, increase flexibility, ensure better performance, or add more functionality. However, extensions have also disadvantages. The whole toolset that is used for business process management (BPM) needs to support the extension. Common components of this toolset are applications for modeling, adapting, executing, monitoring, and analyzing the processes. Moreover, if business partners exchange (parts of) their processes, their toolsets need to understand and support the extensions as well.

In this paper, we provide (i) a classification of existing BPEL extensions and (ii) guidelines to develop extensions. An interesting finding of this work is that only around half of the sixty-two classified extensions conform to the definition of BPEL extension set by the BPEL specification. In some cases, this is because the design of the extension has not carefully taken into account the limitations it had to conform to. However, it is also a symptom that the ways of extending BPEL allowed by the specification are limited, and, in retrospective, perhaps too limited.

2.2.9 A Penalty-Based Approach for QoS Dissatisfaction Using Fuzzy Rules

Title	A Penalty-Based Approach for QoS Dissatisfaction Using Fuzzy Rules [12]
Authors	Barbara Pernici (POLIMI), Seyed Hossein Siadat (POLIMI), Salima Benbernou (UPD/UCBL), Mourad Ouziri (UPD/UCBL)
Type	Model/Methodoly/Experimental Evaluation.
Short description	Quality of Service (QoS) guarantees are commonly defined in Service Level Agreements (SLAs) between provider and consumer of services. Such guarantees are often violated due to various reasons. QoS violation requires a service adaptation and penalties have to be associated when promises are not met. However, there is a lack of research in defining and assessing penalties according to the degree of violation. In this paper, we provide an approach based on fuzzy logic for modeling and measuring penalties with respect to the extent of QoS violation. Penalties are assigned by means of fuzzy rules.
Targeted integration scenario	Quality-driven Multilayer SBA Monitoring and Adaptation & Assumption-based Proactive Monitoring and Adaptation
Contribution to the adaptation problem	We are interested in changes of service quality and the associated penalties in case of non exact fulfilments of QoS stipulated in the SLA. We are aware of degree of penalties. For that, we take advantage of the fuzzy logic for measuring the overall penalties based on the QoS and selecting service adaptation strategies
Contribution to the monitoring problem	–
Integrated SBA Layers for Monitoring	In order to compose services, we need services fulfilling the best QoS.
Integrated SBA layers for adaptation	In order to compose services, we need services fulfilling the best QoS.
Cross-layer mechanisms	–
Unexpected situations handled	–
Architecture elements	–
Requirements/constraints	We assume that the relevant QoS and their associated penalties are collected from the monitoring phase. Our system requires human expertise to define the membership function for each parameters and determine the number of rules.
Refined SBA Life Cycle activities	**Requirements Engineering:** Penalties definition according to the QoS, fuzzy parameters. **Construction:** definition of if-then-else-fuzzy rules. **Identify adaptation needs:** quality factor analysis, importance of penalties. **Identify adaptation strategy:** Selection of adaptation strategies based on an inference approach and adaptation priority.
Future research directions	(i) Identify more varieties of penalties. (ii) Penalties on multi SLAs.

Extended abstract

QoS guarantees defined in contracts may be violated due to various reasons. This situation needs to be handled through applying adaptation techniques not to bring dissatisfaction. The concept of penalty has been used in SLAs to compensate the conditions under which guarantee terms are not met. Despite some works have been done on the description, negotiation and monitoring of SLAs, however there is not much work on the definition of penalty clauses. WS-Agreement specification has been studied to define penalties based on different types of violation. However, penalties are assigned to violation of a single property instead of assigning penalties to violation of overall QoS. Moreover, the approach introduces a method for measuring penalties which is for fixed predefined number of violations, instead of measuring the extent of violation and assigning penalties accordingly. One main issue is how to determine the appropriate amount of penalties as compensations from providers to satisfy customers. As

quality parameters can be satisfied partially, the assessment of penalties can be based on the degree of quality violation. Understanding the violation degree is a prerequisite for assessing penalties. However, measuring such violation is yet an open research challenge. In addition, the influencing factors in defining penalties need to be identified. A static amount of penalty (manual approaches) does not reflect the extent of violation at runtime. The amount and level of penalties are related to the degree of quality violation provided from the provider side. On the other side, the customers characteristics may also affect the amount of penalties. For example a penalty to satisfy a gold/loyal customer is different with the one for an occasional customer. To the best of our knowledge, there is no formal relation between the assigned penalty and its influencing factors. Moreover, the extent and type of penalties are not clearly expressed in related work. However, understanding such relation and providing a mapping between them are complicated issues. We argue what is missing is a suitable mechanism for modelling penalties that takes into account both provider and consumer sides. Apart from the degree of violation, we also consider the state of customer and service provider with respect to their past history (e.g. whether the service has been penalised previously) in determining the right amount of penalties. However, as the relation between a given penalty and its infuencing factors is not linear, conventional mathematical techniques are not applicable for modelling penalties. The goal of this paper is to apply an inference technique using fuzzy logic as a solution to propose a penalty-based approach for compensating conditions in which quality guarantees are not respected. Fuzzy logic is well suited for describing QoS and measuring quality parameters. We demonstrate a penalty inference model with a rule based mechanism applying fuzzy set theory. Measuring an appropriate value for penalties with respect to the amount of violation is described in this work.

Chapter 3

Future Research challenges and Conclusions

3.1 Future Research Challenges

In this section, we present the research challenges on monitoring and adaptation identified during the S-Cube Research Roadmap Workshop (Barcelona, November 22, 2011), attended by 40 project members and associate members.

The overall objective was to identify research challenges that may become relevant after and beyond S-Cube (in 510 years) and which have the potential to radically challenge existing thinking (i.e., beyond incremental). The Research Roadmap Workshop was organized along four topical sessions:

- S1: “Service life-cycle and software engineering”,
- S2: “Service technology foundations”,
- S3: “Multi-layer and mixed-initiative monitoring and ?adaptation for service-oriented systems”,
- S4: “Online service quality prediction for proactive ?adaptation”.

In the following we briefly introduce the challenges emerged for S3, which are going to be presented in a paper to the 2012 ICSE workshop on “European Software Services and Systems Research – Results and Challenges”.

3.1.1 Challenge 1. Approaches for retrieving and analyzing context information to support individuals in performing the right adaptation decisions in user-centric systems.

(i) *User-driven monitoring:* when the user context changes, the way the SBA is monitored should reflect the changes, (ii) *User-driven adaptation:* react to changes in the user context and generate a flexible interaction protocol that allows the user to control and coordinate the execution, (iii) users as *spectators*: at the same time service consumers, service inventors, and data donors, (iv) provision of ways that *encourage and incentivize users* to create and share contents and data with the system and the other peers in the network.

3.1.2 Challenge 2. Decentralized models and techniques to monitor and predict service quality issues.

(i) Techniques to combine information from SOA layers with the one coming from the *network infrastructure*, (ii) definition of *cross-layer quality metrics* considering the whole service delivery chain, including communication networks, (iii) use of *decentralized strategies* for the monitoring and predicting models.

3.1.3 Challenge 3: Techniques for combining and cross-correlating observations, predictions and events from different sources and provided by different techniques.

(i) *Flexible and dynamic correlation* of useful information (observations, predictions, and events) from different sources, across the functional layers, and provided by different analysis, decision and adaptation mechanisms, (ii) *learning correlation rules automatically* to allow SBAs to adapt to unforeseen situations.

3.1.4 Challenge 4: Assurances for adaptation.

(i) Quality assurance techniques to *prevent* run-time design decisions/adaptations to lead to *inconsistent situations*, (ii) assurances of service-oriented systems in order to architect *resilience against unknown situations* and for dealing with rare events, (iii) concepts and techniques to avoid *unwanted co-adaptation, malicious adaptations*, as well as races and other anomalies in order to ensure trustworthy self-adaptation and evolution of open, service-oriented systems, (iv) concepts and techniques for *formally guaranteeing the correctness* of adaptations.

3.2 Conclusions

This deliverable summarized the research carried out by the members of the S-Cube project while investigating and solving problems in relation to the main challenges of the work package JRA-1.2. In this deliverable, we have presented the ways of consolidating our joint research on the validated set of adaptation and monitoring principles, techniques and methodologies with special focus in context and HCI awareness. This work has been presented through several jointly written scientific papers that were systematically analyzed in the context of the deliverable's main aims.

The future work and research directions of the sustainable research and collaboration are as diverse as the research results presented in this deliverable. The diversity of the future work is inherited from the three previously identified research scenarios of the deliverable titled CD-JRA-1.2.5. The research directions are detailed in the summary tables for each contributed paper, here we only provide a short overview on the future approaches S-Cube members identified for handling unexpected situations in context and HCI aware service-based applications. We provide the summaries along the research scenarios they are going to enhance:

Quality-driven Multilayer SBA Monitoring and Adaptation. Future research should comprise the enhancement and evaluation of the new layers incorporated into the SBA (such as platforms, typically seen in cloud computing setups) through the trial of various case studies from different application scenarios. These scenarios should consider situations such as unknown resource locations or heterogeneous resource performance. For the evaluation of the SBA novel analyzer hierarchies, specialized monitoring, tailor-made adaptation strategies and their selection criteria have to be defined to enable flexible selection of the enacted adaptation strategies on the system. Finally, new studies have to be initiated to target the feasibility of managing different kinds of KPI constraints.

Assumption-based Proactive Monitoring and Adaptation. The scenario should be extended towards two directions: (i) comparing the currently competing similarity measures and testing detection algorithms on services deployed in lifelike environments; and (ii) assisting business process designers in determining how to address incompatibilities upon service substitutions. Researchers need to identify new ways to penalize improper proactive changes in the SBA while considering multiple SLA constraints and violations.

Context- based Adaptation and Monitoring. Investigate new techniques dealing with the (semi-) automatic adaptation of the monitor as a reaction to context changes/adaptation in the application. The adaptation history should be used to bring corresponding changes to the process model (process evolution).

Bibliography

[1] The Real-Time Framework – www.real-time-framework.com, 2011.

[2] Meilaender, Dominik; Bucchiarone, Antonio; Cappiello, Cinzia, Di Nitto, Elisabetta; Gorlatch, Sergei. Using a Lifecycle Model for Developing and Executing Real-Time Online Applications on Clouds, In Proc. of 7th International Workshop on Engineering Service-Oriented Architectures (WESOA), 2011.

[3] Elisabetta di Nitto, editor. State of the art report on software engineering design knowledge and Survey of HCI and contextual Knowledge, July 2008. S-Cube project deliverable: PO-JRA-1.1.1. http://www.s- cube-network.eu/achievements-results/s-cube-deliverables.

[4] Ricardo Contreras and Andrea Zisman. 2011. Identifying, modifying, creating, and removing monitor rules for service oriented computing. In Proceedings of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS '11). ACM, New York, NY, USA, 43-49.

[5] Zengin, A., Marconi, A., & Pistore, M. (2011). CLAM: Cross-layer Adaptation Manager for Service-Based Applications. In Proceedings of the International Workshop on Quality Assurance for Service-Based Applications, Qasba 11 (pp. 2127). Acm.

[6] Bucchiarone, A., Pistore, M., Raik, H., & Kazhamiakin, R. (2011). Adaptation of Service-based Business Processes by Context-Aware Replanning. In Proc. SOCA 2011.

[7] S. Guinea, G. Kecskemeti, A. Marconi and B. Wetzstein. "Multi-layered Monitoring and Adaptation", in Service-Oriented Computing: 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings. ISBN: 978-3-642-25535-9.

[8] G. Kecskemeti, M. Maurer, I. Brandic, A. Kertesz, Zs. Nemeth and S. Dustdar: "Facilitating self-adaptable Inter-Cloud management", in Proceedings of the Cloud Computing for Compute and Data Intensive Applications special session of the 20th Euromicro conference on parallel, distributed and network-based processing. Leipniz Supercomputing Centre, Garching, February 15-17th 2012.

[9] Taher, Yehia; Parkin, Michael; Papazoglou, Mike P.; van den Heuvel, Willem-Jan. "Adaptation of Web Service Interactions using Complex Event Processing Patterns", in Service-Oriented Computing: 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings. ISBN: 978-3-642-25535-9.

[10] Bucchiarone, A., Cappiello, C., Nitto, E. D., Gorlatch, S., Mailander, D., & Metzger, A. (2011). Design for Self-adaptation in Service-oriented Systems in the Cloud. In D. Petcu, & J. L. V. Poletti (Eds.), European Research Activities in Cloud Computing. Cambridge Scholars Publishing.

[11] Kopp, Oliver; Grlach, Katharina; Karastoyanova, Dimka; Leymann, Frank; Reiter, Michael; Schumm, David; Sonntag, Mirko; Strauch, Steve; Unger, Tobias; Wieland, Matthias; Khalaf, Rania: A Classification of BPEL Extensions. In: Journal of Systems Integration. Vol. 2(4), Online, 2011

[12] Barbara Pernici, Seyed Hossein Siadat, Salima Benbernou, Mourad Ouziri: A Penalty-Based Approach for QoS Dissatisfaction Using Fuzzy Rules. ICSOC 2011: 574-581

Appendix A

Identifying, Modifying, Creating, and Removing Monitor Rules for Service Oriented Computing

Identifying, Modifying, Creating, and Removing Monitor Rules for Service Oriented Computing

Ricardo Contreras
Department of Computing
City University, London, Northampton Square
London EC1V 0HB, UK
+44 20 7040 8552
Ricardo.Contreras.1@soi.city.ac.uk

Andrea Zisman
Department of Computing
City University, London, Northampton Square
London EC1V 0HB, UK
+44 20 7040 8346
a.zisman@soi.city.ac.uk

ABSTRACT
Monitoring of service-based systems is considered an important activity to support service-oriented computing. Monitoring can be used to verify the behavior of a service-based system, and the quality and contextual aspects of the services participating in the system. Existing approaches for monitoring service-based systems assume that monitor rules are pre-defined and known in advance, which is not always the case. We present a pattern-based HCI-aware monitor adaptation framework to support identification, modification, creation, and removal of monitor rules based on user's interaction with a service-based system and different types of user context. A prototype tool has been implemented to demonstrate the framework.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – corrections, enhancement, extensibility.

General Terms
Algorithms, Management, Verification.

Keywords
Service monitoring, rules, patterns, adaptation, HCI context

1. INTRODUCTION
Service-oriented computing (SOC) has been recognized as an important paradigm for software development. Various approaches and techniques have been proposed to support different areas and activities related to SOC. One of these activities is concerned with monitoring of service-based systems; i.e., the activity of collecting information about the execution of a service-based system and verifying if the system is operating correctly by comparing the collected information with the properties of the system. These properties are known as *monitor properties* or *monitor rules* and can be used to verify the behavior of a service-based system [3][4][25][30], the quality of the services participating in a system [17], and the contextual information of the services participating in the system and the system itself [6][8].

Existing approaches for monitoring service-based systems assume that monitor rules are pre-defined and known in advance. However, this is not always the case given that during execution time of a service-based system it is possible to have changes in the (i) system or set of services used by the system (due to unavailability or malfunctioning of a service), (ii) types of interaction of the users with the system, and (iii) context characteristics of the user interacting with the system. Therefore, it is necessary to have ways of identifying monitor rules, modifying existing rules, removing existing rules that are obsolete, or creating new rules to support the needs of the monitor. We call this process *monitor adaptation*.

As an example, consider a Cultural Event service-based system (CE_SBS) that provides general information about cultural events in a certain city, allows ticket acquisition by different types of users, supports scheduling of different cultural programs in a year, provides catering services for certain special performances (e.g., premier performances, group performances), and allows both customers and professional critics to review cultural events. Suppose that the CE_SBS is used by managers and employees of several venues in a City (e.g., theaters, show houses, concert halls, opera houses), and by customers interested in cultural events that may or not be members of one or several of the venues. For example, managers of different venues (role) use this application to schedule and organize the different programs in a year for his/her venue taking into consideration other performances for that year in different venues; while a member of a venue (role) uses the system to periodically receive information about the different events in the venue based on the member's interest (preferences) and has priority on purchasing tickets before customers that are not members (case (iii) above). In these cases, it is necessary to have monitor rules for the different services used in the system depending on the user context. In addition, customers can book special events that include or not catering services (case (ii) above). In this case, new monitor rules relevant to booking and paying for catering services are required (needs). Moreover, general customers are able to provide simple reviews about a performance that they have seen (skills), while professional critics are supposed to provide reviews of a performance considering several detailed characteristics (skills). Furthermore, suppose the situation that after a while, the service in the system that verifies conflicts and constraints about various events in a City that is used to support the scheduling of events, becomes unavailable. Assume that a new service that provides these functionalities, but also allows searching and allocating necessary physical and personnel resources for the various events is used to replace the initial service. In this case, the monitor system needs to verify the new functionality provided by the service (i.e., resource allocation) (case (i) above).

PESOS'11, May 23-24, 2011, Waikiki, Honolulu, HI, USA

In this paper we present the MADap (Monitor ADaptation) framework for monitor adaptation as defined above. The work presented in this paper has been carried out as part of the European funded Network of Excellence S-Cube [26]. Our framework concentrates on HCI-aware monitor adaptation in which changes in the monitor rules are based on user's interaction with a service-based system and different types of user context. The user context information of our concern includes *cognition*, *role*, *skill*, *need*, and *preferences* of a user, represented in user models based on an ontology that we have created. The framework is based on the use of patterns for the monitor rules representing different context types. The monitor rules are concerned with the execution parts of a service-based system specification for the user context types. The patterns are used to support the (a) identification, (b) modification, (c) creation, and (d) removal of monitor rules.

The remainder of this paper is structured as follows. Section 2 describes an overview of our framework. Section 3 presents the monitor adaptation process used by the framework, including the processes to identify, modify, create, and remove monitor rules. Section 4 presents implementation aspects and initial evaluation of the framework. Section 5 discusses existing related work. Section 6 summarizes the work and describes future work.

2. FRAMEWORK OVERVIEW

The MADap framework assumes two different types of user context, following the classification proposed in [10][12][19], namely (a) *direct user context types* and (b) *related user context types*. The direct user context types represent information of the characteristics of the users and include *role*, *skill*, *need*, *preferences*, and *cognition* context types. The related user context types represent information that may influence user information and include *time*, *location*, and *environment* context types. We have developed an ontology to represent the different user context types. A brief description of the various user context types is presented in Table 1.

Table 1. Description of user context types

Context Type		**Description**
Direct	Role	*Social behavior of an individual within the domain of a service-based system*
	Skill	*The level of expertise of an individual with respect to a service-based system*
	Need	*An individual's requirement or desire from a service-based system*
	Preferences	*An individual's choice over pre-established alternatives of computational resources of a service-based system*
	Cognition	*Individual's characteristics associated with the process of thought (the ways that individuals think, feel or react)*
Related	Time	*The moment an individual interacts with a service-based system*
	Location	*The place where an individual interacts with a service-based system*
	Environ-ment	*Information related to the environment where a service-based system is used*

A graphical representation of the ontology we have developed is shown in Figure 1. In the figure the different context types are represented as classes, with subclasses in some cases, and are associated with a central class representing the user. These associations indicate relationships between the different attributes of a context type (e.g., occupation for context class role) and a user class. For each class their attributes and respective data types are presented inside the class. Please note that at this stage we do not consider associations between the different context types given that the focus of the work is on relationships between users and contexts and not between context and context.

As described in Section 1, the framework is based on the use of *rule patterns* for the different user context types. It is possible to have different patterns for the same context type. Both the patterns and the monitor rules are described in Event-Calculus (EC) [27]. The use of Event Calculus (EC) to describe monitor rules has been advocated in [30] and has shown to be appropriate to support the representation of several types of rules. Event Calculus allows (i) rules to be represented as first order logic, which provides sufficient expressiveness for a large range of applications; (ii) specification of quantitative temporal constraints and relationships that are necessary to be taken into consideration when monitoring service-based systems; (iii) distinction between events and states that are necessary to describe the behavior of a system and interaction of users with the system; and (iv) definition of the influences between events and states.

EC uses *events* and *fluents* to represent the behavior of a system. An event occurs at a specific instance of time and may change the state of a system. A fluent is a condition of a system state and may be affected by the occurrences of events. Both events and fluents are represented in EC by predicates.

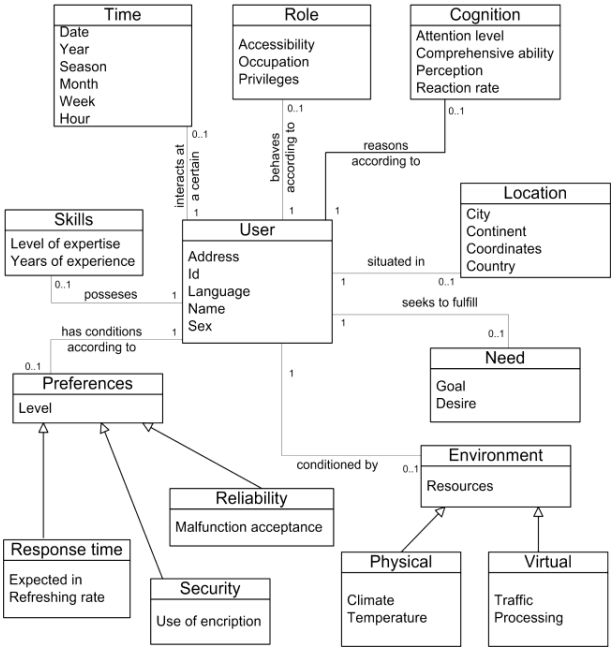


Figure 1: User context ontology

The occurrence of an *event*, at some time *t*, is represented by the predicate *Happens(event,t,R(t1,t2))*, which means an *event* occurs at a time *t*, where *t* is within an interval between *t1* and *t2*. The time boundaries, represented by *t1* and *t2,* can be specified using time variables or arithmetic expressions over time variables, and represent the lower and upper time boundaries. The initialization of a *fluent*, is represented by the predicate *Initiates(event,fluent,t)*, which means a *fluent* starts to holds after an *event* occurs at a time *t*. The predicate *HoldsAt(fluent,t)*, means a *fluent* holds (is valid) at a time *t*. The termination of a *fluent*, is represented by the

predicate *Terminates(event,fluent,t)*, which means a *fluent* ceases to hold after an *event* occurs at a time *t*. A detailed description of Event Calculus can be found in [27].

A rule pattern is composed of two parts, namely (a*) monitor rule* part and (b) *assumptions* part. The monitor rules represent properties of a service-based system that need to be monitored. The assumptions represent event calculus formulae that need to be used to identify state information of the system. An example of a rule pattern for role context type and the instantiation of this pattern for the CE_SBS described in Section 1 for role "Manager" are shown in Figures 2.a and 2.b, respectively.

Rule
Happens (ic_Initial-Event, t1, R(t1,t1) =>
Happens (ic_Event, t2, R(t1,tn))
Assumption
Happens(ic_Event,t, R(t,t)) => **Initiates**(ic_Event,*fluent*, t)

Figure 2.a: Rule pattern for role context type

Rule
Happens (*ic_start_CES*, t1, R(t1,t1)) =>
Happens (*ic_performance_schedule*, t2, R(t1,t1+2500))
Assumption
Happens (*ic_performance_schedule*, t1, R(t1,t1)) =>
Initiates (*ic_performance_schedule*, *per-schedule*, t1)

Figure 2.b: Instantiation of rule pattern in Figure 2.a

The monitor rule part in Figures 2.a and 2.b state that after the initial event of the service-based system specification (*ic_Initial-Event* and its instantiation *ic_Start_CES*) is executed, a system event concerned with a user context role (*ic_Event* and its instantiation *ic_performance_schedule*) should be executed in time t2, where t1 ≤ t2 ≤ tn (t1 ≤ t2 ≤ t1+2500). The assumption part states that when different system events occur, different fluents are initiated. In this case, when event *ic_performance_schedule* occurs, fluent *per-schedule* is instantiated. The patterns used by the framework are general in order to be employed in different types of service-based systems. Events in a pattern can represent either requests for an operation (when specified with *ic* prefix) or responses from an operation (when specified with *ir* prefix).

A rule pattern may have *invariant parts*, which depend on the context type associated with the pattern. An invariant part does not change for distinct instantiations of the pattern. The invariant parts for the role context type pattern in Figure 2.a is shown in Figure 3 (variant parts are represented by ____). Other patterns have been created to represent the other context types and can be found in [20]. They are not shown here due to space limitations.

Rule
Happens (ic_Initial-Event, t1, R(t1,t1) =>
  Happens (____, t2, R(t1,tn))
Assumption
Happens (____, t, R(t, t)) => **Initiates** (____, ____ , t)

Figure 3: Example of invariant part for role type pattern

Figure 4 presents the architectural overview of the MADap framework. As shown in the figure, the main components of the framework are *Rule Adaptor*, *Path Identifier*, *Rule Verifier*, and *Monitor*. The framework also uses *Rule Patterns*, *Semi-instantiated Patterns*, *Monitor Rules*, *User Models, Service-based System (SBS) Specification,* and *Service Level Agreements (SLAs).* It assumes rule patterns and monitor rules in a repository.

The Rule Adaptor is responsible for the identification, modification, creation, and removal of monitor rules. More specifically, it receives events about changes in the context characteristics of the user or interaction of the users with the system, and invokes the Path Identifier to identify paths in the specification of the service-based system that are relevant to the event. The Path Identifier retrieves the parts in the specification that are related to the context type represented in the event and its instance (e.g., context *role*, instance *manager*).

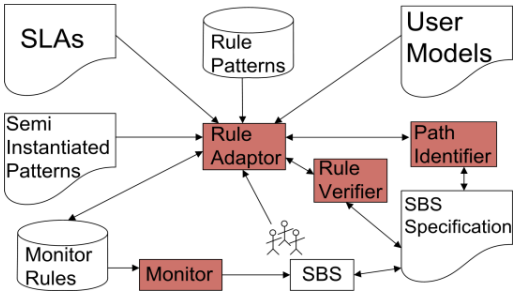


Figure 4. Architecture overview of MADap framework

The Rule Adaptor also uses the context type to identify relevant Rule Patterns for this context type and instantiates these patterns with the identified information from the service-based system specification and User Models. These are called *semi-instantiated rule patterns* since they do not have instantiated values for time variables or time gaps in the rule patterns. The User Models represent the characteristics of the users.

The Rule Verifier is used to verify if an existing monitor rule in the repository is still a valid rule for a service-based system. It is possible to have rules that become obsolete due to changes in the service-based system (e.g., certain functionalities are not executed anymore, new functionalities are added to the system).

We assume specifications of service-based systems in BPEL [7] due to its wide acceptance, and that the BPEL processes have conditions on the various user context types and their instances. These conditions match the syntax of a context type in a received event. This assumption is not unrealistic since different behavior of a service-based system due to different user characteristics needs to be represented in the service-based system specification.

The Rule Adaptor uses the semi-instantiated rule patterns to identify monitor rules. In the case where monitor rules that totally match the semi-instantiated rule patterns are *identified*, these rules are either used as they stand by the Monitor component or have their time values modified, when necessary, and subsequently used by the Monitor component. In the situation in which no rules that match the semi-instantiated rule patterns are identified, new monitor rules are *created* based on the semi-instantiated patterns. In the case in which there are monitor rules that match the invariant parts of the semi-instantiated rule patterns, the Rule Verifier checks if these rules are still valid for the service-based system. In positive case, these rules have their time values *modified*, if necessary. Otherwise, these rules are *removed* from the repository and a new rule based on the semi-instantiated pattern is *created*. The newly created rules are added into the repository and used by the Monitor component to verify the service-based system.

In the framework, we use the Monitor tool described in [30]. However, our approach can be used with any monitor tool that makes use of monitor rules represented in Event Calculus. The

Monitor tool receives requests from a service requestor to verify, at regular intervals, the satisfiability of properties (represented as monitor rules) of a service-based system. It intercepts run-time messages exchanged between a service-based system and its services and verifies the satisfiability of the properties against these messages. It contains (a) a *service client* that is responsible to invoke a service in a service-based system; (b) and *event collector* that is responsible to gather information during the execution of a service-based system and the services deployed by the service based system, or information exchanged between the service client and its respective services; and (c) an *analyzer* that is responsible to check the satisfiability of the properties.

3. MONITOR ADAPTATION PROCESS

In the framework, the monitor adaptation process is triggered by an event representing a context type Ci. Based on the context type, the Rule Adaptor identifies the patterns concerned with context Ci, invokes the Path Identifier to identify the parts of the BPEL specification that are related to Ci (i.e., the parts of BPEL specification with conditions that match Ci), and uses this information to semi-instantiate the identified patterns. The semi-instantiated patterns are compared to existing rules in the repository in order to identify if a relevant rule (a) already exists in the repository and (a.1) can be used as it stands, (a.2) needs to be modified, (a.3) needs to be removed, or (b) needs to be created.

The need to use information from the service-based system specification to semi-instantiate the relevant pattern is due to the fact that in some situations the pattern itself is not sufficient to support the identification of the correct monitor rule in the repository. For example, suppose an event for *customer* role context type. Assume the pattern for context type role shown in Figure 2.a and the monitor rules in Figure 5 below in the rule repository (without the assumptions for simplicity). In this case, all the three rules in Figure 5 match the rule pattern in Figure 2.a. However, rules R1 and R2 are concerned with role *customer* while rule R3 is concerned with role *manager*. If the pattern is used to identify rules in the repository, all three rules will be returned. But, following the part of the BPEL specification for the CE_SBS shown in Figure 6, the events representing the operations *shows* and *timetables* are relevant to role *customer*. Using these events, the two semi-instantiated patterns shown in Figure 7 will be specified and rules R1 and R2 will be correctly matched with the semi-instantiated patterns. It should be noted that information about these events could not be part of the initial patterns since the patterns are general for a certain context type and the events and the monitor rules are specific to a service-based system and instances of the various context types.

R1 **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
Happens (*ic_shows*, t2, R(t1,t1+2000))
R2 **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
Happens (*ic_timetables*, t2, R(t1,t1+4000))
R3 **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
Happens (*ic_performanceSchedule*, t2, R(t1,t1+2500))

Figure 5. Examples of monitor rules for role context type

As shown in Figure 7, the semi-instantiated patterns do not have instantiated values for time variables or time gaps. In order to specify the boundary of the time values, the framework assumes these values to be identified from the response time information of a service, or operations of a service, that are normally defined in Service Level Agreements (SLAs) between the services participating in the service-based system and the system itself. Another option is to use historical execution time data for a service, when available. The assumption that SLAs will be available for participating services is not unrealistic, since SLAs are currently used to establish business agreements between service providers and consumers. Moreover, the response times of a service or operations are attributes that appear in SLAs.

```
…<bpel:condition>
<![CDATA[$input.payload/tns:role_occupation="Customer"]]>
</bpel:condition>
<bpel:invoke name="Shows" partnerLink="Shows"
operation="shows" portType="ns:Shows"
inputVariable="ShowsRequest"
  outputVariable="ShowsResponse"> </bpel:invoke>
<bpel:invoke name="TimeTables"
partnerLink="TimeTables"
  operation="timeTables" portType="ns:TimeTables"
  inputVariable="TimeTablesRequest"
  outputVariable="TimeTablesResponse">
</bpel:invoke> …
```

Figure 6. Part of the BPEL process for CE_SBS

SI_RP1 **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
: **Happens** (*ic_shows*, t2, R(t1,tn))
SI_RP2 **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
: **Happens** (*ic_timetables*, t2, R(t1,tn))

Figure 7. Examples of semi-instantiated role patterns

```
Monitor_Adaptation (SI_Rule, SBS_Spec, SLAs, RRep) {
//SI_Rule: semi-instantiated pattern
//FI_Rule: fully-instantiated pattern
//SBS_Spec: service-based system specification
//SLAs: SLAs for the services and operations in the SI_Rule
//RRep: Rule Repository
//R: Rules in RRep
Search SI_Rule in RRep;
If (RRep has Rules that fully match SI_Rule) {
 For Every R in Rules {
  If (time in SLAs is within time values in R) {
    Do-nothing;}
  Else {Adjust time in R based on SLAs;}
  End If
 } End For
Else {
 Create FI_Rule by instantiating SI_Rule time with times in SLAs;
 If (RRep has Rules that only match invariant parts of SI_Rule) {
  For Every R in Rules {
    If (there is a path in SBS_Spec that uses R) {
      // Rule R is not obsolete
      If (time in SLAs is within time values in R) {
        Do-nothing; }
      Else {Adjust time in R based on SLAs; }
      End If
    Else {Remove R from RRep; }
    End If
  } End For
  Add FI_Rule to RRep; }
 Else {
  //There are no rules that match the semi-instantiated rule
  Create FI_Rule by instantiating SI_Rule time with times in SLAs;
  Add FI_Rule to RRep; }
 End If
} End Monitor_Adaptation
```

Figure 8. Algorithm for the monitor adaptation process

Figure 8 presents an algorithm in pseudo-code for the monitor adaptation process used in the framework. As shown in Figure 8, the process consists of searching in the repository for monitor rules that match semi-instantiated patterns. In the case that there are rules that fully match the semi-instantiated patterns, the process verifies if the time values in the rules are consistent with the response times of the SLAs for the respective operations and services. In positive case, the rules are maintained in the repository. Otherwise, the rules are modified with new time values according to the information in the SLAs.

Rule **Happens** (ic_Initial-Event, t1, R(t1,t1) and
Happens (ir_Event-User-Op, t2, R(t1,tn_1)) =>
Happens (ic_LEvent, tn, R(t1,tn_3)) tn > t2
Assumption **Happens** (ic_LEvent, t1, R(t1, t1)) =>
Initiates (ic_LEvent, *fluent*, t1)

Figure 9: Example of skill context pattern

Rule **Happens** (ic_start_CES, t1, R(t1,t1)) and
Happens (ir_artistsPerformance, t2, R(t1,tn_1)) and
Happens (ir_direction, t3, R(t1,tn_2)) =>
Happens (ic_professionalReview, t4, R(t1,tn_3)
t4>t2, t4>t3
Assumption **Happens** (ic_professionalReview, t1, R(t1,t1))
=> **Initiates**(ic_professionalReview,professionalReview,t1)

Figure 10: Example of semi-instantiated skill pattern

Rule **Happens** (ic_start_CES, t1, R(t1,t1)) and
Happens (ir_artistsPerformance,t2,R(t1,t1+3001)) and
Happens (ir_sceneryAdaptation,t3,R(t1,t1+4002)) and
Happens (ir_direction, t4, R(t1,t1+5003)) =>
Happens (ic_profesionalReview, t5, R(t1,6004)
t5>t2, t5>t3, t5>t4
Assumption **Happens** (ic_profesionalReview, t,R(t,t)) =>
Initiates (ic_profesionalReview,profesionalReview,t)

Figure 11: Example of rule R4

Rule **Happens** (ic_start_CES, t1, R(t1,t1)) and
Happens (ir_artistsPerformance, t2, R(t1,t1+3001)) and
Happens (ir_direction, t3, R(t1,t1+4503)) =>
Happens (ic_profesionalReview, t4, R(t1,6004)
t4>t2, t4>t3
Assumption **Happens** (ic_profesionalReview, t,R(t,t)) =>
Initiates(ic_profesionalReview, profesionalReview, t)

Figure 12: Example of rule R5

In the case in which there are rules in the repository that only match the invariant parts of the semi-instantiated patterns, the time values of the patterns are instantiated based on information from SLAs and new rules are created (fully-instantiated patterns). The process verifies for every identified rule in the repository that matches the semi-instantiated patterns, if the rule is a valid rule. This is done by traversing the service-based system specification and verifying if the information in the rule is still a valid path in the specification. In positive case, the time values for the rule are checked against related SLAs and adjusted if necessary. In negative case, the rule is removed from the repository. The new created rules are added in the repository.

In the case in which there are no monitor rules in the repository that match the semi-instantiated patterns, new rules are created by instantiating the time values of the semi-instantiated patterrns based on information from SLAs (fully-instantiated patterns). These new rules are added to the repository.

As an example, consider the CE_SBS described in Section 1. Suppose that an opera critic accesses the system to input a review for the performance of Carmen that is currently being presented at the London Royal Opera House. In this case, the skill pattern shown in Figure 9 is identified by the Rule Adaptor after receiving an event for context skill referring to *professional critic* (skillful), and the semi-instantiated pattern in Figure 10 is created based on information from the BPEL process. In Figure 9, the variant part of the pattern is shown in grey. Suppose rule R4 shown in Figure 11 in the repository that matches the invariant parts of the semi-instantiated pattern in Figure 10. Assume, that rule R4 is not a valid rule for the current version of the service-based system (operation *sceneryAdaptation* is not in the BPEL process anymore due to changes in the system). R4 is removed from the repository and rule R5 shown in Figure 12, with the time values identified from respective SLAs, is added to the repository.

4. IMPLEMENTATION ASPECTS AND EVALUATION

A prototype tool of the MADap framework has been implemented in Java to demonstrate and evaluate the framework. The tool can be used to adapt monitor rules specified in Event Calculus [27]. The prototype is designed to take as input an event representing one of the direct user context types (see Table 1) and deciding if a monitor rule could be identified, modified, created, or removed. Initial evaluation of the framework was executed to demonstrate that the framework could be used to identify, modify, create, and remove monitor rules based on user context types.

We evaluated the framework in an extension of the CE_SBS described in Section 1 with seven services, namely: S1:Ticket Purchase Service, S2:Payment Service, S3:Performance Information Service, S4:Performance Scheduling Service, S5:Resource Search and Allocation Service, S6:Reviewing Services, and S7:Catering Services. The evaluation was conducted for each direct user context type (role, skill, need, preferences, cognition) in five different cases, as described below.

Case 1: *Empty rule repository*
This case was considered to demonstrate the creation of new monitor rules and assumptions for each different context type. A total of 29 monitor rules and assumptions were created in the repository, broken down as follows:

8 rules/assumptions due to context type role,
6 rules/assumptions due to context type need,
4 rules/assumptions due to context type skill,
8 rules/assumptions due to context type cognition, and
3 rules/assumptions due to context type preferences.

The number of rules (and assumptions) for each context type is directly related to the different roles, needs, skills, cognition, and preference characteristics of the users of the service-based system, and the relevant functionalities of the system for the context types.

Case 2: *Rule repository with 100 rules not related to CE_SBS*
This case was also considered to demonstrate the creation of new monitor rules for each different context type given that all the 100 unrelated rules did not match fully or partially (matching of the invariant parts) the semi-instantiated patterns. As in Case 1 above, 29 monitor rules were created in the repository with the same number of rules for each context type.

Case 3: *Rule repository with all 29 relevant rules for CE_SBS*
This case was considered to verify if monitor rules that can be used for monitoring a service-based system at different stages of

its execution could be identified. For each of the different context types, the relevant rules where identified.

Case 4: *Rule repository with all 29 relevant rules for CE_SBS and with 100 rules not related to the system*
This case was also considered to verify if monitor rules that can be used for monitoring a service-based system at different stages of its execution could be identified. Despite the extra not related rules in the repository, as in Case 3 above, the relevant rules for each different context type were identified from the 29 relevant monitor rules in the repository.

Case 5: *Rule repository with 100 rules not related to the system, 27 rules relevant to the system, and 5 rules that match the invariant pattern parts for each context type*
In this case, the set of 27 relevant rules and 5 rules that match the invariant pattern parts are different for each context type. More specifically, for each context type, the 27 rules were selected from the set of 29 relevant rules by removing two rules for the specific context type. Given the mixture of the rules in the repository, in this case, the framework was able to identify the remaining rules for a certain context from the set of 27 relevant rules, to create the two rules that were initially removed from the set of relevant rules, and to identify 5 rules that match the invariant parts and verify if these rules were supposed to be removed from the repository or maintained.

This initial evaluation has demonstrated that the framework can adapt monitor rules for different user context types in the four proposed ways. It also shows that there is a direct correlation in the number of monitor rules and the different instances of the various context types for a service-based system. In addition, the approach supports the removal of monitor rules for those rules that become obsolete due to changes in the service-based system and the verification of this situation when trying to identify rules or create new rules in the repository.

5. RELATED WORK

In [9], an ontology for context-aware pervasive computing environments is proposed. This ontology is centered on general concepts including people and places. However, in this proposal all the elements are defined according to a specific scenario and most of the identified user context types are related to physical attributes. Contrary, in [16][23] ontologies have been formulated considering the user as the main element. Similar to our approach, in these ontologies a central class represents the user models and is associated to other classes concerned with other user characteristics such as skills or abilities. Our framework considers more specific types of user context, though

Different approaches have been proposed for monitoring service-based systems. In [25] monitoring is performed by checking assumptions and conditions, which specify how services participate in a composition and the conditions the composition must satisfy. Assumptions and conditions are specified as behavior properties in an expressive monitoring language. These behavior properties are used by a monitor to check the system operation, based on intercepted messages from the processes. A similar approach is taken in [30] where behavior is monitored by rules specified in EC. In this approach the rules are composed of assumptions and properties of a service-based system. In [4] monitoring is concerned with timeouts, runtime, and violation of functional contracts, which are described as monitoring rules. These monitoring rules are specified as comments in a BPEL process. When a service is invoked, its content is serialized, as an XML fragment, and sent together with the associated rules to another specially designed web service that acts as a monitor. In [15][24], the authors propose policy-based approaches to monitor web service compositions and management systems, respectively. In [15], the use of declarative policies provides flexibility and reusability when specifying a property. The work in [24] uses extracted information from policies to identify, configure, and instantiate management agents that will be used to monitor management systems. All the above approaches use rules or policies for monitoring the correct behavior of a system. The rules or policies are formulated to monitor particular behaviors and do not consider changes in a service-based system and user context characteristics.

Some work to support dynamic selection of monitoring rules based on context information has been proposed in [13][31]. The approach in [31] uses monitor manager on top of existing monitoring tools to provide a policy driven interface for these tools. The policies describe how the monitoring infrastructure should react in the presence of changes. The rules used by the monitoring tools are not modified.

In [29] patterns for monitoring security properties such as confidentiality, integrity and availability have been proposed. Similar to our work, these patterns are expressed in EC language. Another approach that uses EC to represent patterns to support verification of physical interaction is proposed in [17]. In this work, patterns are prerequisites for an effective physical interaction. In [13] the authors propose the use of a pattern-based approach to support presentation, codification, and reuse of property specification for finite-state verification. This work is used in [28] where temporal logic patterns for runtime monitoring of web service conversations are used.

Recently, a few approaches that support adaptation of service-based systems have started to appear [1][2][5][21][22][24]. The work in [5] proposes an approach towards self-healing for services compositions based on monitoring rules and reaction strategies. Another approach for self-healing is found in the PAWS framework [1] in which monitor and recovery actions are used. In [2] the authors present a context-aware adaptive service approach. The VieDAME framework [22] uses an aspect-oriented approach to allow adaptation of service-based systems for certain QoS criteria based on various alternative services. The work in [21] is based on augmenting service monitoring with online testing to identify possible failures in the system.

Although several approaches have been proposed for monitoring and adaptation of service-based systems, none of these approaches consider the need for adaptation of the monitor rules.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented a monitor adaptation framework (MADap) that supports the identification, modification, creation, and removal of monitor rules due to changes in the (i) service-based system or set of services used in the system, (ii) types of user interaction with the system, and (iii) context characteristics of the user interacting with the system. The framework considers different types of user context such as role, skill, need, preferences, and cognition. MADap is based on the use of monitor rule patterns for each different user context type. A prototype tool has been developed to illustrate and evaluate the framework. We are currently extending the set of patterns for the different types of user context. We are also conducting other evaluation of the

framework for more complex service-based systems. Moreover, we are analyzing the performance of the framework and how a monitor component could use the created, identified, and modified rules during run-time.

7. ACKNOWLEDGMENTS

The work reported in this paper has been funded by the European Community's 7th Framework Programme under the Network of Excellence S-Cube – Grant Agreement no. 215483.

8. REFERENCES

[1] Ardagna, D., Comuzzi M., Mussi, E., Pernici, B., Plebani, P. 2007. PAWS: A Framework for Executing Adaptive Web-Service Processes. *IEEE Softw.* 24, 6, 39-46.

[2] Autili, M., Di Benedetto, P. and Iverardi. P. 2009. Context-aware Adaptive Services: The PLASTIC Approach. In Proc. of the 12th International Conference on Fundamental Approaches to Software Engineering, FASE'09, York, UK.

[3] Barbon, F., Traverso, P., Pistore, M. and Trainotti, M. 2006. Run-Time Monitoring of Instances and Classes of Web Service Compositions, in IEEE International Conference on Web Services, ICWS'06, Chicago, USA.

[4] Baresi, L. and Guinea, S. 2005. Towards Dynamic Monitoring of WS-BPEL Processes. Third International Conference on Service Oriented Computing, ICSOC'05, Amsterdam, The Netherlands.

[5] Baresi, L., Ghezzi, C., Guinea, S. 2007. Towards Self-Healing Compositions of Services. Studies in Computational Intelligence, v. 42, Springer, Heidelberg.

[6] Betini, C., Maggiorini, D. and Riboni, D., 2007. Distributed Context Monitoring for the Adaptation of Continuous Services, In WWW Journal, Special Issue on Multichannel Adaptive Information Systems on WWW. Springer.

[7] BPEL4WS.http://www128.ibm.com/developerworks/ library/specification/ws-bpel/

[8] Brown A. and Ryan, M., 2009. Context-aware Monitoring of Untrusted Mobile Applications, Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec '09, Turin, Italy.

[9] Chen, H., Finin, T., Joshi, A. 2003. An Ontology for Context-Aware Pervasive Computing Environments. The Knowledge Engineering Review, v. 18 n. 3.

[10] Chen G. and Kotz, D., 2000. A Survey of Context-Aware Mobile Computing Research. Technical Report. Dartmouth College, Hanover, NH, USA.

[11] Dery-Pinna, A-M., Fierstone, J. and Picard, E. 2003. Component model and programming: A first step to manage human computer interaction adaptation. In Proc. of 5th Int. Symposium on Human-Computer Interaction with Mobile Devices and Services, MobileHCI'03, Udine, Italy.

[12] Dey A.K. and Abowd, G.D. 2000. The Context Toolkit: Aiding the Development of Context-Aware Applications. Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland.

[13] Dwyer, M.B., Avrunin, G.S. and Corbett, J.C. 1999. Patterns in Property Specifications for Finite-state Verification. 21st International Conference on Software Engineering, Los Angeles, California, USA.

[14] Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G. 2009. Model evolution by run-time parameter adaptation. Proc on the 31st International Conference on Software Engineering, Vancouver, Canada.

[15] Erradi, A., Maheshwari, P., Tosic, V. 2007. WS-Policy based Monitoring of Composite Web Services. Fifth European Conference on Web Services, ECOWS'07, Halle, (Saale), Germany.

[16] Golemati, M., Katifori, A., Vassilakis, C., Lepouras, G., Halatsis, C. 2007. Creating an Ontology for the User Profile: Method and Applications. In Proc. of the First IEEE International Conference on Research Challenges in Information Science, RCIS'07, Ouarzazate, Morocco.

[17] Ishikawa, F., Suleiman, B., Yamamoto, K. and Honiden, S. 2009. Physical interaction in pervasive computing: formal modeling, analysis and verification. International Conference on Pervasive Services, ICPS'09, London, UK.

[18] Ludwig, H., Dan, A. and Kearney, R. 2004 Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. In Second International Conference Service-Oriented Computing, ICSOC'04, New York, USA.

[19] Maiden N., editor. Codified Human-Computer Interaction (HCI) Knowledge and Context Factors. S-Cube project deliverable: PO-JRA-1.1.3 www.s-cube-network.eu/achievements-results/s-cube-deliverables.

[20] MADap: Monitor Adaptation Project. http://vega.soi.city.ac.uk/~abdw747/MADap

[21] Metzger, A., Pohl, K., Sammodi, O., Rzepka, M. 2010. Towards Proactive Adaptation with Confidence: Augmenting Service Monitoring with Online Testing. Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS'10, Cape Town, South Africa.

[22] Moser, O., Rosenberg, F. and Dustdar, S. 2008. Non-intrusive monitoring and service adaptation for WS-BPEL. In Proc. of WWW 2008, Beijin, China.

[23] Nébel, I., Smith, B., Paschke, R. 2003. A user profiling component with the aid of user ontologies. Proc. of workshop learning teaching knowledge adaptivity, Karlsruhe, Germany.

[24] Ouda, A., Lutfiyya, H., Bauer, M. 2010. Automatic Policy Mapping to Management System Configurations. IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY'10, Washington, USA.

[25] Pistore M. and Traverso P. 2007 Assumption-Based Composition and Monitoring of Web Services. In Test and Analysis of Web Services, Springer 2007.

[26] SCube. Software Services and Systems Network of Excellence. http://www.s-cube-network.eu/

[27] Shanahan, M. 1999. The event calculus explained. In Artificial Intelligence Today: recent trends and developments, Springer-Verlag, Berlin, Heidelberg, 1999.

[28] Simmonds, J., Chechik, M., Nejati, S., Litani, E., O'Farrel, B. 2008. Property Patterns for Runtime Monitoring of Web Service Conversations. In Runtime Verification, Springer-Verlag, Berlin, Heidelberg, 2008.

[29] Spanoudakis, G., Kloukinas, C. and Androutsopoulos, K., 2007. Towards security monitoring patterns. In Proc. of the ACM Symposium on Applied Computing, SAC'07, New York, NY.

[30] Spanoudakis, G., Mahbub, K., 2006. Non Intrusive Monitoring of Service Based Systems, In International Journal of Cooperative Information Systems, IJCIS'06.

[31] Talwar, V., Shankar, C., Rafaeli, S., Milojicic, D., Iyer, S., Farkas, K. and Chen, Y. 2006. Adaptive monitoring: Automated change management for monitoring systems. In 13th Workshop of the HP OpenView University Association, HP-OVUA, Cote d'Azur, France.

Appendix B

CLAM (Cross-layer Adaptation manager)

CLAM: Managing Cross-layer Adaptation in Service-Based Systems

Asli Zengin*, Annapaola Marconi*, Luciano Baresi†* and Marco Pistore*
*Fondazione Bruno Kessler – IRST, Trento, Italy
[zengin,marconi,baresi,pistore]@fbk.eu
†Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
baresi@elet.polimi.it

***Abstract*—Service-based systems (SBS) have a complex layered structure where the service-based application (SBA) is implemented through a composition of services, which run on top of service oriented infrastructures. Taking into account the heterogeneous and dynamic execution context of such complex systems, adaptation is not straightforward. While several state-of-the-art adaptation approaches, unaware of each other, target different problems at specific parts of the system, the isolated enactment of those adaptations results in ignoring the overall impact of the adaptation on the whole SBS. In this work, we propose an approach that introduces a cross-layer adaptation manager (CLAM) to tackle this issue. The approach relies on a cross-layer meta model of the SBS and a set of predefined domain specific rules to integrate and coordinate existing analysis and adaptation tools. It assesses the impact of an initial adaptation trigger at different system levels, and if needed, proposes additional adaptations, consistent with the overall system. The paper introduces the proposed approach and presents preliminary results on its first implementation with concrete analysis and adaptation tools.**

I. Introduction

The operation of service-based applications is a complex task. These distributed systems must provide their functionality with the required/agreed qualities of services, cope with the unreliable network on which they operate, and also deal with the changes in the context in which they are executed, or in the partner services with which they interact. This means that all the problems must be discovered as soon as they materialize, and the applications must be able to adapt their behavior to cope with them.

Adapting the behavior may mean changing the actual composition of services, or selecting different partner services. Moreover, the satisfaction of service level agreements in place may also impose changes in the way services are offered. For example the supervision system may renegotiate some quality parameters with the providers of the partner services, change the configuration of the engine that runs the composition (BPEL process), tune the platforms that provide some partner services, and even adjust the infrastructure (resources) used by the application and its partners.

This is to say that a problem (*adaptation need*), which usually materializes at application level, may trigger adaptations at any level of the "usual" service stack [1]: software (application), platform, and infrastructure. Moreover, some adaptations may trigger others, or they may influence some quality parameters, or even the operation, of parts of the system. For example, the selection of a new cheaper service could help decrease provisioning costs, but if the new service were slower than the previous one, it would have a negative impact on the performance of the new application. Similarly, to keep the performance agreed with clients, the same application, or some partner services, may require the provision of additional infrastructural resources to allocate all the different instances of the application and run them efficiently. Again, additional resources may increase the price of the application, and thus a viable solution must find a compromise between the two extremes.

Adaptation is thus a cross- and multi-layer problem, and at each layer it could also target different, possibly conflicting, system aspects. For example, when reorganizing the composition, one could privilege the application's price, its speed, or the compliance with some external regulations.

Many existing solutions [2]–[10] have addressed adaptation in a "local" way by only considering one system aspect at one layer; in contrast this paper fosters a comprehensive approach able to address different layers and aspects concurrently, reason on the dependencies and consequences among them, and identify global solutions. These solutions must harmonize layers and system aspects, and provide an integrated adaptation plan based on local activities.

The paper proposes a consistent and coherent solution for comprehensive adaptations based on CLAM (Cross-Layer Adaptation Manager). CLAM relies on a comprehensive high-level model of the application and of the layers behind it. Each model element is associated with a set of analyzers to understand the problem, solvers, to identify possible solutions, and enactors, to apply them on the element. The coordinated operation of analyzers, solvers, and enactors is governed by predefined rules that identify the dependencies, and consequences, between the elements of the model and run the different tools. For each adaptation need, CLAM produces a tree of alternative adaptations, identifies the most convenient one, and applies it. The paper exemplifies all main concepts through a simple application for the smart management of taxi reservations.

The rest of the paper is organized as follows. Section II motivates and explains the need for holistic, cross-layer adaptations. Section III introduces the approach, Section IV presents the system modeling, followed by the descrip-

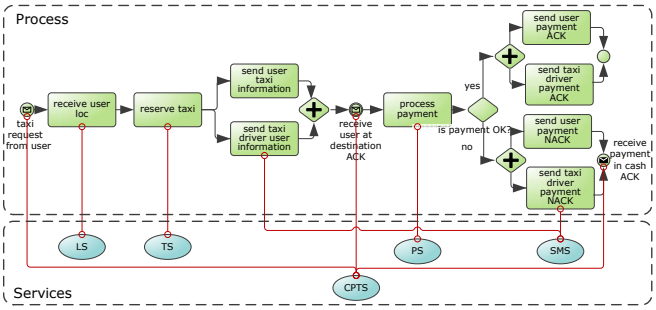


Figure 1. Call & Pay Taxi Application

tion of CLAM in Section V. Next, Section VI presents a first implementation of the concepts, and an assessment of proposed solution. Section VII surveys related approaches, while Section VIII concludes the paper.

II. Problem Statement

In this section we present a scenario to illustrate the cross-layer adaptation problem addressed in this paper. Later it will be used also to explain our approach.

Our scenario, "Call & Pay Taxi", is a service-based system (Figure 1) and composed of the following three layers: *application layer* where the application is run and monitored with respect to the key performance indicators (KPI) of interest, *service layer*, which corresponds to the partner services of the process provided by different platforms, and finally the underlying *infrastructure layer* for the composite service (application) and the partner services.

The layers of our running scenario comprise the following factors:

- *Application layer:* "Call & Pay Taxi" composite service (CPTS), implemented as a BPEL process.
- *Service layer:* a short messaging service (SMS), a location service (LS) and a payment service (PS) provided by the telecom company, and the taxi service (TS) provided by the taxi company.
- *Infrastructure layer:* The underlying platforms on top of which CPTS, SMS, LS, PS and TS run. E.g., workflow engines, the application servers, hardware resources.

In CPTS, the client requests a taxi by sending a text message (SMS) to the application. Then, her location is identified and the taxi company is contacted to organize the actual taxi service. After transporting the user to destination, the process terminates with a successful payment.

On this scenario we show an adaptation case to motivate our problem: The CPTS provider decides to reduce the overall cost and consults a business analyst. Given that technologically it is possible to replace services in the process, the business analyst decides to switch to a cheaper telecom infrastructure. This means replacing SMS, LS and PS in the BPEL process. However, there is a problem with the new LS service's output message format. It provides the client's location in geographical coordinates instead of the full address while in our application design we use full address as the input message to the taxi service. To address this new data mismatch problem, the service composition is adapted by adding a mediator service in the workflow. Basically it converts geographical coordinates into full addresses. Yet it triggers a new problem: We notice that the new service we introduce for data mediation is too costly and in fact increases the overall cost of the process in an unforeseen way.

The new workflow is consistent but the actual goal that led to the substitution of the telecom provider, that is cost reduction, is not met. The updated version of CPTS is more expensive than the original one.

Problem. With the existing approaches when an adaptation is performed, it targets a particular problem occurring at a specific SBS layer. Thus, they tend to propose local solutions to local problems in a way that is isolated from the overall application context. As we can see in our example, this practice may result in (i) new adaptation triggers that are not easy to anticipate, and (ii) undesirable consequences that the eventual adaptation gets useless with respect to our initial intention. To avoid such problems, we must understand the impact of a change across different layers, which have their own characteristics and constraints. In our example while we are trying to improve the cost, one of the application KPIs, we do not know the consequences of replacing the services of telecom provider at all the layers. Consequently, we need an approach to meet the following requirements [11]:

R1. identify the problems that might occur at all the SBS layers due to an adaptation.

R2. if there exist problems, tackle them by proposing new adaptations that are consistent with the overall system.

We believe that addressing this problem is not trivial since we must consider the complex, layered structure of the service-based systems for which we have numerous existing analysis and adaptation approaches proposed and practised independently of each other.

III. The Approach

In service-based systems the usual way to adapt is first to analyze the problem, then to identify a solution (adaptation), and finally to enact it in the system. For each of them, there are already several approaches proposed in the state-of-the-art. However, as we discussed in the previous section, this traditional way to adapt is not always a good practice. Therefore, in our solution we would like to change this conventional order of the adaptation process and orchestrate these existing mechanisms in the CLAM platform to enable cross-layer adaptation. The main idea is to reuse them in order to analyze the impact of an adaptation trigger and to propagate it if required. In this way we can prevent conflicting adaptations and produce a final, validated adaptation strategy aligned with the overall SBS.

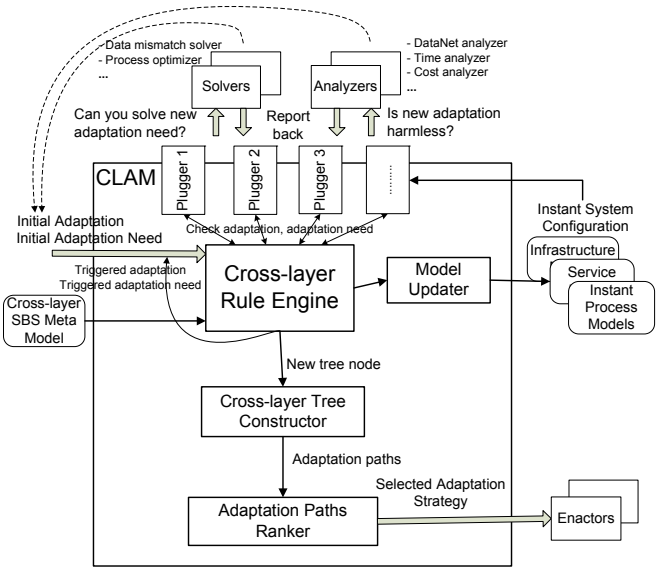


Figure 2. CLAM: Cross-layer Adaptation Manager

Our solution is based on three pillars: (i) an integrated platform where we plug existing adaptation and analysis tools, (ii) a cross-layer meta model of the SBS, which is a fundamental input of the overall impact analysis, (iii) a rule-based analysis methodology for cross-layer adaptation.

We call each of the external mechanisms, which we integrate to the CLAM platform through *pluggers*, a "component". They can be analyzers, solvers, or enactors. *Analyzers* check the compatibility of an SBS configuration with respect to some local constraints of the system. *Solvers* propose adaptations to the specific problems (adaptation needs) identified by the analyzers. *Enactors* apply and implement the proposed adaptations in the system.

Figure 2 presents the architecture of CLAM to help the reader understand how the whole proposal works. When a new adaptation/adaptation need is triggered by one of the solvers/analyzers, CLAM gets activated to initiate the impact analysis. In order to figure out which parts of the system are affected due to a new trigger, CLAM needs a *Cross-layer SBS Meta Model* where the SBS layer dependencies are explicitly presented. Then for the affected system parts it looks up in its *predefined rules* that are stored in the *cross-layer rule engine*, and it identifies the relevant components to invoke.

In case of receiving an adaptation trigger, CLAM identifies a set of analyzers to validate the new *system configuration* regarding the affected system parts. Since analyzers are associated with a specific part of the system, they perform their own local analysis based on local constraints. Later they produce a report for CLAM whether the proposed new configuration is OK or not.

When an analyzer reports back a problem with the new configuration, it means it is a new adaptation need that must be addressed by a solver. In this case, similarly the rule engine identifies a proper solver and invokes it. If the solver reports back an adaptation to solve the problem, it indicates that the problem, which is arisen from the previous analyzer invocation, is addressed and we can continue our process until all the identified components are invoked and all the problems arisen on the path are tackled iteratively. Instead, if at some point no adaptation is proposed by a solver, we have to stop the process at that point, and it implies that the initial trigger that CLAM received could not be validated, thus cannot be enacted in the SBS.

During the whole process, each time the rule engine receives a report it contacts *cross-layer tree constructor* to enable the incremental construction of the cross-layer adaptation tree where branches keep alternative adaptation paths that can address the negative impacts of an initial adaptation.

In the final step, the constructed tree is passed to the *adaptation paths ranker* so that one of the validated alternative paths can be selected as a complete adaptation plan and deployed in the SBS through enactors.

IV. System Meta Model

SBS's layered structure already contains implicitly the dependencies among the elements of the different layers, while those dependencies are not trivial among adaptations coming from different layers. We would like to benefit from the layer dependencies and create the meta model of the SBS in a cross-layer manner so that they get explicit to be easily used by CLAM.

The meta model is not hard-coded in the tool, but it must be provided by the user. In this paper, we use the usual hierarchical representation of an SBS, along with the meta model of Figure 3.

The meta model is created at design time as follows:

- *System layers.* In this case, we have layers: application, service, and infrastructure, each of them with its characteristics. CLAM can also consider different layers, elements and characteristics for other application domains. E.g., for a security critical SOA application, security might be considered as a separate, new layer.
- *Layer elements.* For our scenario, at each layer we can talk about stakeholders, structure, internals and quality aspects. The stakeholders might be service or infrastructure providers at different layers. The structures are the process, partner services and the infrastructure that is composed of an application server and the underlying resources. The layer internals are process activities for the process, service operations for the services and the infrastructural ingredients for the infrastructure. Considering quality, we can talk about KPIs at application level, and QoSs at service and infrastructure levels.
- *Relations among elements.* The graph edges display the dependency between the different system elements. We distinguish two types of dependency relations: *has* and *consumes*. While *has* can only be a relation between

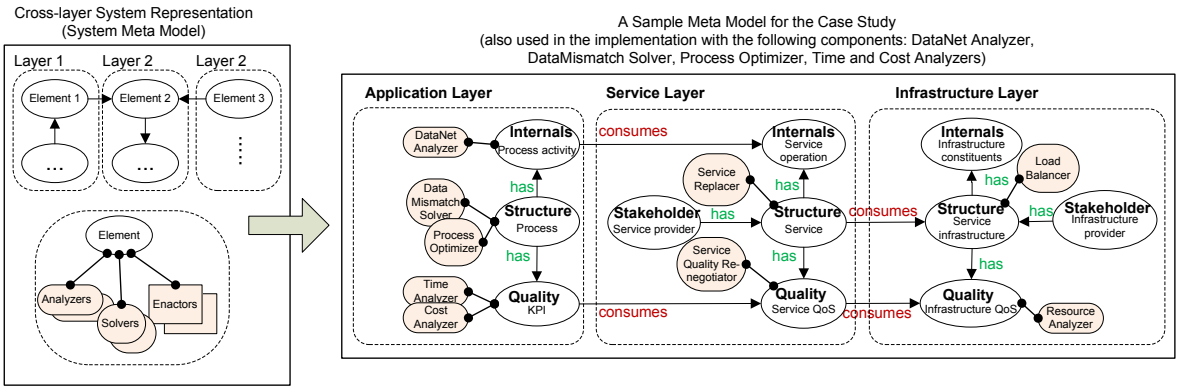


Figure 3. System Modeling for CLAM

two elements of the same layer; *consumes* defines the inter-layer relations where an element from a layer relies on another element from a different layer. For example, at infrastructure level, the provider *has* the infrastructure and the infrastructure *has* QoS attributes. On the other hand, since services are running on top of infrastructures, naturally the structure element from the service layer has a *consumes* relation with the structure element from the infrastructure layer. Similarly, the availability of a service depends on the availability of the infrastructure on top of which it is run. Then, naturally there is a *consumes* relation between service QoS and infrastructure QoS.

- *Components associated with system elements.* For each system element we have analyzers, solvers and enactors. An analyzer gets an adaptation proposal from a solver and validates it for the local constraints related to the system element that it is associated with, and produces an output to state whether the constraints hold or they are violated by the given proposal. For example, a time analyzer associated with a KPI node in the model can check the compatibility of a modified process with respect to the execution time to see whether it is still in the desired range. A constraint violation that is detected by an analyzer is a newly triggered adaptation need that must be addressed by a proper solver. Thus, a solver gets an adaptation need and produces an adaptation action or a set of alternative actions to handle the need. For instance, if we had a problem with process execution time, we might try to parallelize some process activities by using a process optimizer as a solver, or we might try to re-negotiate the response time of partner services at QoS level. On the other hand, enactors are invoked only in the end when the overall CLAM analysis is carried out and a selected adaptation path is to be deployed. For example, a process migrator is an enactor that can migrate the running process instances to a new process model.

The system meta model serves us in two ways: (i) navigating the graph to reason on the overall analysis (ii) instantiating the model with concrete value assignments to the elements (we call them system configurations, e.g. M_0, M_1...) to keep track of alternative adaptation actions during the analysis. E.g., for our reference scenario, CPTS, we can instantiate a system configuration with the current BPEL file in use for the execution, partner services SMS, LS, PS that are provided by the current telecom provider X, and the TS that is provided by RadioTaxi Trento. Similarly we can instantiate each graph node with concrete elements and overall graph instantiation will correspond to a system configuration (i.e., an M_i).

V. CLAM Platform

Our solution proposes a comprehensive impact analysis of an adaptation/adaptation need triggered in the SBS. In this section we introduce CLAM rules and present a comprehensive description of this impact analysis, which we also call "CLAM analysis".

First, we would like to elaborate the concepts related to the adaptation:

Adaptation need, produced by an analyzer, is a compatibility problem of an SBS instance considering a local constraint imposed by a system element. Adaptation needs are predefined and associated with system elements at design time. For example, if the current system configuration is violating cost KPI, a cost analyzer would report the incompatibility that cost should be reduced.

Adaptation action is produced by a solver as a set of changes on system elements in the current system configuration. A change could be "modifying", "replacing", "adding" or "removing" a concrete element in the cross-layer SBS model. If we introduced a data mediator service sID to the composition pID, a data mismatch solver would produce the action: "add service sID; modify process pID" to solve the mismatch problem caused by the new LS. In general, a solver can produce a set of alternative adaptation actions to the given adaptation need.

Adaptation strategy is a sequence of adaptation actions validated by CLAM. Validation is either through the ap-

proval of the new configuration obtained through the proposed adaptation, or if it is not the case, through identifying the new problem (i.e., adaptation need) and proposing an additional adaptation action that solves this problem.

CLAM Rules. Analyzers, solvers, and enactors share a common language to deal with adaptation. An analyzer gets a local constraint and the relevant elements of the current configuration as input and produces an adaptation need as output, a solver gets an adaptation need and similarly the relevant elements of the current configuration as input and proposes a set of alternative adaptation actions as output, while an enactor gets an adaptation action to be applied to the system. These relations are governed by the following rules:

1. Analyzer Rules help decide which analyzers to invoke when there is a change in a system element due to an adaptation. These rules associate analyzers with system elements.

E.g., *process* $\rightarrow$ *cost analyzer, time analyzer*

2. Solver Rules help decide which solvers to invoke when an adaptation need is produced by an analyzer. These rules associate a set of solvers with each adaptation need defined in the system.

E.g., *reduce process time* $\rightarrow$ *process optimizer, service quality re-negotiator, service replacer*

3. Enactor Rules help decide which enactors to invoke when an adaptation action is to be deployed in SBS. Similarly these rules associate a set of enactors with each adaptation action included in the system.

E.g., *add service* $\rightarrow$ *dynamic service binder*

Whenever we add a new component in CLAM, we need to (i) define a new set of rules and update the existing ones if necessary, (ii) enable its integration to the platform by adapting its interface through the CLAM plugger, and (iii) assign a priority to it. This is both to organize the invocation order when there is more than one component and to have preferences over the components with the same functionality.

When analyzers/solvers return some results, CLAM needs to keep track of new adaptation needs/adaptation actions both to iteratively continue its analysis and to incrementally build the adaptation strategies. The analysis is accomplished based on the rules, and the adaptation strategies are built and kept through the construction of a cross-layer adaptation tree, T. Each time an adaptation need/adaptation action triggers CLAM, it creates a new tree T. The results produced by a component motivates a new edge E, which contains the status of the report, and a new node N, which contains the newly obtained configuration of the system and the queue of remaining components to invoke (Figure 4).

CLAM Algorithm. Let us see how CLAM performs the impact analysis. We distinguish three main parts in the algorithm:

1. Components Queue This part manages the dynamic queue that keeps an up-to-date set of components to be utilized during the whole analysis.

- *Adding components.* When there exists a set of new components to be invoked, we get the priorities of components and we update the queue by adding the new components.
- *Deleting components.* When a component invocation is completed (report is received), the component is deleted from the queue.
- *Empty queue.* When there is no component left in the queue to invoke, we terminate the analysis.

2. Components Discovery Through the execution of CLAM rules, this part clarifies when we need to find an analyzer (or analyzers) and how to do it, and similarly when we need to find a solver and how to do it.

- *Analyzer discovery.* When we receive a set of alternative adaptation actions from a solver, for each alternative, we identify the system elements it would modify, and mark them as "changed". Next, we identify the elements that have a "consumes" relation with the changed ones, and mark them as "affected". Finally, we search rules and get the analyzers associated with all the changed and affected elements.
- *Solver discovery.* When we receive an adaptation need from an analyzer, we search rules and get the set of solvers connected to this adaptation need.

3. Tree Construction This part describes how we gradually construct the tree upon receiving component reports.

- *Positive analyzer report.* When CLAM receives a positive report from an analyzer, which means for a given system configuration the relevant constraint is satisfied, we update the queue, get the current node of the tree, and append the new edge, which contains the report status and then the new node, which contains the same system configuration and the updated queue. If the updated queue is empty, we mark this newly created node as a green leaf and terminate the analysis. If the queue is not empty, we set the newly created node as the current node, and call the next component in the queue and wait for a new report.
- *Negative analyzer report with a new adaptation need.* When CLAM receives a new adaptation need from an analyzer, we update the queue, and identify the solvers that may allow us to satisfy the adaptation requirement. If no solver is found, we mark the current node as a red leaf and terminate the analysis for that path. If more than one solvers are found, we select the one with the highest priority, and add it to the queue. Next, we get the current node, and similarly create and append the new edge and new node, and continue the analysis by invoking the next component in the queue.
- *Negative analyzer report.* When CLAM receives a negative report from an analyzer and no adaptation need is proposed, we mark the current tree node as a red leaf and terminate the analysis for that path.

- *Positive solver report.* When CLAM receives a new set of alternative adaptation actions from a solver, for each adaptation, we update the queue and the configuration by including the elements that would be changed by the adaptation. Then, we identify the analyzers needed for validation. Next, we add the identified analyzers to the queue, update the tree with the new node, and continue the analysis with the next component.
- *Negative solver report.* When CLAM receives a negative report from a solver, i.e., no adaptation is proposed, we terminate the analysis.

After constructing the tree, CLAM extracts the paths from the root to the green leaves, each of which is an alternative adaptation strategy. It selects one strategy based on quality criteria and invokes the relevant enactors to apply its adaptation actions to the running system.

VI. Implementation

We implemented the first version of CLAM platform in Java where we defined the rules using JBoss Drools - The Business Logic integration Platform[1]. In order to use the platform, we need to provide as input (i) the system meta model which is created at design time by the user, (ii) the components to be used during the CLAM analysis. Once components are decided, we need to integrate them with the platform as described in the previous section.

The current implementation uses the following state-of-the-art analyzers and solvers: time and cost analyzers associated with the element quality, dataNet analyzer associated with the element process activity at the application layer, process optimizer associated with the adaptation need "reduce time" and data mismatch solver associated with the need "remove data mismatch". Currently, CLAM integrates and coordinates all these tools, but other could be easily plugged as well.

In this paper we focused on how to perform the CLAM impact analysis rather than the deployment of adaptation results in the SBS. Thus, current implementation does not have enactors integrated with CLAM.

The tools used in the implementation are:

QoS4BPEL is used for both time and cost analyses in our platform [12]. It gets as input the BPEL file of the application and the execution times and costs of each process activity in the BPEL, then it produces an aggregate value for them. Then the aggregate values can easily be compared with KPI target values to detect if a KPI is violated or not regarding a new process model.

Mismatch Patterns and Adaptation Aspects is used as both dataNet analyzer and data mismatch solver [10]. In dataNet analysis, whenever a new service is introduced to the composition, the compatibility of its interface is checked against the data flow requirements of the process. The tool gets as input the BPEL file and the WSDL file

[1]http://www.jboss.org/drools

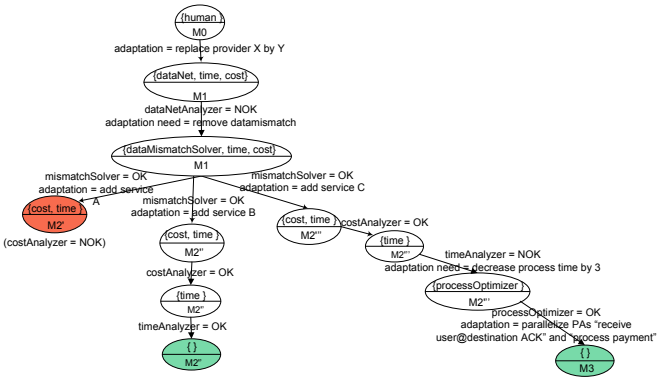


Figure 4. CLAM Tree for Alternative Adaptation Paths

of the new service and then performs the analysis based on some predefined mismatch patterns for the application. If a mismatch is identified, it is also capable of producing an adaptor, i.e., a data mediator, which can be introduced to the composition as a service.

Structural Adjustment of BPEL is used as process optimizer to reduce the execution time of a process [13]. While in the case study we used it for time optimization, the tool is capable of optimizing the memory usage of the process as well. It gets as input the BPEL file and produces an optimized BPEL if any task parallelization is possible.

A. CLAM at Work

The resultant tree that CLAM produces for our case study is given in Figure 4. When we receive the new adaptation action "replace telecom provider X by Y", first we update the system configuration from M_0 to M_1 since we now have the SMS, LS and PS of the new telecom provider together with their new QoS attributes. Then, we identify the changed and affected system elements. Changed elements are: service provider, service, service operation and service QoS (due to *has* relation) and those affected are: process activity and KPI (due to *consumes* relation). Then, we check the analyzers connected to these elements and identify dataNet, time and cost analyzers. Next, these new analyzers to be invoked are put in the queue and ordered according to their invocation priorities. Since now we have the updated system configuration and the updated queue, we can add the new node to the tree and invoke the next component from the queue, in this case the dataNet analyzer. It reports that the analysis is not OK, there is a data mismatch problem for the LS of the new telecom provider. We find the matching solver for the new adaptation need and in this way we continue to construct the tree gradually. A tree path is terminated when we end up with either an empty queue (green leaf) or a negative analyzer/solver report (red leaf).

B. Evaluation and Discussion

This section provides a first attempt to assess CLAM and to discuss the preliminary results obtained on the case study.

This is not a complete and thorough empirical assessment of the work done, but it provides some interesting indicators to the reader:

- *Time performance:* The construction of the tree of Figure 4 took around 3 seconds on a laptop with a 2 GHz Intel Pentium M Processor, Windows XP, and 1 GB RAM. In the future we would like to observe the change in the execution time by including both more adaptation cases where several trees of different lengths are produced, and more components integrated. Even though the current version of CLAM works offline, these figures provide a preliminary idea of how CLAM would perform at runtime. However, both overall execution time and run-time usability also depends on the performance of the tools plugged in CLAM.
- *Loop problem:* In the computation of the adaptation tree we had to deal with the problem of infinite paths. First, different sequences of adaptations, i.e., different tree paths, might bring the SBS to the same configuration (M_i). In the current implementation, we track the M_is that we keep in the nodes. If after a sequence of iterations, we reach the same configuration, we stop there. Second, we might have infinite number of adaptations, each of which brings the SBS to a new M_i. We address this problem by setting as a system parameter the maximum number of adaptation triggers on a tree path.
- *Integrity and extensibility:* CLAM provides a coordinated platform for tools, designed in isolation from each other and that they do not know each other, they do not care about other system elements, which might be affected due to a modification that they make on the system. In such cases an adaptation can harm a system element while it is improving another one. Moreover, plugging-in a new tool in CLAM is not expensive. Upon learning the input and output data of the new component, the CLAM designer should define and update the relevant rules, assign a priority to the tool, and instantiate a new plugger for the integration with the platform. This will make the tool visible to the rule engine such that it can be taken into account during the analyses.
- *Correctness and completeness:* For the time being we can state the following: If we assume that (i) the existing state-of-the-art tools that are in use by our platform run correctly and (ii) that CLAM is fed with the complete system model and with a complete set of analyzers and solvers; our approach is correct in the sense that it searches for and, if possible, finds a complete adaptation strategy, which is consistent with the whole set of local constraints in the SBS.

VII. Related Work

Existing work for SBS adaptation mostly focuses on specific aspects of the application where the adaptation problem is solved in a narrow scope, without taking into account its consequences for the whole SBS stack. BPM adaptation [2], [3], dynamic service binding [4], [5], self adaptation and self healing systems [6], [7], QoS-awareness [8], [9], mediator design for service interactions [10] and finally context-awareness [14] are relevant aspects. In principle, those approaches can serve our platform as solver components and in this way they can be aligned with all the SBS elements and be more effective.

Yet, there are few approaches in literature that propose the use of cross-relations for adaptation. In [15] coordinated adaptation is introduced for multiple applications interacting with each other in the same environment. These applications are not composed, but rather single entities which are affected by the same contextual attributes such as sharing common resources. The authors claim that a coordinated platform is important before adapting these applications to the context changes, since they have interactions in the same environment. This problem is addressed in a narrow scope, only in terms of management of shared resources. Another problem they mention is conflict resolution for two adaptation mechanisms. However, they expect the user to perceive the conflict, and modify the mechanisms accordingly.

[16] analyzes the dependencies of KPIs on process quality factors from different functional levels of an SBA such as QoS parameters, and then an adaptation strategy is decided to improve all the negatively affected quality metrics in the SBA. This work proposes a set of adaptations for KPI violations through the consideration of the non-functional layer dependencies. However, the work is at preliminary stage and the rules how to tackle violations are unclear.

Another approach is [17], which presents a framework for cross-layer adaptation of service oriented applications that comprise organization, coordination and service layers. The application stack is similar to the SBS stack, which we introduce in this paper. They propose a technique where the cross-layer adaptation designer prepares the taxonomies of adaptation mismatches, and later designs the adaptation templates, also known as patterns, that define generic solutions to tackle mismatches. Their technique directly models cross-layer adaptation templates, i.e., their dependencies are known from design time. These are like fixed patterns for cross-layer adaptation cases. Instead, we model the application in a cross-layer fashion and then discover the cross-layer adaptation paths on the fly through the coordination of available tools and mechanisms.

Like our approach, [18]–[20] have a cross-layer representation of the application model. While [19], [20] target a limited number of adaptation cases such as service replacement, [18] makes use of the cross-layer model for monitoring and analysis rather than adaptation.

VIII. Conclusions and Future Work

In this paper we presented CLAM, a holistic SBS management framework that can deal with cross- and multi-

layer adaptation problems. This is achieved in two ways: on the one hand CLAM identifies the system elements affected by the adaptation actions, and on the other hand it identifies an adaptation strategy that solves the adaptation problem by properly coordinating a set of components, i.e., analyzers and solvers integrated in the system. The proposed solution relies on a comprehensive high level system model that allows to easily extend the framework by plugging-in new components. We implemented the proposed framework and evaluated it on a service-based application for taxi reservation.

In the future we will enhance the criteria for the selection of the best adaptation strategy with costs for adaptation enactments. Moreover, we plan to address the issue of analyzing multiple adaptations simultaneously instead of analyzing a single adaptation case. In the meanwhile, we will continue to evaluate the framework on application scenarios entailing higher level of complexity and requiring to deal with different system layers and concerns (e.g. Cloud-based applications).

ACKNOWLEDGMENT

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).

REFERENCES

[1] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[2] L. T. Ly, S. Rinderle, and P. Dadam, "Integration and verification of semantic constraints in adaptive process management systems," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 3–23, 2008.

[3] A. Brogi and R. Popescu, "Automated Generation of BPEL Adapters," in *ICSOC 2006:Service-Oriented Computing*, pp. 27–39, 2006.

[4] K. Verma, K. Gomadam, A. Sheth, J. Miller, and Z. Wu, "The meteor-s approach for configuring and executing dynamic web processes," *LSDIS METEOR-S project. http://lsdis. cs. uga. edu/projects/meteor-s/techRep6-24-05. pdf. Technical report*, 2005.

[5] M. Colombo, E. D. Nitto, and M. Mauri, "SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules," in *In ICSOC06*, pp. 191–202, 2006.

[6] M. F. L. Console, "Ws-diamond: An approach to web services - diagnosibility, monitoring and diagnosis," 2007.

[7] A. Charfi, T. Dinkelaker, and M. Mezini, "A plug-in architecture for self-adaptive web service compositions," in *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, (Washington, DC, USA), pp. 35–42, IEEE Computer Society, 2009.

[8] E. Nitto, M. Penta, A. Gambi, G. Ripa, and M. L. Villani, "Negotiation of service level agreements: An architecture and a search-based approach," in *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, (Berlin, Heidelberg), pp. 295–306, Springer-Verlag, 2007.

[9] K. Christos, C. Vassilakis, E. Rouvas, and P. Georgiadis, "Qos-driven adaptation of bpel scenario execution," in *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, (Washington, DC, USA), pp. 271–278, IEEE Computer Society, 2009.

[10] W. Kongdenfha, H. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul, "Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters," *IEEE Transactions on Services Computing*, pp. 94–107, 2009.

[11] A. Zengin, R. Kazhamiakin, and M. Pistore, "Clam: Cross-layer management of adaptation decisions for service-based applications," in *2011 IEEE International Conference on Web Services*, pp. 698–699, IEEE, 2011.

[12] M. Dumas, L. García-Bañuelos, A. Polyvyanyy, Y. Yang, and L. Zhang, "Aggregate quality of service computation for composite services," *Service-Oriented Computing*, pp. 213–227, 2010.

[13] N. Rasadka and A. Marconi, "Optimizing bpel compositions via automatic process re-writing," *Technical Report*, 2011.

[14] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "Paws: A framework for executing adaptive web-service processes," *IEEE Softw.*, vol. 24, no. 6, pp. 39–46, 2007.

[15] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday, "An architecture for the effective support of adaptive context-aware applications," in *Proceedings of Mobile Data Management (MDM'01)*, (Berlin), pp. 15–26, Springer, January 2001.

[16] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann, "Adaptation of service-based applications based on process quality factor analysis," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pp. 395–404, Springer, 2010.

[17] R. Popescu, A. Staikopoulos, P. Liu, A. Brogi, and S. Clarke, "Taxonomy-driven adaptation of multi-layer applications using templates," in *SASO'10*, pp. 213 –222, 2010.

[18] L. Baresi, M. Caporuscio, C. Ghezzi, and S. Guinea, "Model-Driven Management of Services," in *ECOWS'10*, pp. 147–154, 2010.

[19] U. Tripathi, K. Hinkelmann, and D. Feldkamp, "Life cycle for change management in business processes using semantic technologies," *Journal of Computers*, vol. 3, no. 1, p. 24, 2008.

[20] B. Burgstaller, D. Dhungana, X. Franch, P. Grunbacher, L. López, J. Marco, M. Oriol, R. Stockhammer, J. Universitat, and J. Universitat, "Monitoring and Adaptation of Service-oriented Systems with Goal and Variability Models," tech. rep., Universitat Politècnica de Catalunya, 2008.

Appendix C

Adaptation of Service-based Business Processes by Context-Aware Replanning

Adaptation of Service-based Business Processes by Context-Aware Replanning

Antonio Bucchiarone, Marco Pistore and Heorhi Raik
Fondazione Bruno Kessler
Via Sommarive, 18, Trento TN 38100, Italy
{*bucchiarone,pistore,raik*}*@fbk.eu*

Raman Kazhamiakin
SayService srl
Via alla Cascata, 56C, Trento TN 38100, Italy
raman@sayservice.it

***Abstract*—Service-based business processes are typically used by organizations to achieve business goals through the coordinated execution of a set of activities implemented as services and service compositions. Since they are executed in dynamic, open and non-deterministic environments, business processes often need to be adapted to exogenous context changes and execution problems. In this paper we provide an adaptation approach that can automatically adapt business processes to run-time context changes that impede achievement of a business goal. We define a formal framework that adopts planning techniques to automatically derive necessary adaptation activities on demand. The adaptation consists in identifying recovery activities that guarantee that the execution of a business process can be successfully resumed and, as a consequence, the business goals are achieved. The solution proposed is evaluated on a real-world scenario from the logistics domain.**

***Keywords*-service-based applications; adaptation; service composition; context;**

I. Introduction

In recent years, service-oriented architectures have been widely used for the realization of complex business processes. In such processes, activities are realized through the invocation of a set of available services (both software services and human-based services).

Modern business processes often operate in dynamic, open and non-deterministic environments. This means that the execution context is volatile, and the outcome of some activities is not completely controllable. In addition to this, the set of available services and the set of business policies is also changing dynamically. Dynamic context changes or undesirable outcome of some activities may often cause abnormal termination of the process and prevent the achievement of the business goals. The solution might be the run time modification (or *adaptation*) of the basic process so that it can properly react to such extraordinary situations.

The most trivial approach to achieve adaptation consists in analyzing at design time all the extraordinary situations, developing corresponding recovery activities, and embedding them into a reference process, using standard mechanisms such as exception handling [7] or dedicated mechanisms for encoding adaptation into business process languages [13], [16]. Still, such *built-in* adaptation can hardly be used for complex processes, where adaptation cases may be too many and require too complex recovery activities to be "manually" implemented at design-time. It also provides very poor scalability since even minor changes in the execution environment (e.g., business policies are changed, or new services and capabilities are offered by service providers) may require a substantial re-design and re-implementation of recovery activities.

In order to address these limitations of built-in adaptation, many approaches perform the adaptation at run time, when a specific problem and a specific context for adaptation are defined. The vast majority of these approaches can be characterized as *rule-based*, since they predefine rules on how to transform a basic process in extraordinary situations. Some approaches introduce transformation rules explicitly [5], [15], [19], others define process variants [10], [12], [11] or use aspect-oriented methodologies [14], [1], [11]. Rules are more general than built-in adaptation since, while being written at design time, they are applied to a targeted process at run time, when more information on the extraordinary situation and execution context is available. This also results in much better scalability.

Rule-based approaches, however, still share a major drawback with built-in approaches. Since the rules are specified at design time, the designer has to choose a particular adaptation tactic for a certain extraordinary situation. At run time, it may happen that the tactic chosen is not applicable (e.g., if it requires the usage of a service that became unavailable), or a better tactic could be available (e.g., newly appeared services may allow for better recovery approaches). Maintaining an updated and consistent set of adaptation rules is a very time-consuming and error-prone task that requires profound knowledge of the execution environment, as well as advanced supporting tools. Moreover, sophisticated adaptation tactics usually require considerable amount of rules to be specified.

In this work, we propose a *goal-based* approach to business process adaptation in service-based applications. In our approach, adaptation activities are not explicitly represented. Instead, we build a formal framework that enables dynamic derivation of adaptation activities considering all aspects of the environment (current context, business policies, available services etc.) To support this, we 1) define a formal model for services and for service-based applications that enables adaptation. The formal model is based on the notion of

goals, and on service annotations that describe how services contribute to goal achievement; and 2) propose an execution engine that observes process execution and triggers adaptation on demand. In other words, the execution engine detects extraordinary situations that require adaptation and figures out adaptation goals. The actual adaptation of the business process is based on service composition via automated planning technique [2], [6]: when triggered by the execution engine, the planning algorithm [6] generates a composition of available services that achieves the adaptation goals in compliance with business policies.

Unlike rule-based approaches, our goal-based approach guarantees that if solutions exist they will be found automatically, without involving design activities by process analysts. In this regard, it overcomes the limitations of the rule-based approaches discussed above. Compared to other existing solutions, we also expect it to provide better flexibility, scalability and designer's productivity when applied to the business processes acting in extremely dynamic environments.

The rest of the paper is structured in the following way. In Section II we present our motivating example and discuss the main challenges we face. While Section III provides an overall picture of our approach, Section IV gives its formalization details. Section V is devoted to the evaluation of the algorithm performance and adaptation modelling overhead. In Section VI we provide profound comparison with other dynamic approaches and discuss a few open issues we plan to address in the future.

II. Motivating Example

In order to illustrate the problems addressed in the paper and to evaluate our approach we use a case study from the logistics domain. The case study is inspired by the operation of the sea port of Bremen, Germany [3]. The port receives ships loaded with cars and has to organize the delivery of the cars to retailers. Before delivered, each car goes through a number of steps such as unloading, registration, storing, different types of treatment ordered by a retailer etc.

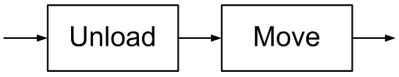
In order to handle cars, a service-based application is introduced. It includes a number of services implementing different car-related activities [1], and a service-based business process that orchestrates the services and implements the workflow. Some of the services do not appear in the business process but are used for the adaptation purposes. The context, in which the car handling process operates, exposes some important features: it is open for changes and situations that are not controlled by the process (e.g., car damage or storage overbooking), dynamic (a set of available services are changing) and non-deterministic (service operations may have various outcomes).

In this paper, we use two versions of the logistics case study: the *simple* version will be exploited to exemplify formal definitions and to illustrate the motivating problems, while the *complex*, extended, version will provide more comprehensive evaluation of the approach and demonstrate its feasibility in real-world settings.

[1]We remark that in this domain certain services are not fully-fledged electronic services (e.g., a service for unloading a car), but rather the wrappers for human-based activities.

A. Simple scenario

Upon its arrival to the port, a car has to be 1) unloaded from the ship to the terminal and 2) moved to the storage area. Services **Unload** and **Move** implement these two activities. The reference process looks like this:



During the process execution, an extraordinary situation may happen. In particular, the car may be damaged (while moving to the storage area, being unloaded, or even before). To deal with that, the application provides two additional services: **Pull** service is used to pull the car to the repair area and **SlowRepair** service is used to repair the car. Finally the application defines the following policies: 1) a car can be driven only if it is completely operable; 2) pulling is applicable only to broken cars and is not applicable on the ship; 3) slow repair requires that the car is brought to the dedicated repair area. This example allows us to demonstrate two important problems.

Context-aware adaptation. If the car gets damaged, a corresponding adaptation depends on the current process context (here, the current location of the car). If the car is on the ship, it still has to be unloaded, and only then pulled to the repair area and finally repaired, while if the car is damaged again immediately after the repair, the **Repair** Service can be applied to it immediately (since the car is already at the repair area). Even this very simple example shows that the same failure may require significantly different adaptation activities depending on the current context. Analyzing and defining these adaptations "manually" using exception handling or through the rules becomes time-demanding and error-prone in real-world scenarios. Indeed, the designer should have a global vision of the application and its context to define appropriate actions, which becomes complicated when the number of relevant context features and their interleaving increases.

Scalability in dynamic environment. Assume that the operation of the repair area is temporarily suspended and mobile repair teams are organized instead. They repair broken cars on the spot, without pulling them to the repair area. Consequently, a new service called **FastRepair** is introduced, while the **SlowRepair** is temporarily removed from the list of available services. As one can see, the **FastRepair** is not just another implementation of car repair procedure, but also has different usage policies (e.g., pulling is not needed anymore).

Such a service replacement has a dramatic impact on the adaptability of the application. Indeed, the adaptation logic should take the new service and the associated policies into

account. While the service replacement can be done at run-time (i.e., service is created and added to the repository without necessarily redeploying the whole application), the changes in the adaptation specification – for the built-in and rule-based approaches – cannot. In built-in approaches predefined adaptation activities affected by the change must be manually redesigned, in rule-based approaches some rules have to be manually re-specified and the whole rule system has to be revalidated. This restricts the applicability of such adaptations in highly dynamic environments.

B. Complex scenario

The above problems become much more prominent in the complex version of our logistics scenario [2]. In this version the business process contains 5 basic milestones (related to the car registration, complex post-transportation treatment and delivery) built on top of 25 services with complex stateful behavior. Furthermore, the range of context features that the scenario takes into account and that are relevant for the adaptation is much wider and includes, for example, the different grades of car damage (important not only for car moving but also for the treatment, diagnosis, and delivery), availability of facilities, car characteristics, etc.

Consequently, the size of the adaptation specification significantly increases: much more new rules should be added; the built-in adaptation activities become more sophisticated.

To overcome these problems the adaptation framework should enable dynamic adaptation, where the need for the adaptation and the solution is automatically derived from the current context and application state, without explicit encoding of all possible situations and solutions for all possible services. To accomplish this, we need:

- a model to characterize the relevant context properties of the application and its adaptation logic. Such a model should decouple the application context, the adaptation requirements, and the constituent services so that their changes (e.g., changed business policies or new services dynamically introduced) would not affect the whole model, thus enabling reuse in dynamic settings.
- a technique to construct the adaptation activities on-the-fly, using the current context, available services, and the application policies and goals, rather than using predefined solutions.
- a context-aware execution and adaptation environment, where the processes are continuously observed and adapted according to the specified model when required by the context changes.

More details about the motivating examples are provided through the paper.

III. Overall approach

In this section we present a goal-based adaptation approach that addresses the problems outlined in Section I. In this approach, the execution of the service-based business process and the evolution of its context are continuously verified against the desired model of the application. This model encodes the business policies over the elements of the domain. Whenever a deviation is detected (due to occurrence of some exogenous event or situation), our framework triggers adaptation, by automatically composing and executing available services taking into account the current state of the process and its context. Specifically, the adaptation aims to bring the process to the expected context configuration, such that no business policies are violated and the process can safely resume its execution.

As such, the approach relies on the following key elements:

- *Modular context-aware representation of the application*;
- *Context-aware execution framework*;
- *Run-time adaptation based on automated service composition*;

1) Context-aware application model: In our model we represent the relevant aspects of the context, the application, and the constituent services.

The important characteristics of the business context are represented with *context properties*. The evolution of these properties may be caused by the application (i.e., as a result of service execution) or by exogenous events in the environment. For example, the property "car status" defines whether the car is damaged or not; the "Repair" service makes it evolve from "damaged" state to "ok", while the car can get damaged (changing state from "ok" to "damaged") as a result of an exogenous event. From this perspective, the state of the application context is captured by the states of all the context properties.

On top of the context properties and services it is possible to define application-specific policies. Such policies define the "desired" context configurations, in which the application should be executed. The policies relate different context properties and constrain the applicability of the process activities. For example, the policy "car can be driven only if operable" requires that the "Move" activity is possible only if the value of the context property "car status" is "ok". The goal of execution framework is, therefore, to guarantee that such policies are not violated by the process execution.

Finally, in order to construct an appropriate adaptation as a composition of services that would achieve the desired context configuration, we annotate the service operations with context *effects*. The effects encode impact of the service operation on the context, i.e., the changes in the context property diagrams. As such, effects relate execution of services to context evolution by indicating at which step of service execution the context is affected. This information is then used during the construction of the adaptation; the resulting composition is organized in a way that the effects corresponding to the execution of the constituent services result in the desired context configuration.

Note the flexibility and the reusability of the proposed

[2] http://soa.fbk.eu/Logistics.zip

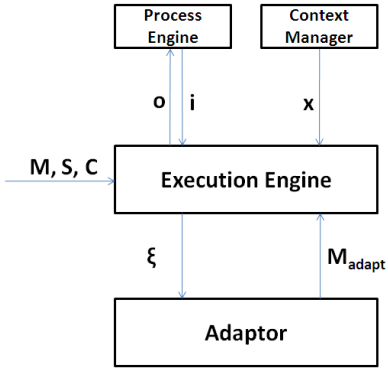


Figure 1. Context-aware Adaptation Architecture.

model: the changes in the application policies do not require modification of the domain model and of the business process; different services may be related (via annotations) to the same context properties; introducing a new service or modifying the existing one affects only the annotation part without any implications on the application and the adaptation logics.

2) Execution framework: The reference architecture of our execution framework is depicted in Figure 1. The execution and adaptation of reference process *M* is managed by the *Execution Engine*. The activities of the process are executed by the *Process Engine* that exchanges messages with the component services S. While executing a process, the Execution Engine sends the activity i to be executed, receives the execution outcome o (if any), and accordingly updates its internal context model C consisting of context property diagrams. On the other side the *Context Manager* continuously monitors[3] the environment in order to detect exogenous context changes.

Before executing the next activity the *Execution Engine* checks if the process needs to be adapted: if the observed context configuration violates the policies associated to the current state of the process, the execution is suspended and the adaptation is initiated. Based on the current configuration of the context and the policies, the *Execution Engine* derives the *adaptation problem* ξ and sends a corresponding adaptation request to the *Adaptor*. In response, the Adaptor generates adaptation process M_{adapt} that changes context so that the policies are satisfied and the process is "unblocked". The Execution Engine executes process M_{adapt} and then comes back to the execution of the main process.

As one can see, the adaptation is completely *run-time* and *automated*; it takes into account the current context and the available services dynamically, without hard-coding them in the adaptation logic. Another important aspect is that the approach *constantly observes the context and can immediately react to the critical changes*. In particular, even if during the execution of the adaptation process some other problem is detected, the process is immediately terminated and a new adaptation is requested.

3) Adaptation: An adaptation problem sent to the Adaptor comprises the current status of the system (values of the context properties), a set of available services to be used for the adaptation, and adaptation goals (i.e., the policies to be satisfied). Given this problem, the Adaptor generates a service composition M_{adapt} using automated planning techniques [6]: the service specifications, the model of the context (i.e., context property diagrams), and the goal specifications are transformed into a planning problem and the resulting plan is then transformed back into a composite service.

We remark that in certain cases the *Adaptor* may fail to find an appropriate composition. Indeed, in the presence of many exogenous context changes and non-deterministic services it is not possible to guarantee that the desired configurations can be reached. To overcome this problem, we adopt the following approach. First, we consider some exogenous context changes and the service outcomes (usually negative, e.g., service errors) as "improbable"[4]. These changes will not be considered by the planner, which increases the chances to find the solution and boosts the performance of the planning algorithm. Second, if during the execution of the adaptation process one of these improbable situations occurs, the execution terminates and another round of adaptation is initiated.

IV. Formal Framework and Implementation

In this section we present formal definition of the elements of our adaptation framework, namely 1) process context, 2) process and component services, 3) policies, and 4) adaptation problem. We also provide examples related to the motivating case study exploited in the paper. After that, we formally describe the way our adaptation approach operates.

A. Formal Framework

1) Context Property: Every context property is modelled with a *context property diagram*, which is a state transition system capturing all possible property values and value changes. Each transition is labeled with the corresponding *context event*. As we mentioned in Section III, a context property may evolve as an effect of service invocations, which corresponds to the "normal" behavior of the domain, but also as a result of volatile – "exogenous" – changes. In this regard, we distinguish *controllable* (i.e., triggered through services) and *uncontrollable* (i.e., exogenous) events. Some of the exogenous events may be marked as "improbable".

Definition (Context Property Diagram): *Context property diagram c is a tuple $\langle L, L_0, E, T \rangle$, where:*

[3]We remark that the specific context monitoring technique is out of the scope of the paper; approaches like [4] may be exploited for this purpose;

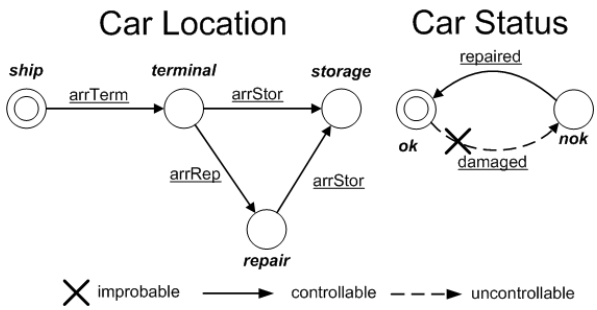
[4]Improbable changes or outcomes can be determined by dedicated data mining techniques like the ones used for process analysis [18]

- *L is a set of configurations and $L_0 \subseteq L$ is a set of initial configurations;*
- *$E = E_U \cup E_C$ is a set of context events, where E_U is a set of uncontrollable and E_C is a set of controllable events, such that $E_U \cap E_C = \emptyset$;*
- *$E'_U \subset E_U$ is a set of improbable uncontrollable events;*
- *$T \subseteq L \times E \times L$ is a transition relation;*

We denote with $L(c)$ and $E(c)$ the corresponding elements of context property diagram c.

Consequently, the overall context is a set of context property diagrams C. The state of the context is a product of states of its property diagrams.

Example. Context property diagrams for the simplified car logistics case study may be specified as follows:



The Car Location diagram captures how the car location can change over time. Initially, the car is on the *ship*. The reference process aims to unload the car to the *terminal* and move it to the *storage*. The *repair* location is where the car can be repaired. Similarly, the Car Status diagram represents car operability status. An example of an improbable and uncontrollable change could be where the car status changes from *ok* to *nok* by the exogenous event *damaged*.

2) Services and Processes: In order to model service-based processes and services with complex protocols (e.g., specified in BPEL) we use state transition systems, where transitions correspond to service actions (i.e., sending/receiving messages to/from services or performing internal assignments and decisions).

Each transition can be additionally annotated with context *effects*. An effect of a service action is a set of context events that fire when the action is executed.

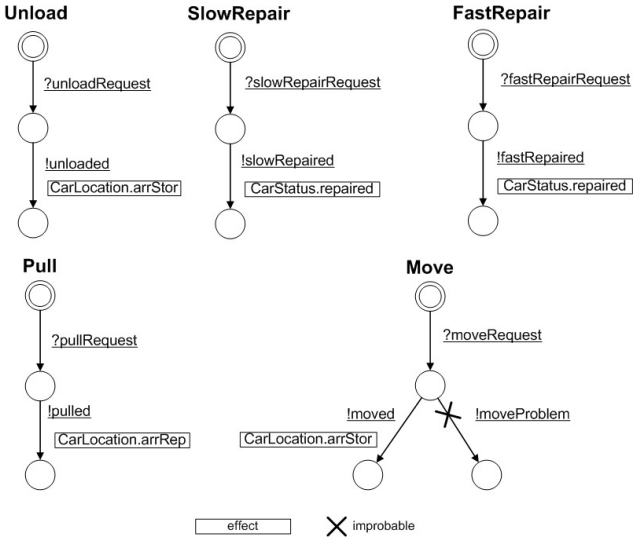
Finally, it is possible to denote some of the non-deterministic outcomes as *improbable*.

Definition (Annotated Service): *Annotated service s defined over context property diagrams C is a tuple $\langle L, L_0, I, O, T \rangle$, where:*

- *L is a set of states and $L_0 \subseteq L$ is a set of initial states;*
- *I is a set of input actions (receiving a message);*
- *O is a set of output actions (sending a message) and $O' \subset O$ are improbable outcomes in non-deterministic operations;*
- *$T \subseteq L \times A \times E^* \times L$ is a transition relation, where $A = I \cup O \cup \{\tau\}$ is a set of actions; $E = \bigcup E_C(c_i), c_i \in C$, thus E^* stands for an effect;*

We denote with $T(s)$ a set of transitions of s.

Example. Annotated service protocols for the simplified car logistics case study are the following:



The service actions are annotated with effects over context properties presented before. For example, one of the (non-deterministic) outcomes of the **Move** service is annotated with the effects $CarLocation.arrStor$ representing the successful movement. The other outcome is labeled as improbable. We remark that in our framework non-deterministic services with multiple "probable" outcomes are also allowed.

3) Policies: We allow for two types of policies. *Service-based policies* constrain the execution of a service action to certain context configurations. They allow for flexible service-specific policies. *Event-based policies* constrain the occurrence of a context event to certain context configurations. They allow for more general domain-specific policies.

Definition (Policy): *Event- or service-based policy r defined over services S and context C are tuples $\langle e, P \rangle$ or $\langle t, P \rangle$ respectively, where $e \in \bigcup_{c \in C} E(c)$ is a context event, $t \in \bigcup_{s \in S} T(s)$ is a service transition and $P \subseteq \prod_{c \in C} L(c)$ is a set of allowed context configurations*

Example. To declare that the the car can move to storage only when it is operable, we define an event-based policy $\langle CarLocation.arrStor, CarStatus \in \{ok\} \rangle$ (i.e., event $CarLocation.arrStor$ corresponding to the arrival at the storage area is possible only in case the value of the CarStatus context property is "ok"). Analogously, for two services with similar functionality (**SlowRepair** and **FastRepair**) the usage policies may differ (**SlowRepair** can only be used at location *Repair*, while there are no restrictions on the other): thus we use service-specific policy $\langle ?slowRepairRequest, CarLocation \in \{repair\} \rangle$.

4) Adaptation Problem: When the next service action in a process cannot be executed due to violation of some policy (e.g., the next action *!moveRequest* cannot be executed because the car is damaged, which violates the policy), the

adaptation is required. The adaptation problem includes the context model (i.e., a set of context property diagrams), annotated specifications of all available services, the current context configuration, and a set of goal configurations of the context. Adaptation problem can be formalized as follows:

Definition (Adaptation Problem):

Adaptation problem ξ is a tuple $\langle C, S, \mathcal{I}, \mathcal{G} \rangle$, where:

- *C is a set of context property diagrams;*
- *S is a set of services defined over C;*
- *$\mathcal{I} \in L(c_1) \times \ldots \times L(c_n), , c_j \in C$ is the current configuration of context property diagrams C and annotated services S;*
- *$\mathcal{G} \subseteq L(c_1) \times \ldots \times L(c_n), c_j \in C$ is a set of goal configurations;*

We denote with $C(\xi)$, $S(\xi)$, $\mathcal{I}(\xi)$, $\mathcal{G}(\xi)$ the corresponding elements of the horizontal adaptation problem ξ.

The solution to the adaptation problem ξ is a process M_{adapt} that orchestrates the services $S(\xi)$. When executed from the current configuration $\mathcal{I}(\xi)$, in the absence of improbable context changes and service outcomes M_{adapt} guarantees that context property diagrams C finish in one of the goal configurations $\mathcal{G}(\xi)$.

B. Adaptation Strategy and Derivation of an Adaptation Process

In order to automatically solve adaptation problems, we adopt and adjust the service composition approach presented in [2]. According to it, a composition problem is transformed into a planning problem and planning techniques are used to resolve it. Similarly, we transform an adaptation problem into a planning problem. A planning domain is derived from adaptive problem ξ. In particular, a set of n services $(s_1, \ldots, s_n \in S(\xi))$, m context property diagrams $(c_1, \ldots, c_m \in C(\xi))$ and business policies are transformed into state transition systems (STS) using transformation rules similar to those presented in [2]. While encoding services as STSs, we also prohibit all "improbable" events and outcomes.

In this way, we obtain a set of STSs $\Sigma_{s_1} \ldots \Sigma_{s_n}$ and $\Sigma_{c_1} \ldots \Sigma_{c_m}$. The planning domain Σ is a product of all STSs of the annotated services and context property diagrams synchronized on effects and policies:

$$\Sigma = \Sigma_{s_1} \| \ldots \| \Sigma_{s_n} \parallel \Sigma_{c_1} \| \ldots \| \Sigma_{c_m}$$

Initial state r of the planning domain is derived from the current configuration $\mathcal{I}(\xi)$ of the context $I(\Sigma) = r$.

Finally, set of goal configurations $\mathcal{G}(\xi)$ is transformed into set of configurations of a planning domain G^{Σ}. We denote the planning goal as a reachability goal $\rho = G^{\Sigma}$.

Finally, we apply the approach of [6] to domain Σ and planning goal ρ and generate a controller Σ_c (plan), such that $\Sigma_c \triangleright \Sigma \models \rho$ (domain Σ reaches goal ρ when controlled by Σ_c). State transition system Σ_c is further translated into executable process M_{adapt}, which implements the adaptation strategy chosen. The back translation from an STS into an executable specification is simple: input actions in Σ_c model an incoming message from a component service, while output actions in Σ_c model an outcoming message to a component service.

Correctness of the approach. The proof of the correctness of the approach consists in showing that, under the aforementioned assumptions, all the executions of the adaptation process M_{adapt} (translation of controller Σ_c) implement the adaptation strategy. Here we outline the key points of the proof. It is easy to see that each execution θ of the adaptation process is also a run of the domain, i. e., if $\theta \in \Pi(\Sigma_c)$ then $\theta \in \Pi(\Sigma)$. Under the planning requirement that all the executions of the domain terminate in goal states, we get that all executions of the domain implement the adaptation strategy. As a consequence, the following theorem holds.

Theorem (Correctness of the approach): *Let:*

- *$\Sigma_{s_1}, \ldots, \Sigma_{s_n}$ be the STS encoding of services $s_1, \ldots s_n$ and*
- *$\Sigma_{c_1}, \ldots, \Sigma_{c_m}$ be the STS encoding of context property diagrams $c_1, \ldots c_m$.*

Let Σ_c be the controller for a particular composition problem

$$\Sigma = \Sigma_{s_1} \| \ldots \| \Sigma_{s_n} \parallel \Sigma_{c_1} \| \ldots \| \Sigma_{c_m}$$

$$I(\Sigma) = r, \rho = G^{\Sigma}$$

i.e., $\Sigma_c \triangleright \Sigma \models \rho$. Then execution $\Pi(\Sigma_c)$ implements the adaptation strategy.

From the planning perspective, the assumptions that no improbable context changes and service outcomes happen during the execution of M_{adapt} has two important consequences for our approach. First, it increases the probability the plan exists (and, therefore, is found by the algorithm). Second, it significantly reduces the amount of reachable states in the planning domain and consequently increase the algorithm performance.

For the same adaptation problem, more than one solution may exist, especially if several services have similar or identical functionality. In this case, the planner returns the plan that is shortest with respect to the number of execution steps.

V. Evaluation

In order to evaluate the effectiveness of the proposed approach, we have implemented the architecture that realizes the adaptation framework described in Section III. The goal of the evaluation was to study the complexity and overhead of the adaptation modelling, efficiency of the adaptation at run-time, and the scalability of the approach in a dynamic environment.

1) Experimental setup: Using our prototype implementation, we have conducted a series of experiments with the complex version of the logistics scenario. We have modelled and tested two variants of reference scenario: one to evaluate modelling overhead and the performance, and another to evaluate the scalability of the approach. The performance

of the planning-based run-time adaptation has been tested on a 2GHz, 3Gb Dual Core machine running Windows.

Variant 1. The reference business process is composed of 5 basic milestones (Ship Unloading, Storage, Technical Treatment, Consignment and Delivery) realized as an orchestration of 20 services. Besides that, 5 other services are also available for the adaptation activities. The adaptation logic of the application is based on 16 context aspects (e.g., car location, car status, storage ticket, etc.) and is governed by a set of 12 business policies over the process and involved services. In this scenario, 24 situations that require adaptation have been identified (e.g.,light and severe car damages, occupied treatment stations, etc.).

Variant 2. This version extends Variant 1 with a new service for in-place repairing of the cars (**FastRepair**). On the contrary, another repair service (**SlowRepair**) is dynamically removed from the repository. Note that in the scenario the process and the adaptation cases remain the same, while the way the process adapts to the same problem will change.

2) Modeling Effort and Complexity: Following our approach, adaptation modelling involved explicit representation of 16 relevant context properties (including their values, changes, and events), annotation of 25 services with corresponding context effects, and encoding of 12 business policies. Note, however, that the explicit modelling of 24 adaptation cases is not required by our framework.

While, in general, such modelling effort may seem significant, in practice it is comparable to the effort needed to encode the adaptation logic using alternative methodologies like, for example, rule-based approaches. To verify this, we have analyzed the implementation of a very similar scenario following the approach presented in [9]. The adaptation was encoded as ECA (event-condition-action) rules defined using AGG tool[5]. While the approach was different and the context properties and service annotations were not modelled explicitly, the effort necessary for the encoding of the rules and policies was comparable.

It is important to notice that our modelling approach enables modularization and re-use of different specification elements. Indeed, the work on annotating services can be entrusted to service providers. If necessary, each service annotation can be reviewed and adjusted by its provider separately from other services of the application. On the contrary, in rule-based approaches any change in services and/or application policies may imply revision of the whole set of rules. Similarly, some elements of our model can be reused by other applications in a given domain: the context model and service annotations, once defined, may be adopted by different business processes.

Finally, the solution provided by our approach is correct with respect to the specified policies by construction. In the approaches, where the adaptation logic is encoded manually, one has to perform non-trivial analysis activities that may require additional efforts. In the rule-based implementation[6] we spent more then 2 hours to tune and evaluate the overall model.

3) Performance of Runtime Adaptation: To evaluate the performance and the outcomes of our adaptation at runtime, we experimented with different adaptation cases of the above scenario. The results of the experiments unveil that, even though the vast majority of service protocols are fairly simple and have 2 to 4 BPEL-like activities, the constructed adaptation processes typically have more than 8 actions (in several cases, due to non-determinism, up to 60 actions) and quite complex structure. Nontheless, the construction of the adaptation process never exceeded 4 seconds, which was much less than the normal execution time of the main process. This shows the high applicability of our algorithm to complex scenarios in different domains.

4) Scalability: To obtain an idea on the scalability of our modelling approach, we considered a second variant of the scenario, where the **FastRepair** service is introduced to replace the **SlowRepair** service. In our approach, this extension required only proper annotation of a new service. Moreover, such annotation can be completed by the service provider independently from the application, where the service is exploited. In the same situation, the rule-based implementation required modifying the set of adaptation rules related to the car damage and reverifying the whole rule system to avoid inconsistencies, which is much more complex and time consuming procedure.

VI. Related Work and Conclusions

In this section we focus on the research on dynamic process adaptation, where the adaptation activities are automatically derived at run time taking into account the actual state of the execution environment.

In most cases, dynamic adaptation approaches are reduced to monitoring services utilized by a process (availability or QoS properties) and replacing them if violation is detected (e.g., [20], [17]). In this case, the "dynamism" mainly consists in 1) discovering a replacing service at run time and 2) preforming service replacement in a running process instance. Although in this paper we presented a context-based adaptation, which is very different from service rebinding, we presume that our framework can be easily extended to perform complex service replacements (e.g., automatic replacement of a failed services composition with another one that performs the same task). The ability to deal with QoS-indicators is in our future work list.

An adaptation approach similar to ours is presented in [8]. SmartPM is a formal framework based on situation calculus and IndiGolog language that provides run-time adaptation of a process to unplanned exceptions. The problem of adaptation is eventually reduced to a classical planning problem. However, SmartPM is currently a theoretical framework and the efficiency and overhead of adaptation modelling

[5]AGG (Attributed Graph Grammars): http://tfs.cs.tu-berlin.de/agg.

[6]http://soa.fbk.eu/Logistics-AGG.zip/

cannot be evaluated so far. Moreover, classical planning cannot deal with non-deterministic services, which are a natural property of modern SOA systems. As opposed to SmartPM, our approach is able to deal with stateful and non-deterministic services. Our formal framework intuitively represents the relevant business concepts (context, policies, processes) and, in combination with tools translating BPEL to state transition systems [7], allows for efficient modelling tools that minimize the adaptation modelling effort.

The main extension that we plan is the ability to use abstract activities (e.g., "to unload a car") in the business process. In this case, the refinement of an abstract activity with executable service composition is done automatically and at run time taking into account the current status of the execution environment. Our preliminary study suggests that our approach can be easily modified to enable abstract activity refinement.

Currently, the adaptation strategy chosen simply tries to "unblock" the next action in the process, which might be not enough in some scenarios. We plan to consider other strategies, e.g., to roll the execution back to a branching point in an attempt to follow another branch.

Our adaptation mechanism is currently applied to instances of a business process. It is possible to use the execution history of adapted instances as a training set to progressively improve the process model (*process evolution*).

We also consider planning adaptation activities using optimality criteria different from simple execution steps count (e.g., minimal side effects, minimal execution cost of the process and other QoS indicators).

ACKNOWLEDGMENT

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

REFERENCES

[1] V. Agarwal and P. Jalote. From Specification to Adaptation: An Integrated QoS-driven Approach for Dynamic Adaptation of Web Service Compositions. In *Proc. ICWS*, pages 275–282, 2010.

[2] P. Bertoli, R. Kazhamiakin, M. Paolucci, M. Pistore, H. Raik, and M. Wagner. Control Flow Requirements for Automated Service Composition. In *Proc. ICWS'09*, pages 17–24, 2009.

[3] F. Böse and J. Piotrowski. Autonomously controlled storage management in vehicle logistics applications of RFID and mobile computing systems. *International Journal of RT Technologies: Research an Application*, 1(1):57–76, 2009.

[4] D. Maggiorini C. Bettini and D. Riboni. Distributed Context Monitoring for the Adaptation of Continuous Services. *World Wide Web*, 10:503–528, 2007.

[7] http://astroproject.org/

[5] M. Colombo, E. di Nitto, and M. Mauri. SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In *Proc. ICSOC'06*, pages 191–202, 2006.

[6] M. Pistore D. Shaparau and P. Traverso. Contingent planning with goal preference. In *Proc. AAAI'06*, pages 927–934, 2006.

[7] R. de Lemos and A. B. Romanovsky. Exception handling in the software lifecycle. *Comput. Syst. Sci. Eng.*, 16(2):119–133, 2001.

[8] M. de Leoni. Adaptive Process Management in Highly Dynamic and Pervasive Scenarios. In *Proc. YR-SOC*, pages 83–97, 2009.

[9] H. Ehrig, C. Ermel, O. Runge, A. Bucchiarone, and P. Pelliccione. Formal analysis and verification of self-healing systems. In *FASE*, pages 139–153, 2010.

[10] A. Hallerbach, T. Bauer, and M. Reichert. Capturing variability in business process models: the Provop approach. *Journal of Software Maintenance*, 22(6-7):519–546, 2010.

[11] G. Hermosillo, L. Seinturier, and L. Duchien. Using Complex Event Processing for Dynamic Business Process Adaptation. In *Proc. IEEE SCC*, pages 466–473, 2010.

[12] Z. Jaroucheh, X. Liu, and S. Smith. Apto: A MDD-Based Generic Framework for Context-Aware Deeply Adaptive Service-Based Processes. In *Proc. ICWS*, pages 219–226, 2010.

[13] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. P. Buchmann. Extending BPEL for Run Time Adaptability. In *Proc. EDOC'05*, pages 15–26, 2005.

[14] W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati. An Aspect-Oriented Framework for Service Adaptation. In *Proc. ICSOC'06*, pages 15–26, 2006.

[15] I. Lanese, A. Bucchiarone, and F. Montesi. A Framework for Rule-based Dynamic Adaptation. In *Proc. TGC 2010*, pages 284–300, 2010.

[16] A. Marconi, M. Pistore, A. Sirbu, H. Eberle, F. Leymann, and T. Unger. Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation. In *Proc. ICSOC/ServiceWave*, pages 445–454, 2009.

[17] George Spanoudakis, Andrea Zisman, and Alexander Kozlenkov. A service discovery framework for service centric systems. In *IEEE SCC*, pages 251–259, 2005.

[18] Wil M. P. van der Aalst. Process discovery: Capturing the invisible. *IEEE Comp. Int. Mag.*, 5(1):28–41, 2010.

[19] X. YongLin and W. Jun. Context-Driven Business Process Adaptation for Ad Hoc Changes. In *Proc. IEEE ICEBE'08*, pages 53–60, 2008.

[20] Yanlong Zhai, Jing Zhang, and Kwei-Jay Lin. Soa middleware support for service process reconfiguration with end-to-end qos constraints. In *ICWS*, pages 815–822, 2009.

Appendix D

Multi-layered Monitoring and Adaptation

Multi-layered Monitoring and Adaptation*

Sam Guinea[1], Gabor Kecskemeti[2], Annapaola Marconi[3], and Branimir Wetzstein[4]

[1] Politecnico di Milano
Deep-SE Group - Dipartimento di Elettronica e Informazione
Piazza L. da Vinci, 32 - 20133 Milano, Italy
guinea@elet.polimi.it
[2] MTA-SZTAKI
Laboratory of Parallel and Distributed Systems
Kende u. 13-17, 1111 Budapest, Hungary
kecskemeti@sztaki.hu
[3] Fondazione Bruno Kessler
via Sommarive 18, 38123 Trento, Italy
marconi@fbk.eu
[4] University of Stuttgart
Institute of Architecture of Application Systems
Universitaetsstr. 38, 70569 Stuttgart, Germany
wetzstein@iaas.uni-stuttgart.de

Abstract. Service-based applications have become more and more multi-layered in nature, as we tend to build software as a service on top of infrastructure as a service. Most existing SOA monitoring and adaptation techniques address layer-specific issues. These techniques, if used in isolation, cannot deal with real-world domains, where changes in one layer often affect other layers, and information from multiple layers is essential in truly understanding problems and in developing comprehensive solutions.
In this paper we propose a framework that integrates layer specific monitoring and adaptation techniques, and enables multi-layered control loops in service-based systems. The proposed approach is evaluated on a medical imaging procedure for Computed Tomography (CT) Scans, an e-Health scenario characterized by strong dependencies between the software layer and infrastructural resources.

1 Introduction

Service-based systems are built under an open-world assumption. Their functionality and quality of service depend on the services they interact with, yet these services can evolve in many ways, for better or for worse. To be sure these evolutions do not lead to systems that behave inadequately or fail, service-based

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (Network of Excellence S-Cube – http://www.s-cube-network.eu/)

systems must be able to re-arrange themselves to cope with change. A typical way of dealing with these issues is to introduce some variant of the well-known monitor-analyze-plan-execute (MAPE) loop into the system [6], effectively making the system self-adaptive.

The service abstraction has become so pervasive that we are now building systems that are multi-layered in nature. Cloud-computing allows us to build software as a service on top of a dynamic infrastructure that is also provided as a service (IaaS). This complicates the development of self-adaptive systems because the layers are intrinsically dependent one of the other. Most existing SOA monitoring and adaptation techniques address one specific functional layer at a time. This makes them inadequate in real-world domains, where changes in one layer will often affect others. If we do not consider the system as a whole we can run into different kinds of misjudgments. For example, if we witness an unexpected behavior at the software layer we may be inclined to adapt at that same layer, even though a more cost-effective solution might be found either at the infrastructure layer, or by combining adaptations at both layers. Even worse, a purely software adaptation might turn out to be useless due to infrastructural constraints we fail to consider. Similar considerations are made in case of the unexpected behavior at the infrastructure layer, or at both.

In this paper we propose a framework that integrates software and infrastructure specific monitoring and adaptation techniques, enabling multi-layered control loops in service-based systems. All the steps in the control loop acknowledge the multi-faceted nature of the system, ensuring that we always reason holistically, and adapt the system in a coordinated fashion. In our prototype we have focused on the monitoring and adaptation of BPEL processes that are deployed onto a dynamic infrastructure.

Building upon our past experiences we have integrated process and infrastructure level monitoring [2,8] with a correlation technique that makes use of complex event processing [1]. The correlated data, combined with machine-learning techniques, allow us to pinpoint where the problems lie in the multi-layered system, and where it would be more convenient to adapt [7,12]. We then build a complex adaptation strategy that may involve the software and/or the infrastructure layer [13], and enact it through appropriate effectors.

The proposed approach is evaluated on a medical imaging procedure for Computed Tomography (CT) Scans, an e-Health scenario characterized by strong dependencies between the software layer and infrastructural resources.

The rest of this paper is organized as follows. Section 2 gives a high-level overview of the integrated monitoring and adaptation framework used to enable the multi-layered control loops. Section 3 details the software and infrastructure monitoring tools and how they are correlated using complex event processing. Section 4 explains how decision trees are used to identify which parts in the system are responsible for the anomalous behaviors and what adaptations are needed. Section 5 explains how we coordinate single-layer adaptation capabilities to define a multi-layered adaptation strategy, while Section 6 presents the tools used to actually enact the adaptations. Section 7 evaluates the integrated

approach on a medical imaging procedure. Section 8 presents related work, and Section 9 concludes the paper.

2 The Integrated Monitoring and Adaptation Framework

We propose an integrated framework that allows for the installation of multi-layered control loops in service-based systems. We will start with a conceptual overview, and then provide more details on the single techniques we have integrated in our prototype.

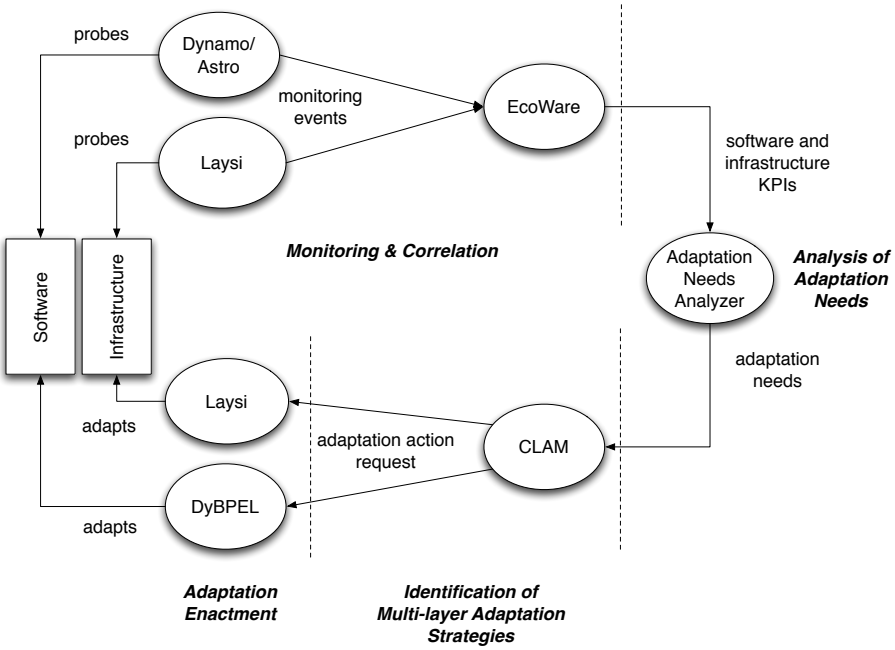


Fig. 1. The Monitoring and Adaptation Framework.

Figure 1 gives a high-level view of our integrated monitoring and adaptation framework, as used in a multi-layered software and infrastructure system. To establish self-adaptation, the framework applies a slight variation of the well-known MAPE control loop. Dashed vertical lines separate the four main steps in the loop, while oval shapes represent the concrete techniques that we have integrated – detailed later in Sections 3–6.

In the *Monitoring and Correlation* step, sensors deployed throughout the system capture run-time data about its software and infrastructural elements. The collected data are then aggregated and manipulated to produce higher-level correlated data under the form of general and domain-specific metrics. The main

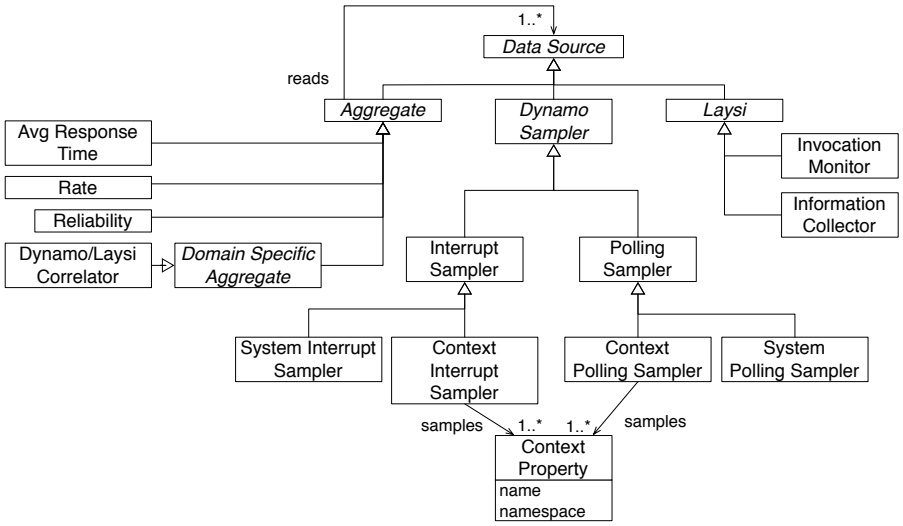


Fig. 2. The Monitoring and Correlation Model.

goal is to reveal correlations between what is being observed at the software and at the infrastructure layer to enable global system reasoning.

In the *Analysis of Adaptation Needs* step, the framework uses the correlated data to identify anomalous situations, and to pinpoint and formalize where it needs to adapt. It may be sufficient to adapt at the software or at the infrastructure layer, or we may have to adapt at both.

In the *Identification of Multi-layer Adaptation Strategies* step, the framework is aware of the adaptation capabilities that exist within the system. It uses this knowledge to define a multi-layer adaptation strategy as a set of software and/or infrastructure adaptation actions to enact. A strategy determines both the order of these actions and the data they need to exchange to accomplish their goals.

In the *Adaptation Enactment* step, different adaptation engines, both at the software and the infrastructure layer, enact their corresponding parts of the multi-layer strategy. Each engine typically contains a number of specific modules targeting different atomic adaptation capabilities.

3 Monitoring and Correlation

Monitoring consists in collecting data from a running application so that they can be analyzed to discover runtime anomalies; event correlation is used to aggregate runtime data coming from different sources to produce information at a higher level of abstraction. In our integrated framework we can obtain low-level data/events from the process or from the context of execution using Dynamo [2], or from the infrastructure using Laysi [8]. We can then manipulate the data to

obtain higher-level information using the event correlation capabilities provided by EcoWare [1]. Figure 2 gives an overview of the kind of data sources available through Dynamo, Laysi, and EcoWare.

Dynamo provides means for gathering events regarding either (*i*) a process' internal state, or (*ii*) context data[5]. `Interrupt Sampler`s interrupt a process at a specific point in its execution to gather the information, while `Polling Sampler`s do not block the process but gather their data through polling.

The `Invocation Monitor` is responsible for producing low-level infrastructure events through the observation of the various IaaS systems managed by Laysi. These events signal a service invocation's failure or success, where failures are due to infrastructure errors. The infrastructure, however, can also be queried through the `Information Collector` to better understand how services are assigned to hosts. The differences between the utilized infrastructures and the represented information are hidden by the information collector component of the MetaBroker service in Laysi.

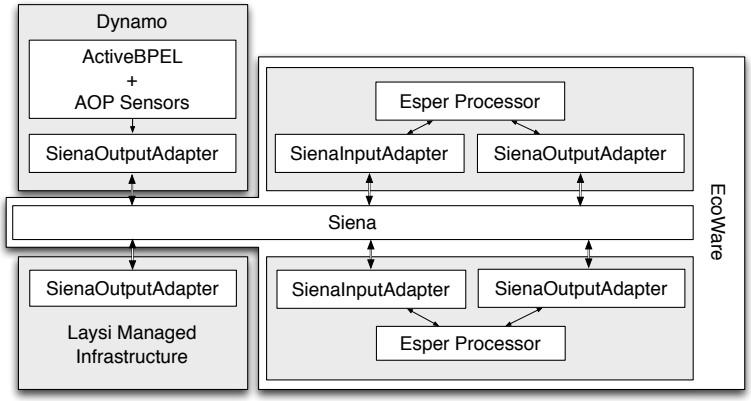


Fig. 3. The Dynamo and EcoWare Architecture.

The events collected through Dynamo and Laysi can be further aggregated or manipulated by EcoWare. We can use a predefined aggregate metric such as `Reliability`, `Average Response Time`, or `Rate`, or we can use a domain-specific aggregate whose semantics is expressed using the Esper event processing language. Aggregates process events coming from one or more data sources and produce new ones that can be even further manipulated in a pipe-and-filter style.

For our integrated approach we developed a domain-specific aggregate called the `Dynamo/Laysi Correlator` to correlate events produced at the software and the infrastructure layers. This component exploits a correlation data set that is

[5] We intend as context any data source, external to the system, that offers a service interface.

artificially introduced by Dynamo in every service call it makes to the Laysi infrastructure. The correlation data contains the name of the process making the call to Laysi, the invocation descriptor in the form of a unique JSDL (Job Submission Description Language) document, and a unique ID for the process instance that is actually making the request. These data are also placed within the events that are generated by the Invocation Monitor, allowing EcoWare to easily understand which software- and infrastructure-level events are related. Figure 3 gives an overview of the technical integration of Dynamo, Laysi, and EcoWare, which is achieved using a Siena publish and subscribe event bus. Input and output adapters are used to align Dynamo, Laysi, and the event processors with a normalized message format.

4 Analysis of Adaptation Needs

Monitoring and correlation produce simple and complex metrics that need to be evaluated. A Key Performance Indicator consists of one of these metrics (e.g., overall process duration) and a target value function which maps values of that metric to two or more categories on a nominal scale (e.g., "process duration < 3 days is *good*, otherwise *bad*" defines two KPI categories). These KPI categories allow us to interpret whether, and how, KPI metric values conform to business goals. If monitoring shows that many process instances have bad KPI performance, we need to (*i*) analyze the influential factors that lead to these bad KPI values, and (*ii*) find adaptation actions that can improve those factors and thus the KPI. Figure 4 shows an overview of the KPI-based Adaptation Needs Analyzer Framework [7,12] and its relation to the overall approach. It consists of two main components: an `Influential Factor Analysis` component and an `Adaptation Needs Analysis` component.

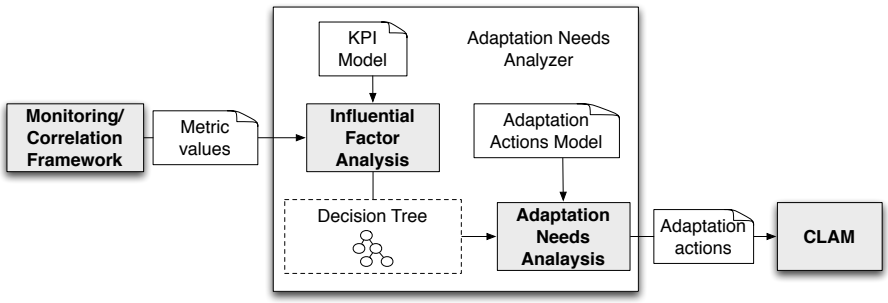


Fig. 4. Adaptation Needs Analysis Framework.

The Influential Factor Analysis receives the metric values for a set of process instances within a certain time period. In this context, interesting metrics are measured both on the process level and the service infrastructure level. At the

process level, metrics include the durations of external service calls, the duration of the overall business process, the process paths taken, the number of iterations in loops, and the process' data values. Service infrastructure metrics describe the service invocation properties which include the status of the service invocation (successful, failed), and properties such as the infrastructure node on which the service execution has been performed.

It uses machine learning techniques (decision trees) to find out the relations between a set of metrics (potential influential factors) and the KPI category based on historical process instances [12]. The algorithm is fed with a data set, whereby each data item in this set represents one process instance and the values of all the metrics that were measured for that instance and the KPI category that has been evaluated. The algorithm creates a decision tree in which nodes represent metrics (e.g., the duration of a particular activity), outgoing edges represent conditions on the values of the metric, and leaves represent KPI categories. By following the paths from the root of the tree to its leaves, we can see for which combinations of metrics and values particular KPI categories have been reached (e.g., if duration of activity A was above 3 hours and activity B was executed on node 2 the KPI value was bad).

Based on this analysis the next step is to use the Adaptation Needs Analysis component to identify the adaptation needs, i.e., what is to be adapted in order to improve the KPI [7]. The inputs to this step are the decision tree and an adaptation actions model which has to be manually created by the user. The model contains different adaptation actions, whereby each specifies an adaptation mechanism (e.g., service substitution, process structure change) and how it affects one or more of the metrics used in the Influential Factor Analysis. For example, an adaptation action could be to substitute service A in the process with service B, and its effect could be "service response time < 2 h". The Adaptation Needs Analysis extracts the paths which lead to bad KPI categories from the tree and combines them with available adaptation actions which can improve the corresponding metrics on the path. As a result, we obtain different sets of potential adaptation actions. However, each of these sets does not yet take cross-layer dependencies between adaptation actions into account. This is performed in the next step by the CLAM framework.

5 Identification of Multi-layer Adaptation Strategies

The main aim of the Cross Layer Adaptation Manager (CLAM) [13] is to manage the impact of adaptation actions across the system's multiple layers. This is achieved in two ways: on the one hand CLAM identifies the application components that are affected by the adaptation actions, and on the other hand, it identifies an adaptation strategy that properly coordinates the layer-specific adaptation capabilities. CLAM relies on a model of the multi-layer application that contains the current configuration of the application's components (e.g. business processes with KPIs, available services with stated QoS and general information, available infrastructure resources) and their dependencies (e.g. business activity

A is performed by service S). When the CLAM identifies the components that are affected by the adaptation actions, it uses a set of `checkers`, each associated with a specific application concern (e.g. service composition, service performances, infrastructure resources), to analyze whether the updated application model is compatible with the concern's requirements. The goal is to produce a strategy that is modeled as an Adaptation Tree. The tree's root represents the model's initial configuration; its other nodes contain the configurations of the model, as updated by the adaptation actions, and the checkers that need to be invoked at each step; its edges represent the outcome of the invoked checkers.

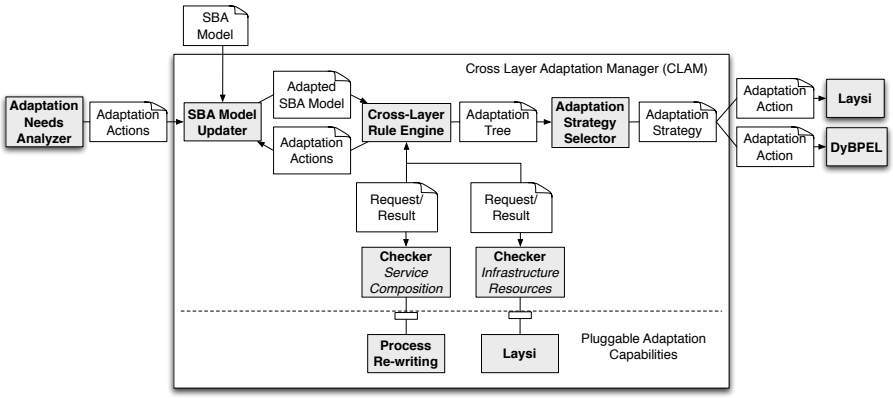


Fig. 5. CLAM: Cross-layer Adaptation Manager

Figure 5 presents an overview of CLAM's architecture. Whenever a new set of adaptation actions is received from the Adaptation Needs Analyzer, the `SBA Model Updater` module updates the current application model by applying the received adaptation actions. CLAM requires that all the adaptation actions be applicable with respect to the current model. However, this is guaranteed in the proposed multi-layer framework by the Adaptation Needs Analyzer.

The adapted model is then used by the `Cross-layer Rule Engine` to detect the components in the layers affected by the adaptation and to identify, through the set of predefined rules, the associated adaptation checkers. If some constraints are violated, the checker is responsible for searching for a local solution to the problem. This analysis may result in a new adaptation action to be triggered. This is determined through the interaction with a set of pluggable application-specific adaptation capabilities.

The Cross-layer Rule Engine uses each checker's outcome to progressively update the strategy tree. If the checker triggers a new adaptation action, the Cross-layer Rule Engine obtains a new adapted model from the Model Updater, and adds it as a new node to the strategy tree, together with the new checkers to be invoked. If the checker reports that the adaptation is not compatible and

that no solution can be found, the node is marked as a red leaf; the path in the tree that leads from the root to that specific node represents an unsuccessful strategy. On the contrary, if all checks complete successfully, the node is a green leaf that can be considered a stable configuration of the application, and the corresponding path in the tree represents an adaptation strategy that can be enacted.

If multiple adaptation strategies are identified, the `Adaptation Strategy Selector` is responsible of choosing the best strategy by evaluating and ranking the different strategies according to a set of predefined metrics. The selected strategy is then enacted passing the adaptation actions to the adaptation enactment tools.

Due to our scenario's requirements we have currently integrated two specific adaptation capabilities into our framework: the `Process Re-Writing` planner, responsible of optimizing service compositions by properly parallelizing sequential activities, and `Laysi`, whose aim is to guarantee a correct and optimized usage of infrastructure resources.

The Process Re-writing Planner is an adaptation mechanism that, given a BPEL process and a set of optimization requirements, automatically computes a new version of the process that maximizes the parallel execution of activities. This is done taking into account a set of data and control flow requirements that characterize the process' correct behavior (e.g. activity A cannot be executed in parallel with B, activity A must follow activity B, etc.), as well as any interaction protocols the partner services may require (e.g. if service S expects activity A to be executed before activity B, than this protocol requirement will be satisfied).

Laysi offers self-management capabilities for service infrastructures and allows new infrastructure level requirements to be evaluated before the actual service invocations take place. Hence, upon receiving the possible parallel execution options from Process Re-writing Planner, the CLAM architecture presents these options as requirements (including the required parallelism and time constraints) to Laysi for all the not-yet executed service calls. In response, Laysi determines the feasibility of the proposed requirements taking into account that a rearrangement of the service infrastructure may be needed. If the system decides to enact the adaptation and the infrastructure needs to be rearranged, Laysi will ensure the next invocation can meet its agreed constraints according to the adaptation enactment tasks specified in the following section.

6 Adaptation Enactment

In our integrated approach we enact software adaptations through `DyBPEL`, and infrastructure adaptations through `Laysi`. CLAM issues specific actions of the chosen adaptation strategy to each tool in a coordinated fashion.

In the proposed integration, DyBPEL is responsible of enacting the process restructuring adaptations identified by the Process Re-writing Planner.

DyBPEL extends an open-source BPEL execution engine (ActiveBPEL) with the capability to modify a process' structure at run time. The change can be

applied to a single process instance or to an entire class of processes. DyBPEL consists of two main components: a `Process Runtime Modifier`, and a `Static BPEL Modifier`. The runtime modifier makes use of AOP techniques to intercept a running process and modify it in one of three ways: by intervening on its BPEL activities, on its set of partnerlinks, or on its internal state. The runtime modifier takes three parameters. The first is an XPath expression that uniquely identifies the point in the process execution in which the restructuring has to be activated. The second is an XPath expression that uniquely identifies the point in the process in which restructuring needs to be achieved (it can be different than the point in which the restructuring is activated). The third is a list of restructuring actions. Supported actions consist of the addition, removal, or modification of BPEL activities, partnerlinks, and data values. When dealing with BPEL activities we must provide the BPEL snippet that needs to be added to the process, or used to modify one of the process' existing activities. When dealing with partnerlinks we must provide the new partnerlink that needs to be added to the process, or used to modify an existing one. When dealing with the process' state we must uniquely identify a BPEL variable within the process to be added or modified, and the XML snippet that will consist of its new value.

When the process restructuring needs to be more extensive, we can use the static BPEL modifier. It supports the same kinds of modifications to the process' activities, partnerlinks, and internal variables, except that the modifications are performed on the process' XML definition. This operation is completely transparent to users. First of all, already running instances are not modified and changes are only applied to new instances. Second, using the same endpoint, all new process requests are forwarded to the newly deployed version of the process.

Regarding infrastructure adaptation, Laysi always performs service requests on a best-effort basis. Each service invocation is handled individually and the various calls are assumed to be independent. Consequently, the performance of the service requests might not be aligned with the higher layers of the service-based system. To provide better alignment with the service composition layer we can specify special constraints about service placement (e.g. service instance A should be hosted within the same provider as service instance B) and availability within the infrastructure (e.g. a service instance should be available before the invocation request is placed in the call queue of Laysi). These constraints are derived directly from the business process and the future interactions between the available service instances hosted by the infrastructure. Laysi constructs the service infrastructure on five layers: meta negotiators, meta brokers, service brokers, automatic service deployers, and the physical infrastructures (grid resources or cloud based virtual machines). These infrastructure layers autonomously adapt themselves to the placed constraints (e.g. placement, availability, CPU, memory, pricing). The autonomous behavior of the infrastructure may involve (*i*) new service instance deployment in high demand situations, (*ii*) service broker replacement in case of broken or low performing physical infrastructures, and/or (*iii*) negotiation bootstrapping if a new negotiation technique is required.

7 The CT Scan Scenario

The application domain considered in this paper concerns the medical imaging procedure for Computed Tomography (CT) Scans. A CT Scan is an X-ray based medical test that, exploiting sophisticated image processing algorithms, produces cross-sectional images of the inside of the body. These images can be further processed to obtain three dimensional views.

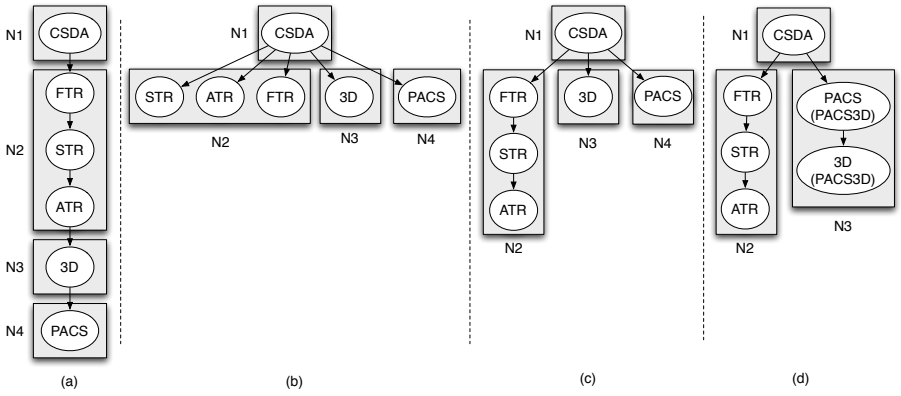


Fig. 6. Evolution of the CT scan scenario.

Figure 6(a) describes the typical CT Scan process. White ovals represent software services, while gray rectangles tell us the infrastructure nodes hosting them. During the Cross Sectional Data Acquisition phase (service `CSDA`) the CT scanner acquires X-ray data for individual body cross sections depending on which parts of the body need to be scanned. These data are then used by complex image processing services (offered by various hosts in the infrastructure) to obtain a set of cross-sectional images from different perspectives as well as 3D volumetric information. The services are the Frontal Tomographic Reconstruction service (`FTR`), the Sagittal Tomographic Reconstruction service (`STR`), the Axial Tomographic Reconstruction service (`ATR`), and the 3D volumetric information service (`3D`). Finally, the data is stored to a picture archiving and communication system using the `PACS` service.

These activities require enormous processing power. To keep costs down, the hospital only maintains the resources needed for emergency CT scans. During burst periods, such as during the public opening hours of the CT laboratory, it relies on an infrastructure dynamically extensible with virtual machines from IaaS cloud infrastructures managed by Laysi.

In the following we show how our approach can be used to automatically adapt this multi-layered system. The CT Scan process is initially designed by a domain expert on the basis of the common medical procedure. The obtained process is a simple sequence of actions that does not embed any optimization with

respect to its performance (Figure 6 (a)). The domain expert also specifies his goals for the quality of the medical procedure using a set of KPIs. For instance, an important goal is to ensure that the processing time of a CT scan does not rise above 60 minutes. An advanced user, such as a hospital IT technician, defines a set of adaptation actions that can be used to improve the process' performance: (*i*) the parallelization of process activities; (*ii*) the substitution of some services (for example, the use of a more costly PACS3D service capable of substituting both services PACS and 3D); (*iii*) the deployment of a service onto a new infrastructural node with specific characteristics.

At run time we collect monitoring events both at the software and the infrastructure level and correlate them using EcoWare. After a certain period of time, we notice that the CT process' performance is degrading: in the last 400 scans about 25% have not achieved their desired overall CT scan processing time. In order to identify the reasons for this behavior, the Influential Factor Analysis is fed the following process and infrastructure level metrics: (*i*) the duration of each process activity; (*ii*) the duration of the whole process with respect to the *the type of the CT scan* (whole body, head, kidney etc.) as it determines the amount of work to be done; (*iii*) the particular infrastructure node a service execution has been executed on; (*iv*) the status of a service execution (successful, faulted); and (*v*) the type of infrastructure the services have been executed on (internal or external – available through Laysi).

The Influential Factor Analysis shows that from the 100 scans which violated the KPI target, 90 scans have been "whole body CT scans" executed on an external infrastructure. It also shows that the infrastructure has caused service execution faults only in 12 cases (out of 400). Finally, all scans performed on the internal infrastructure were successful. Based on this analysis, the Adaptation Needs Analysis selects predefined adaptation actions which can improve the "overall process duration in case of whole body CT scans". It selects process activity parallelization as it is the only adaptation which has been specified to have a direct positive effect on this metric.

This adaptation action is passed to the CLAM which updates the process model so that all activities are executed in parallel. The Cross-Layer Rule Engine detects the change in the process model and understands that these changes have to be checked by the composition checker and by the infrastructure checker (as the parallel execution of services has to be supported by the underlying infrastructure). The composition checker invokes the Process Re-Writing Planner which considers the original data- and control-flows of the process. It notices that the activities cannot all be executed in parallel since five of them depend on CSDA's results; thus the planner returns a new adaptation action which ensures that CSDA is executed first, while all the other activities are conducted in parallel. The model is updated in CLAM as shown in Figure 6 (b) and a new node is added to the adaptation tree. In the next step, the Cross-Layer Rule Engine invokes the infrastructure checker component which, through Laysi, discovers that the activities for tomographic reconstruction (i.e. FTR, STR, and ATR) can only be executed on the node N2. The Rule Engine handles this

new adaptation need by invoking again the Process Re-Writing Planner with a new set of control-flow constraints (i.e. FTR, STR, and ATR must be executed sequentially). The resulting process structure is shown in Figure 6 (c), in which, after CSDA, there are three parallel branches. In one of these branches FTR, STR, and ATR are executed sequentially, while, in the other two, the process executes the 3D and PACS services. The model is updated and the infrastructure checker component is invoked again. This new version of the process passes the infrastructure validation and, since there are no more checkers to be invoked, the corresponding strategy is enacted. In particular, the adapted process is handed over to DyBPEL, which manages the transition to the new process definition.

The adapted process is executed and after a certain period of time we notice that the number of KPI violations has been reduced to 10%, and that most KPI violations happen when the PACS service's execution time is too high. Therefore, two alternative adaptation actions are found: either (*i*) move the PACS service instance to another (better performing) node, or (*ii*) replace PACS with the new service PACS3D. Both alternatives are passed to CLAM.

The CLAM Rule Manager invokes the infrastructure checker with the constraint to the Laysi infrastructure stating that the PACS service should never be executed on node N4. Unfortunately, Laysi responds that, due to constraints, this is not possible and that PACS must always be executed on N4. The CLAM Rule Manager drops the first adaptation action alternative as it is not realizable, and repeats the procedure with the second adaptation action, the substitution of the PACS service with a service called PACS3D, capable of providing both storage and 3D reconstruction at a higher cost. This alternative has to be checked both by the composition checker and by the infrastructure checker. The Process Re-writing Planner detects that a new process restructuring is necessary: a new control-flow requirement is introduced by the protocol of the PACS3D service which requires to receive and store all the X-Ray data information (PACS) before computing the 3D Scan (3D). The SBA Model resulting from this new adaptation action is depicted in Figure 6 (d). The parallel branches are now only two, one for FTR, STR, and ATR, and one for PACS3D which is called twice, once to perform 3D reconstruction, and once to perform storage. The infrastructure checker validates the new model and the corresponding strategy is enacted.

8 Related Work

There are not many approaches in literature that integrate multi-layered monitoring and adaptation of service-based systems. There are however many that focus on layer-specific problems. For example, Moser et al. [10] present VieDAME, a non-intrusive approach to the monitoring of BPEL processes. The approach accumulates runtime data to calculate QoS values such as response time, accuracy, or availability. It also provides a dynamic adaptation and message mediation service for partnerlinks, using XSLT or regular expressions to transform messages accordingly. Colombo et al. [3] extend the BPEL composition language with policy (re)binding rules written in the Drools language. These rules take the form

of if-then-else statements, allowing service bindings to depend on process data collected at run time. The approach also provides mediation capabilities through a special-purpose mediation scripting language.

Researchers that do consider multi-layered applications, on the other hand, tend to concentrate either on monitoring them or on adapting them. We present the most prominent research being done in both these fields. Foster et al. [5] have proposed an extensible framework for monitoring business, software, and infrastructure services. The framework allows different kinds of reasoners, tailored to different kinds of services, to be integrated and to collaborate to monitor decomposable service level agreement terms and expressions. The framework automatically assigns the decomposed atomic terms to specific reasoners, yet the approach does not support the correlation of terms monitored at different layers. Mos et al. [9] propose a multi-layered monitoring approach that considers service and infrastructure level events produced by services deployed to a distributed enterprise service bus. Basic computations can be performed on the events to produce aggregate information (e.g., averages) or complex event processing can be used for more complex correlations and verifications. The resulting data are analyzed by comparing them to thresholds, and the knowledge collected at the various levels are presented through appropriately differentiated user interfaces and visualization techniques. The approach does not correlate knowledge collected at the different levels.

Regarding multi-level adaptation, Efstratiou et al. [4] present an approach for adapting multiple applications that share common resources. These applications are not composed, but rather single entities affected by the same contextual attributes. Since these applications live in the same space they need to coordinate how they manage the shared resources to avoid conflicts. However, they expect the users to perceive and model the conflicts manually. Finally, Popescu et al. [11] propose a framework for multi-layer adaptation of service-based systems comprised of organization, coordination and service layers. In this approach a designer needs to prepare a taxonomy of the adaptation mismatches, and then a set of adaptation templates, known as patterns, that define generic solutions for these mismatches. This differs from our proposed approach since we do not require on design-time knowledge but discover our strategies on-the-fly.

9 Conclusion and Future Work

In this paper we have presented an integrated approach for monitoring and adapting multi-layered service-based systems. The approach is based on a variant of the well-known MAPE control loops that are typical in autonomic systems. All the steps in the control loop acknowledge the multi-faceted nature of the system, ensuring that we always reason holistically, and adapt the system in a cross-layered and coordinated fashion. We have also presented initial validation of the approach on a dynamic CT scan scenario.

In our future work we will continue to evaluate the approach through new application scenarios, and through the addition of new adaptation capabilities

and adaptation enacting techniques. We will also integrate additional kinds of layers, such as a platforms, typically seen in cloud computing setups, and business layers. This will also require the development of new specialized monitors and adaptations. Finally, we will study the feasibility of managing different kinds of KPI constraints.

References

1. L. Baresi, M. Caporuscio, C. Ghezzi, and S. Guinea. Model-Driven Management of Services. In *Proceedings of the Eighth European Conference on Web Services, ECOWS*, pages 147–154. IEEE Computer Society, 2010.
2. L. Baresi and S. Guinea. Self-Supervising BPEL Processes. *IEEE Trans. Software Engineering*, 37(2):247–263, 2011.
3. M. Colombo, E. D. Nitto, and M. Mauri. SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In *Proceedings of the Fourth International Conference on Service Oriented Computing, ICSOC*, pages 191–202, 2006.
4. C. Efstratiou, K. Cheverst, N. Davies, and A. Friday. An architecture for the effective support of adaptive context-aware applications. In *Proceedings of the Second International Conference on Mobile Data Management, MDM*, pages 15–26. Springer, 2001.
5. H. Foster and G. Spanoudakis. SMaRT: a Workbench for Reporting the Monitorability of Services from SLAs. In *Proceedings of the 3rd International Workshop on Principles of Engineering Service-oriented Systems, PESOS*, pages 36–42. ACM, 2011.
6. P. Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. *IBM TJ Watson Labs.*, October 2001.
7. R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann. Adaptation of Service-Based Applications Based on Process Quality Factor Analysis. In *ICSOC/ServiceWave Workshops*, pages 395–404, 2010.
8. A. Kertész, G. Kecskemeti, and I. Brandic. Autonomic SLA-Aware Service Virtualization for Distributed Systems. In *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP*, pages 503–510, 2011.
9. A. Mos, C. Pedrinaci, G. A. Rey, J. M. Gomez, D. Liu, G. Vaudaux-Ruth, and S. Quaireau. Multi-level Monitoring and Analysis of Web-Scale Service based Applications. In *ICSOC/ServiceWave Workshops*, pages 269–282, 2009.
10. O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive Monitoring and Service Adaptation for WS-BPEL. In *Proceeding of the 17th international conference on World Wide Web, WWW*, pages 815–824. ACM, 2008.
11. R. Popescu, A. Staikopoulos, P. Liu, A. Brogi, and S. Clarke. Taxonomy-Driven Adaptation of Multi-layer Applications Using Templates. In *Proceedings of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO*, pages 213–222, 2010.
12. B. Wetzstein, P. Leitner, F. Rosenberg, S. Dustdar, and F. Leymann. Identifying Influential Factors of Business Process Performance using Dependency Analysis. *Enterprise IS*, 5(1):79–98, 2011.
13. A. Zengin, R. Kazhamiakin, and M. Pistore. CLAM: Cross-layer Management of Adaptation Decisions for Service-Based Applications. In *Proceedings of the 9th International Conference on Web Services, ICWS*, 2011.

Appendix E

Facilitating self-adaptable Inter-Cloud management

Facilitating self-adaptable Inter-Cloud management

G. Kecskemeti*, M. Maurer†, I. Brandic†, A. Kertesz*, Zs. Nemeth*, and S. Dustdar†
*MTA SZTAKI Computer and Automation Research Institute
H-1518 Budapest, P.O. Box 63, Hungary
{kecskemeti, keratt, zsnemeth}@sztaki.hu
†Distributed Systems Group
1040 Vienna, Argentinierstr. 8/181-1, Austria
{ivona, maurer, dustdar}@infosys.tuwien.ac.at

***Abstract*—Cloud Computing infrastructures have been developed as individual islands, and mostly proprietary solutions so far. However, as more and more infrastructure providers apply the technology, users face the inevitable question of using multiple infrastructures in parallel. Federated cloud management systems offer a simplified use of these infrastructures by hiding their proprietary solutions. As the infrastructure becomes more complex underneath these systems, the situations (like system failures, handling of load peaks and slopes) that users cannot easily handle, occur more and more frequently. Therefore, federations need to manage these situations autonomously without user interactions. This paper introduces a methodology to autonomously operate cloud federations by controlling their behavior with the help of knowledge management systems. Such systems do not only suggest reactive actions to comply with established Service Level Agreements (SLA) between provider and consumer, but they also find a balance between the fulfillment of established SLAs and resource consumption. The paper adopts rule-based techniques as its knowledge management solution and provides an extensible rule set for federated clouds built on top of multiple infrastructures.**

***Keywords*-Cloud Computing; Knowledge Management; Autonomous; Federations; Infrastructure as a Service**

I. Introduction

Cloud Computing represents a novel computing paradigm where computing resources are provided on demand following the rules established in form of Service Level Agreements (SLAs). SLAs represent the popular format for the establishment of electronic contracts between consumers and providers stating the terms of use, objectives and penalties to be paid in case objectives are violated. Thus, appropriate management of Cloud Computing infrastructures [7], [8], [10], [11] is the key issue for the success of Cloud Computing as the next generation ICT infrastructure [1], [2], [3]. Thereby, the interaction of the system with humans should be minimized while established SLAs with the customers should not be violated. Since Cloud Computing infrastructures represent mega scale infrastructures comprising up to thousands of physical hosts, there is a high potential of energy waste by overprovisioning resources to keep SLA violation levels as low as possible.

Recent related work presents several concepts for the management of competing priorities of both, prevention of violation of established SLAs while reducing energy consumption of the system. As presented in [4], knowledge management techniques have been used to implement an autonomic control loop, where Cloud infrastructures are autonomously managed in order to keep the balance between SLA violations and resource consumption. Thus, the knowledge management (KM) system suggests reactive actions for preventing possible SLA violations and optimizing resource usage, which results in lower energy consumption. However, reactive actions of the system (as presented in [4], [5]) consider only intra-Cloud management, e.g., application or virtual machine (VM) reconfiguration.

On the other hand, there is considerable work in Cloud federation mechanisms without dealing with self-management issues of the system. A Federated Cloud Management (FCM) architecture proposed in [6] acts as an entry point to cloud federations and incorporates the concepts of meta-brokering, cloud brokering and on-demand service deployment. In this paper, we extend FCM by introducing an integrated system for reactive knowledge management and federation mechanisms suitable for on-demand generation and autonomous management of hybrid clouds. Besides intra-Cloud level (i.e., application and intra-VM management), we also target inter-Cloud management, where VMs are instantiated and destroyed on demand to prevent SLA violations and to minimize resource wastage.

This paper is organized as follows: first, we gather related approaches in Section II. In Section III we introduce the architecture for Cloud federations and provide a short overview on its main components. In Section IV, we present the changes and extensions applied to the architecture to accomplish autonomous behavior. Finally, we conclude our research in Section V.

II. Related work

Bernstein et al. [15] defines two use case scenarios that exemplify the problems faced by users of multi-cloud systems. They define the case of VM Mobility where they identify networking, specific cloud VM management interfaces and the lack of mobility interfaces as the three major obstacles. They also discuss a storage interoperability and federation

scenario, in which storage provider replication policies are subject to change when a cloud provider initiates subcontracting. However, they offer interoperability solutions only for low-level functionality of clouds that are not focused on recent user demands, but on solutions for IaaS system operators.

Buyya et al. in [16] suggests a cloud federation oriented, just-in-time, opportunistic and scalable application services provisioning environment called InterCloud. They envision utility-oriented federated IaaS systems that are able to predict application service behavior for intelligent down- and up-scaling infrastructures. They also present a market-oriented approach to offer InterClouds including cloud exchanges and brokers that bring together producers and consumers. Producers are offering domain specific enterprise Clouds that are connected and managed within the federation with their Cloud Coordinator component. Finally, they have implemented a CloudSim-based simulation that evaluates the performance of the federations created using InterCloud technologies. Unfortunately, users face most federation-related issues before the execution of their services, therefore the concept of InterClouds cannot be applied in user scenarios this paper is targeting.

RightScale [9] offers a multi-cloud management platform that enables users to exploit the unique capabilities of different clouds, which has a similar view on Cloud federations to our approach. It is able to manage complete deployments of multiple servers across more clouds, using an automation engine that adapts resource allocation as required by system demand or system failures. They provide server templates to automatically install software on other supported cloud infrastructures. They also advertize disaster recovery plans, low-latency access to data, and support for security and SLA requirements. RightScale users can select, migrate and monitor their chosen clouds from a single management environment. They support Amazon Web Services, Eucalyptus Systems, Flexiscale, GoGrid, and VMware. The direct access to IaaS systmes is performed by the so-called Multi-Cloud Engine, which is supposed to perform brokering capabilities related to VM placement. Unfortunately we are not aware of any publications that detail the brokering operations of these components, therefore we cannot provide any deeper comparisons to our approach.

There has been considerable work on energy efficiency in ICT systems. Their common goal is to attain certain performance criteria for reducing energy consumption. Liu et al. [18] show how to save energy by optimizing VM placement via live migration. Meng et al. [19] try to increase efficient resource by provisioning multiple specific VMs together on a physical machine. Some authors as Kalyvianaki [20] focus on optimizing specific resource type as CPU usage, or only deal with homogeneous resources [21]. While most authors assume a theoretical energy model behind their approaches, Yu [22] targets the more basic question

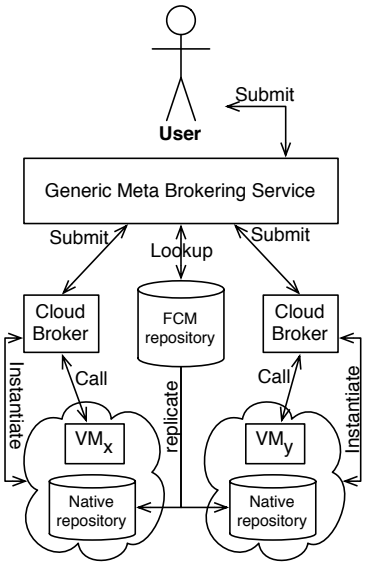


Figure 1. The original Federated Cloud Management architecture

of how to effectively measure energy consumption in Cloud computing environments in a scalable way. Besides, none investigated energy efficiency in Cloud federations.

Considering the use of Knowledge Management Systems (KM) and SLAs, Paschke et al. [23] look into a rule based approach in combination with a logical formalism called ContractLog. It specifies rules to trigger after a violation has occurred, e.g., it obliges the provider to pay some penalty, but it does not deal with avoidance of SLA violations. Others inspected the use of ontologies as knowledge bases (KBs), but only at a conceptual level. Koumoutsos et al. [24] view the system in four layers (i.e., business, system, network and device) and break down the SLA into relevant information for each layer, but they give no implementation details. Bahati et al. [25] also use policies, i.e., rules, to achieve autonomic management. They provide a system architecture including a KB and a learning component, and divide all possible states of the system into so called regions, which they assign a certain benefits for being in this region. However, the actions are not structured and lack a coherent approach. They are mixed together into a single rule, which makes them very hard to manage. On the contrary, we provide a well-structured and extendable approach that also investigates how actions are defined to avoid counteracting recommendations of the KM system.

III. Federated Cloud Management architecture

Figure 1 shows the Federated Cloud Management (FCM) architecture (first introduced in [6]), and its connections to the corresponding components that together represent an interoperable solution for establishing a federated cloud environment. Using this architecture, users are able to execute

services deployed on cloud infrastructures transparently, in an automated way. Virtual appliances for all services should be stored in a generic repository called FCM Repository, from which they are automatically replicated to the native repositories of the different Infrastructure as a Service cloud providers.

Users are in direct contact with the *Generic Meta Brokering Service* (GMBS – [13]) that allows requesting a service by describing the call with a WSDL, the operation to be called, and its possible input parameters. The GMBS is responsible of selecting a suitable cloud infrastructure for the call, and submitting to a Cloud-Broker (CB) in contact with the selected infrastructure. Selection is based on static data gathered from the *FCM Repository* (e.g., service operations, WSDL, appliance availability), and on dynamic information of special deployment metrics gathered by the Cloud-Brokers (see Section IV-B2). The role of GMBS is to manage autonomously the interconnected cloud infrastructures with the help of the Cloud-Brokers by forming a cloud federation.

Cloud-Brokers are set up externally for each IaaS provider to process service calls and manage VMs in the particular cloud. Each Cloud-Broker [14] has its own queue for storing the incoming service calls, and it manages one virtual machine queue for each virtual appliance (VA). Virtual machine queues represent the resources that can currently serve a virtual appliance specific service call. The main goal of the Cloud-Broker is to manage the virtual machine queues according to their respective service demand. The default virtual machine scheduling is based on the currently available requests in the queue, their historical execution times, and the number of running VMs.

Virtual Machine Handlers are assigned to each virtual machine queue and process the VM creation and destruction requests in the queue. Requests are translated and forwarded to the underlying IaaS system. VM Handlers are infrastructure-specific and built on top of the public interfaces of the underlying IaaS. Finally, the Cloud-Broker manages the incoming service call queue by associating and dispatching calls to VMs created by the VM Handler.

As a background process, the architecture organizes virtual appliance distribution with the *automatic service deployment* component [17]. This component minimizes pre-execution service delivery time to reduce the apparent service execution time in highly dynamic service environments. Service delivery is minimized by decomposing virtual appliances and replicating them according to demand patterns, then rebuilding them on the IaaS system that will host the future virtual machine. This paper does not aim to further discuss the behavior of the ASD, however it relies on its features that reduce virtual appliance replication time and transfer time between the FCM and the native repositories.

IV. Self-adaptable Inter-Cloud management architecture

This paper offers two options to incorporate the concepts of knowledge management (KM) systems into the Federated Cloud Management architecture: local and global. Local integration is applied on a per deployed component basis, e.g., every Cloud-Broker utilizes a separate KM system for its internal purposes. In contrast, global integration is based on a single KM system that controls the autonomous behavior of the architectural components considering the available information from the entire cloud federation. In this section, first, we discuss which integration option is best to follow, then we introduce the extensions made to a KM system in order to perform the integration.

A. Knowledge management integration options

When *local* integration is applied, each knowledge manager can make fine-grained changes – e.g., involving actions on non-public interfaces – on its controlled subsystem. First, the meta-broker can select a different scheduling algorithm if necessitated by SLA violation predictions. Next, the Cloud-Broker can apply a more aggressive VM termination strategy, if the greenness of the architecture is more prioritized. Finally, if the storage requirements of the user are not valid any more, the FCM repository removes unnecessarily decomposed packages (e.g., when the used storage space approaches its SLA boundaries, the repository automatically reduces the occupied storage). However, the locally made reactions to predicted SLA violations might conflict with other system components not aware of the applied changes. These conflicts could cause new SLA violation predictions in other subsystems, where new actions are required to maintain the stability of the system. Consequently, local reactions could cause an *autonomic chain reaction*, where a single SLA violation prediction might lead to an unstable system.

To avoid these chain reactions, we investigated *global* integration (presented in Figure 2) that makes architecture-wide decisions from an external viewpoint. High-level integration is supported by a monitoring solution – deployed next to each subcomponent in the system (GMBS, the various Cloud-Brokers and repositories) – that determines system behavior in relation to the settled SLA terms. Global KM integration aggregates the metrics received from the different monitoring solutions, thus operates on the overall architecture and makes decisions considering the state of the entire system before changing one of its subsystems. However, adaptation actions are restricted to use the public operations of the FCM architecture (e.g., new cloud selection requests, new VM and call associations or repository rearrangements). Consequently, the global integration *exhausts adaptation actions* earlier than the local one, because of metrics aggregation and restricted interface use. For instance, if aggregated data hides the cause of a possible

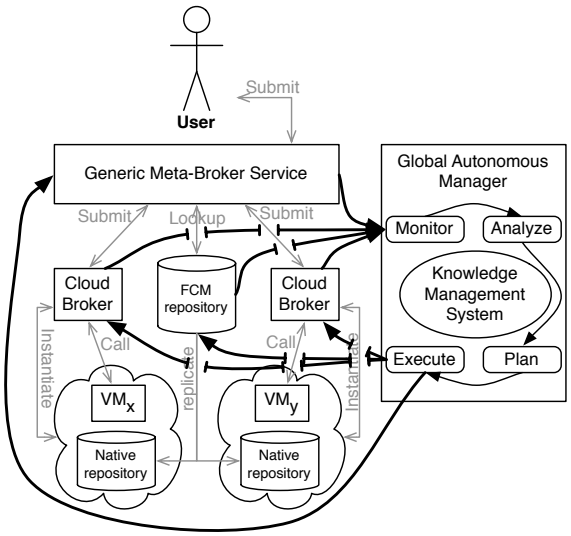


Figure 2. Global integration of the knowledge management system

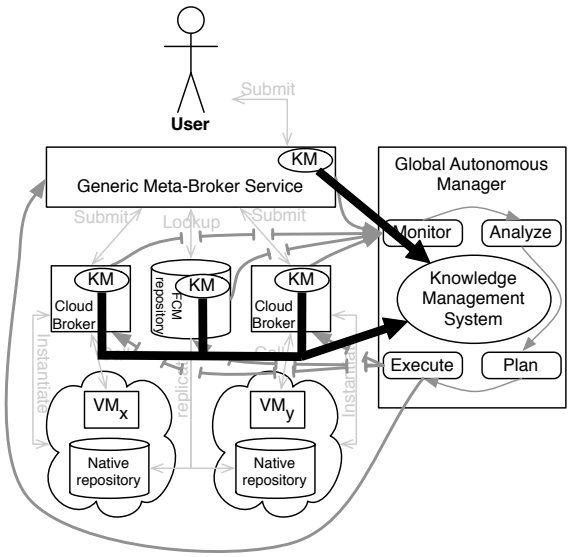


Figure 3. Hybrid integration of the knowledge management system

future SLA violation, then global KM cannot act without user involvement.

In this paper, we propose to use a *hybrid* KM system (revealed in Figure 3) combining both global and local integration options. The hybrid system avoids the disadvantages of the previous solutions by enabling global control over local decisions. In our system, local actions can be preempted by the global KM system by propagating predicted changes in aggregated metrics. Based on predicted changes, the global KM could stop the application of a locally optimal action and prevent the autonomic chain reaction that would follow the local action. On the other hand, if the global system does not stop the locally optimal action, then it enables the execution of more fine-grained actions postponing adaptation action exhaustion.

Action	Involved component	Integration
Reschedule calls	Meta-Broker	Global
Rearrange VM queues	Cloud-Broker	Global
Extend/Shrink VM Queue	Cloud-Broker	Local
Rearrange VA storage	FCM repository	Global
Self-Instantiated Deployment	Service instances	Local

Table I
SUMMARY OF AUTONOMOUS ACTIONS

B. Knowledge management system extensions

This subsection first lists the possible autonomic actions in our KM system, then it analyzes the collected monitoring data that can indicate the need for autonomous behavior. Finally, based on these indicators, we conclude with the rules triggering the adaptation in our FCM components.

1) Actions: Based on the affected components, the architecture applies four basic types of actions on unacceptable behavior. First, at the meta-brokering level, the system can organize a *rescheduling of several service calls*. E.g., the autonomous manager could decide to reschedule a specific amount of queued calls – $c \in Q_x$, where c refers to the call, and Q_x specifies the queue of the Cloud-Broker for IaaS provider x. Consequently, to initiate rescheduling, the knowledge manager specifies the amount of calls (N_{cr}) to be rescheduled and the source cloud (C_s) from which the calls need to be removed. Afterwards, the meta-broker evaluates the new situation for the removed calls, and schedules them to a different cloud, if possible.

Second, at the level of cloud brokering, the system could decide either to *rearrange the VM queues* of different Cloud-Brokers, or alternatively to *extend or shrink the VM queue* of a specific Cloud-Broker. *VM queue rearrangement* requires global KM integration in the system so it can determine the effects of the queue rearrangement on multiple infrastructures. The autonomous manager accomplishes rearrangement by destructing VMs of a particular virtual appliance in a specific cloud and requesting new VMs in another one. Consequently, the autonomous manager selects the virtual appliance (VA_{arr} – appliance to rearrange) that has the most affected VMs. Then it identifies the amount of virtual machines (N_{vmtr}) to be removed from the source cloud (C_s) and instantiated in a more suitable one (C_d).

The queue rearrangement operations have their counterparts also in case of local KM integration. The VM *queue extension and shrinking* operations are local decisions that are supported by energy efficiency related decisions. In case of queue shrinking, some of the virtual machines controlled by the local Cloud-Broker are destructed. However, under bigger loads, virtual machines could be in the process of

performing service calls. Therefore, the autonomous manager can choose between the three VM destruction behaviors embedded into the Cloud-Brokers: (i) destroy after call completed, (ii) destroy right after request and put the call back to the local service call queue and finally, (iii) destroy right after request and notify the user about call abortion. As a result, the autonomous manager specifies the number of VMs to extend (N_{ex}) or shrink (N_{shr}) the queue with and the destruction strategy (S_{dest}) to be used.

Third, on the level of the FCM repository, the autonomous manager can make the decision to *rearrange virtual appliance storage* between native repositories. This decision requires the FCM repository either to remove appliances from the native repositories, or to replicate its contents to a new repository. Appliance removal is only feasible, if one of the following cases are met: (i) the hosting cloud will no longer execute the VA, (ii) the hosting cloud can download the VA from third party repositories or finally, (iii) the appliance itself was based on an extensible appliance that is still present in the native repository of the hosting cloud. The objective of the rearrangement is to reduce the storage costs in the federation at the expense of increased virtual machine instantiation time for VMs of the removed appliances. Conclusively, the rearrangement decision should involve the decision on the percentage (N_{repr}) of the reduced or replicated appliances that should participate in the rearrangement process.

Finally, when virtual appliances are built with embedded autonomous capabilities (internal monitoring, KM system etc.), then virtual machines based on them are capable of *self-initiated deployment*. If a service instance gets either overloaded or dysfunctional according to its internal monitoring metrics, then the instance contacts the local Cloud-Broker to instantiate a new virtual machine just like the instance is running in. In case of overloading, the new instance will also be considered for new *Call→VM* associations. In case of dysfunctional instances, the system creates a proxy service inside the original VM replacing the original service instance. This proxy is then used to forward the requests towards the newly created instance until the current VM is destructed.

2) Monitored metrics: After analyzing the various autonomous actions that the KM system can exercise, we investigated the monitoring system and the possible metrics to be collected for the identification of those cases when the architecture encounters unsatisfactory behavior. Currently, we monitor and analyze the behavior of Cloud-Brokers, the FCM repository and individual service instances.

Since Cloud-Brokers represent the behavior of specific IaaS systems, most of the measurements and decisions are made based on their behavior. All measurements are related to the queues of the Cloud-Broker; therefore we summarize their queuing behavior. Cloud-Brokers offer two types of queues: the call queue (Q_x, where x identifies the specific Cloud-Broker that handles the queue) and the VM queues ($VMQ_{x,y}$, where y identifies the specific service – or appliance VA_y – the queued VMs are offering). The members of the call queue represent the service calls that a Cloud-Broker needs to handle in the future (the queue is filled by the meta-broker and emptied by the Cloud-Broker through associating a call with a specific VM). On the other hand, VM queues are handled on a more complex way: they list the currently handled VMs offering a specific service instance. Consequently, the Cloud-Brokers maintain VM queues for all service instances separately. Entries in the VM queues are used to determine the state of the VMs:

$$State : VM \rightarrow \begin{cases} WAITING \\ INIT \\ RUNNING.AVAILABLE \\ RUNNING.ACQUIRED \\ CANCEL \end{cases} \quad (1)$$

- **Waiting:** the underlying cloud infrastructure does not have resources to fulfill this VM request yet.
- **Init:** the VM handler started to create the VM but it has not started up completely yet.
- **Running and available:** the VM is available for use, the Cloud-Broker can associate calls to these VMs only.
- **Running and acquired:** the VM is associated with a call and is processing it currently.
- **For cancellation:** the Cloud-Broker decided to remove the VM and stop hosting it in the underlying infrastructure.

Based on these two queues the monitor collects the metrics listed in the following paragraphs.

To support decisions for *service call rescheduling*, the system monitors the call queue for all available Cloud-Brokers for a specific service call s:

$$q(x, s) := \{c \in Q_x : (type(c) = s)\}, \quad (2)$$

where $type(c)$ defines the kind of the service call c is targeting.

Call throughput measurement of available Cloud-Brokers is also designed to assist *call rescheduling*:

$$throughput(x) := \frac{1}{max_{c \in Q_x}(waitingtime(c))}, \quad (3)$$

where $waitingtime(c)$ expresses the time in sec a service call has been waiting in the specific Q.

We define the average waiting time of a service s by

$$awt(s, Q_x) := \frac{\sum_{c \in q(x,s)} waitingtime(c)}{|q(x, s)|}, \quad (4)$$

and the average waiting time of a queue by

$$awt(Q_x) := \frac{\sum_{c \in Q_x} waitingtime(c)}{|Q_x|}. \quad (5)$$

To distinguish the Cloud-Brokers, where *VM queue rearrangements* could occur, we measure the number of service instances that are offered by a particular infrastructure:

$$vms(x,s) := \{vm \in VMQ_{x,s} : State(vm) = RUNNING.AVAILABLE \vee State(vm) = RUNNING.ACQUIRED\} \quad (6)$$

The call/VM ratio for a specific service managed by a specific Cloud-Broker:

$$cvmratio(x,s) := \frac{|q(x,s)|}{|vms(x,s)|} \quad (7)$$

This ratio allows the global autonomous manager to plan *VM queue rearrangements* and equalize the service call workload on the federated infrastructures. When applied with the local KM system, this ratio allows the system to decide on *extending and shrinking the VM queues* of particular services and balance the service instances managed by the local Cloud-Broker.

The load of the infrastructure managed by a specific Cloud-Broker:

$$load(x) := \frac{\sum_{\forall s} |vms(x,s)|}{\sum_{\forall s} |VMQ_{x,s}|} \quad (8)$$

The load analysis is used for *VM queue rearrangements* in order to reduce the number of waiting VMs in the federation. When applied locally, along with the call/vm ratio the load analysis is utilized to determine when to *extend or shrink the VM queues* of various services. As a result, Cloud-Brokers could locally reorganize their VM structures that better fit the current call patterns.

To support the remaining autonomous actions, the FCM repository and individual service instances are also monitored. First, the system monitors the accumulated storage costs of a virtual appliance in all the repositories ($r \in R$) in the system (expressed in US dollars/day):

$$stcost(VA_s) := \sum_{\forall r} locstcost(r, VA_s), \quad (9)$$

where $locstcost(r, VA_s)$ signifies the local storage cost at repository r for appliance VA_s. To better identify the possible appliance storage rearrangements the system also analyzes the usage rate of appliances in the different repositories expressed in the number of times the VMs based on the appliance have changed status from $INIT$ to $RUNNING.AVAILABLE$ in a single day ($deployfreq(r, VA_s)$).

Finally, individual services are monitored to support self-instantiated deployment. Here we analyze the service availability (expressed as the % of time that the instance is available for external service calls) of the specific service instance deployed in the same VM where the monitoring system is running.

3) Basic rules for applying actions: We decided to formulate the knowledge base (KB) as a rule-based system. Rules are of the form "WHEN condition THEN action" and can be implemented e.g., using the Java rule engine Drools [26]. We define several rules based on the previously defined measurements and actions, and present them in Drools-related pseudo code. The working memory of the KM system, which is the main class for using the rule engine at runtime, does not only consist of the specified rules, but also of the objects whose knowledge has to be modeled, and that are currently active in the Cloud federation (like a Cloud-Broker, the native repository, different queues, etc.). These objects are typically modeled as Java classes, and thus referred to as Cloud-Broker(), NativeRepository(), etc.

Figure 4 shows the rule for *rescheduling service calls*. Line 1 states the unique name the rule can be identified with in the KB. This way, rules can be dynamically altered or replaced if different global behavior due to changing high-level policies (i.e., changing from energy efficient to SLA performant) is required. Lines 3-4 state the conditions that have to be fulfilled to trigger the actions in lines 6-9. At first, we look for a Cloud-Broker C_s (line 3), whose throughput falls below the average of all the queues' throughputs ($mean()$) plus a multiple of their standard deviation ($std()$, line 4). If such C_s is found, the rule is executed. We have to decide to which Cloud C_d (line 6) to move N_{cr} service calls (line 7), and finally invoke the appropriate public interface methods of the Cloud brokers at stake (lines 8-9). As C_d we choose the Cloud with maximum throughput. The equalizeQs() method (line 7) tries to equal out the average waiting times of the queues of C_s and C_d. It takes the last service call $\hat{s}$ out of Q_s, retrieves its average waiting time $awt(\hat{s}, Q_s)$ and calculates the new estimated average waiting time for Q_s and Q_d by $awt(Q_s) := awt(Q_s) - awt(\hat{s}, Q_s)$ and $awt(Q_d) := awt(Q_d) + awt(\hat{s}, Q_d)$, respectively. Then it adds $\hat{s}$ to Q_d. It continues this procedure as long as $awt(Q_s) \geq awt(Q_d)$, and returns the number of service calls that have been hypothetically added to Q_d. The rule could then either really add the chosen calls to C_d as presented in line 9, or return them to the meta-broker

Figures 5 and 6 show possible rules for removing VAs from a Cloud's native repository due to high local or global costs, respectively. Both rules try to find a repository r and a VA VA_x that have been inserted into the working memory of the rules engine (lines 3-4), and remove the specified VA from the repository (line 8), when certain conditions hold. In Figure 5 the removal action is executed when two conditions hold: First, the local storage cost of the VA at the specified resource exceeds a certain threshold. The threshold is calculated as the average local storage costs at all repositories for the same VA plus a multiple of its standard deviation. Second, the deploy frequency of the VA

1 rule "Reschedule calls"
2 **WHEN**
3 $C_s : Cloudbroker()$
4 $throughput(x) < mean(throughput(.)) + \delta \cdot std(throughput(.))$
5 **THEN**
6 $C_d := \arg\max throughput(.)$
7 $N_{cr} := equalizeQs(C_s, C_d)$
8 $calls := remove(N_{cr}, C_s)$; //removes last N_{cr} entries in Q_{C_s}.
9 $add(calls, C_d)$;

Figure 4. Rule for rescheduling calls

1 rule "Remove VA from native repository due to high local costs"
2 **WHEN**
3 r : NativeRepository()
4 VA_x : VirtualAppliance()
5 $locstcost(r, VA_x) > mean(locstcost(., VA_x)) + \delta \cdot std(locstcost(., VA_x))$
6 $deployfreq(r, VA_x) < mean(deployfreq(., VA_x))$
7 **THEN**
8 $remove(VA_x, r)$ //removes VA_x from native repository r

Figure 5. Rule for removing VA from native repository of a specific Cloud due to high local costs

at this repository falls below a certain threshold, which is the mean deploy frequency of the VA at all repositories. In short, the VA is instantiated less often than other VAs, but its cost is higher than for other VAs, so the VA should be removed. Figure 6 takes a global perspective and checks whether the overall storage cost for the VA exceeds a certain threshold (defined similarly as with Figure 5, line 5). Then, the VA is removed from the repository that has the lowest deployment frequency (line 6).

The remaining rules can be specified according to the

1 rule "Remove VA from native repository due to high global costs"
2 **WHEN**
3 r : NativeRepository()
4 VA_x : VirtualAppliance()
5 $stcost(VA_x) > mean(stcost(.)) + \delta \cdot std(stcost(.))$
6 $r_{min} : \arg\min deployfreq(., VA_x)$
7 **THEN**
8 $remove(VA_x, r_{min})$ //removes VA_x from native repository r_{min}

Figure 6. Rule for removing VA from native repository of a specific Cloud due to high global costs

actions and measurements as explained before. However, their specific parameters may have heavy impact on the overall performance of the system. These parameters are to be learned by the KM system. In our future work, we plan to evaluate the system performance with the extension of the simulation engine presented in [4].

V. Conclusion

This paper presented an approach to extend federated cloud management architectures with autonomous behavior. Our research uses knowledge management systems to facilitate the decision making process of the classical monitoring-analysis-planning-execution loop. Using the FCM architecture as the basis of our further investigations, we analyzed different approaches to integrate the knowledge management system within this architecture, and found a hybrid approach that incorporates fine-grained local adaptation operations with options for high-level override. Then this research pinpointed the adaptation actions and their possible effects on cloud federations. Finally, we established metrics that could indicate possible SLA violations in federations, and defined rules that could trigger adaptation actions in the case of predicted violations.

Regarding future works, we plan to investigate more the green aspects in the autonomous behavior of cloud federations. We also aim at defining new rules for advanced action triggers and evaluate the applicability of another knowledge management approaches like case based reasoning. Finally, we also plan to investigate the effects of the autonomous behavior on the overall performance of the cloud federation on an experimental system.

Acknowledgment

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 under grant agreement N^o RI-283481 (SCI-BUS), also from the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 – Foundations of Self-Governing ICT Infrastructures (FoSII).

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 2009, vol. 25, issue. 6, pp. 599–616.

[2] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Computer Communication Review*. vol 39, 1, pp. 50–55, 2008.

[3] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, vol. 15, pp. 313–341, December 2008.

[4] M. Maurer, I. Brandic and R. Sakellariou. Simulating Autonomic SLA Enactment in Clouds using Case Based Reasoning. In *Towards a Service-Based Internet, Third European Conference*, ServiceWave 2010, Ghent, Belgium, December, 2010 pp. 25-37

[5] M. Maurer, I. Brandic and R. Sakellariou. Enacting SLAs in Clouds Using Rules. In *Proceedings of the 17th International Euro-Par Conference on Parallel and Distributed Computing*, Bordeaux, France, September, 2011

[6] A. Cs. Marosi, G. Kecskemeti, A. Kertesz and P. Kacsuk. FCM: an Architecture for Integrating IaaS Cloud Systems. In *Proceedings of The Second International Conference on Cloud Computing, GRIDs, and Virtualization*. Rome, Italy. September, 2011.

[7] Amazon Web Services LLC. Amazon elastic compute cloud. http://aws.amazon.com/ec2/, 2009.

[8] Rackspace Cloud. http://www.rackspace.com/cloud/, 2011.

[9] RightScale website. http://www.rightscale.com/, 2011.

[10] Eucalyptus cloud. http://www.eucalyptus.com/, 2011.

[11] OpenNebula cloud. http://opennebula.org/, 2011.

[12] The World Wide Web Consortium. http://www.w3.org/TR/wsdl, 2009.

[13] A. Kertesz and P. Kacsuk. GMBS: A new middleware service for making grids interoperable. *Future Generation Computer Systems*, 2010, vol. 26, issue 4, pp. 542–553.

[14] A. Cs. Marosi and P. Kacsuk. Workers in the clouds. In *Proceedings of the 19th Euromicro Conference on Parallel, Distributed and Network-Based Processing* (PDP 2011), Ayia Napa, Cyprus, February 9-11, 2011. pp. 519–526

[15] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond and M. Morrow. Blueprint for the Intercloud Protocols and Formats for Cloud Computing Interoperability. In *Proceedings of The Fourth International Conference on Internet and Web Applications and Services* (2008, pp. 328-336)

[16] R. Buyya, R. Ranjan and R. N. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. *Lecture Notes in Computer Science: Algorithms and Architectures for Parallel Processing*. Volume 6081, 20 pages, 2010.

[17] G. Kecskemeti, G. Terstyanszky, P. Kacsuk, and Zs. Nemeth. An Approach for Virtual Appliance Distribution for Service Deployment. *Future Generation Computer Systems*, 2011, vol. 27, issue 3, pp 280–289.

[18] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen. Greencloud: a new architecture for green data center. In *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, ICAC-INDST '09, pages 29–38, New York, NY, USA, 2009. ACM.

[19] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 11–20, New York, NY, USA, 2010. ACM.

[20] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 117–126, New York, NY, USA, 2009. ACM.

[21] B. Khargharia, S. Hariri, and M. S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, 2008.

[22] Y. Yu and S. Bhatti. Energy measurement for the cloud. *Parallel and Distributed Processing with Applications, International Symposium on*, 0:619–624, 2010.

[23] A. Paschke and M. Bichler. Knowledge representation concepts for automated SLA management. *Decision Support Systems*, 46(1):187–205, 2008.

[24] G. Koumoutsos, S. Denazis, and K. Thramboulidis. SLA e-negotiations, enforcement and management in an autonomic environment. *Modelling Autonomic Communications Environments*, pages 120–125, 2008.

[25] R. M. Bahati and M. A. Bauer. Adapting to run-time changes in policies driving autonomic management. In *ICAS '08: Proceedings of the 4th Int. Conf. on Autonomic and Autonomous Systems*, Washington, DC, USA, 2008. IEEE Computer Society.

[26] Drools, The Business Logic integration Platform. www.drools.org, 2011.

Appendix F

Adaptation of Web Service Interactions using Complex Event Processing Patternst

Adaptation of Web Service Interactions using Complex Event Processing Patterns

Yéhia Taher, Michael Parkin, Mike P. Papazoglou, Willem-Jan van den Heuvel

European Research Institute for Service Science, Tilburg University, The Netherlands
{y.taher|m.s.parkin|mikep|wjheuvel}@uvt.nl

Abstract. Differences in Web Service interfaces can be classified as signature or protocol incompatibilities, and techniques exist to resolve one or the other of these issues but rarely both. This paper describes *complex event processing* approach to resolving both signature and protocol incompatibilities existing between Web Service interfaces. The solution uses a small set of operators that can be applied to incoming messages individually or in combination to modify the structure, type and number of messages sent to the destination. The paper describes how CEP-based adapters, deployable in CEP engines, can be generated from automata representations of the operators through a standard process and presents a proof-of-concept implementation.

1 Introduction

Web services allow the integration of distributed software through standard interface definition languages, transport mechanisms and aspects such as security and quality of service. Web Service interfaces (i.e., WSDL, BPEL, etc.) define the messages and protocol that should be used to communicate with the service [7]. However, if two services wish to interact successfully, they must both support the same messages and protocol through the implementation of compatible WSDL and/or BPEL documents. Unfortunately, this is difficult to achieve in practice; Web Services are often developed independently and follow different standards or approaches in constructing their interfaces and Web Service compositions will often use them in ways that were not foreseen in their original design and construction [2,3]. Therefore, it is likely that most Web Services will be incompatible as services will not support the same interface.

This is a short paper describing a general approach to resolving differences between Web Services protocols through the use of Complex Event Processing (CEP [4]) technology. Specifically, we extend our previous work [10,11] to show how a small set of general operators can be used to match the messages from one service with those of another. By using a *continuous query engine* running within a CEP platform, we demonstrate signature and protocol adaptation between Web Services in a proof-of-concept implementation.

This paper is structured as follows: Section 2 describes our CEP-based approach to signature and protocol adaptation; Section 3 introduces the operators used to resolve differences in Web Service protocols; Section 4 presents the

CEP solution and a proof-of-concept implementation; Section 5 compares related work; Section 6 contains conclusions and our plans for future work.

2 Approach

Incompatibilities between Web Service protocols can be classified as either [2,3]:

1. Signature Incompatibilities arise due to the differences between services in expected message structure, content and semantics. In Web Services, XML schema provides defines a set of 'built-in' types to allow the construction of complex input and output message types from these primitives. This flexibility in constructing message types in XML often means that a message from one Web Service will not be recognized by another and, therefore, there is a requirement to provide some function that maps the schema of one message to another [6].

2. Protocol Incompatibilities are found when Web Services wish to interact but are incompatible because they support of different message exchange sequences. For example, if two services perform the same function, e.g., accept purchase orders, but Service A requires a single order containing one or more items while Service B expects an order message for each item, there is a mismatch in their communication protocols that must be resolved in order for them to interoperate. To solve these incompatibilities, there are two approaches: a) to force one of the parties to support the other's interface, or b) to build an adapter that receives messages, converts them to the correct sequence and/or maps them into a desired format and sends them to their destination. However, both of these solutions are unsatisfactory; imposing the development of an interface for each target service can lead to an organization having to maintain a different client for each service it uses, and the implementation of bespoke ad-hoc point-to-point adapters is costly and not-scalable.

Our solution is to automate the generation of adapters so the process is repeatable and scalable and remove the necessity to build costly bespoke adapters. Our approach for generating adapters is described in [9], which presents an algorithm for detecting signature and protocol incompatibilities between two Web Service protocol descriptions (i.e., interfaces) and a CEP-based mediation framework to perform protocol adaptation practically. This paper completes the mediation framework by showing how the incompatibilities found between two Web Service protocols, classified according to a set of basic transformation patterns by the algorithm in [9], can be transformed into configurable automata *operators* which are used to generate *adapters*.[1] In Section 3 we describe the operators required for each transformation pattern then in Section 4 show how they are used to generate CEP adapters and deployed to a CEP engine.

Complex Event Processing technology can discover relationships between events through the analysis and correlation of multiple events and triggers and take actions (e.g., generate new events) from these observations. CEP does this

[1] Adapters are therefore the components that resolve sets of incompatibilities found between two services and are aggregations of predefined *operators* who's purpose is to resolve individual, specific incompatibilities.

by, for example, modeling event hierarchies, detecting causality, membership and/or timing relationships between events and abstracting event-driven processes into higher-level concepts [4]. CEP platforms allow streams of data to run through them to detect conditions that match the *continuous computational queries* (CCQs, written in a Continuous Computation language, or CCL) as they occur. As a result, CEP has an advantage in performance and capacity compared to traditional approaches: CEP platforms typically handle many more events than databases and can process throughputs of between 1,000 to 100k messages per second with low latency. These features make a CEP platform an excellent foundation for situations that have real-time business implications.

In the context of Web Services, *events* occur when SOAP messages are sent and received. Therefore, CEP adaptation requires the platform to consume incoming messages, process them and send the result to its destination. However, a CCQ written for a particular adaptation problem is similar to the bespoke adapter solution described earlier. To offer a universal solution and a scalable method for Web Service protocol adaptation, we automate the generation and deployment of CCQs to transform incoming message(s) into the required output message format(s) using the predefined set of transformation operators.

3 Operators

[3] describes five basic transformation patterns that can reconcile protocol mismatches. We have developed an operator for each of these patterns that can be applied individually or in combination to incoming messages to achieve a transformation in both the structure, type and number of messages sent to the destination — i.e., to resolve both signature and protocol incompatibilities.

The operators developed for each of the transformation patterns are: *Matchmake*, which translates one message type to another, solving the *one-to-one* transformation; *Split*, a solution for the *one-to-many* pattern, which separates one message sent by the source into two or more messages to be received separately; the *Merge* operator is the opposite of the *Split* operator (i.e., it performs a *many-to-one* transformation) and combines two or more messages into a single message; the *Aggregate* operator is used when two or more of the same message from the source service interface correspond to one message at the target service and is a solution for the *one*$^+$*-to-one* transformation; finally, *Disaggregate* performs the opposite function to *Aggregate* operator.

Following [9], the operators are represented as *configurable automata*. Transitions between states represent both observable and non-observable actions. *Observable actions* describe the behavior of the operator vis-à-vis the service consumer and provider, i.e., an action is observable if it is a message consumption or transmission event. *Unobservable actions* describe the internal transitions of the operator, such as the transformation of a messages contents, and are performed transparently to the source and target services.

Transitions caused by observable actions are denoted as $<a,?/!m,a'>$, where a is the starting state and a' the end state following the consumption *(?)* or

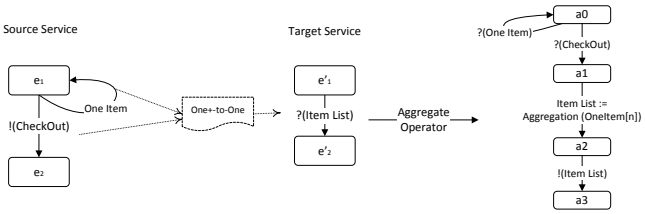


Fig. 1. The Aggregate Operator

transmission *(!)* of message *m*. An unobservable action is denoted as $<a, \psi, a'>$, where *a* and *a'* are the start and end states following internal action ψ.

For reasons of space it is not possible to describe all five operators in detail and we have chosen the *Aggregate* operator to illustrate how they work. Figure 1 shows the operator to resolve one^{+}*-to-one* incompatibilities between services, e.g., when a customer submits a purchase order for each item but the retailer expects a list of all items together. To resolve this incompatibility the aggregate operator consumes and stores *?OneItem* messages until it receives the message *(?CheckOut)* indicating all messages have been sent. The operator aggregates the stored messages into a list of items message using *ItemList=Aggregation(OneItem[n])* and forwards the new message using *!(ItemList)*.

4 CEP-Based Adaptation

4.1 General Principles

The adaptation of interactions between source and target services is specified using automata, therefore deploying them as CEP adapters requires their translation into continuous queries. To do this, we modeled message consumption and transmission actions as events. For each message type consumed or transmitted we create an input or output stream. A continuous query subscribes to the input stream of messages it wants to adapt and publishes the adapted message(s) to the corresponding output stream(s). For convenience, we name the input/output stream the same name as the message it consumes or transmits. This method allows a CEP engine to intercept messages exchanged between two services, to detect patterns of incompatibilities and implement corresponding adaptation solution, i.e., combinations of the operators encoded as continuous queries.

4.2 Conceptual Architecture

Figure 2 illustrates the conceptual architecture of the CEP implementation that translates the adapter specified in automata into continuous queries, i.e., via `Automata` → `Continuous Queries`. This includes the creation of input and output streams for the continuously running queries in the CEP engine, waiting for messages arriving through input streams.

In the second step, the `SOAP Message Interceptor`'s role is to control the exchange of messages between the two services. Upon receipt of a message, the

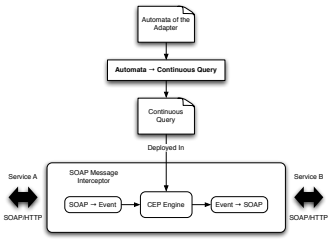


Fig. 2. Conceptual CEP Adaptation Architecture

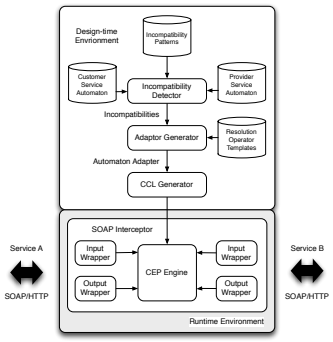


Fig. 3. Architectural Framework for the CEP Solution

Interceptor sends it to the input stream with the same name (through `SOAP` $\rightarrow$ `Event`). The message received is published as an event and is consumed by the query that subscribes to the input stream. The message(s) produced as a result of applying the operators is published to the corresponding output stream. Once on the output stream, the message is consumed by the SOAP message interceptor (through `Event` $\rightarrow$ `SOAP`) and sent to the target.

Figure 4 shows the transformation of the *Aggregation* automata to a continuous query. First, an input stream and a window to store messages arriving on the input stream are created for action *?(m1)* and an input stream is created for action *?(m2)*. The aggregation query is then specified: it subscribes to the window where messages from action *?(m1)* are stored and to the input stream for action *!(m2)* actions. After an *!(m2)* action, messages in the window with the same correlation criteria as new message are aggregated into a single message, the input messages are removed and the result is published to the output stream.

Figure 5 shows a concrete example where messages arriving through the input stream *Order_In* are stored in the window *Order_Win*. When the message arrives through *CheckOut_In* indicating order number #03203 is complete, messages in *Order_Win* with the same order number (#03203, the correlation criteria) are aggregated into a single message. The final message, containing *Item1* and *Item2*, is published to *Order_Out*.

4.3 Proof of Concept

This section illustrates the practical generation and deployment of CEP-based adapters using *model transformation*. It has two stages: the *design phase* models the adapter using operator automata through the use of an incompatibility detection process to produce a platform independent model, whilst the *transformation phase* takes the platform independent model to produce the adapter as a CCQ for a CEP engine, i.e, a platform specific model.

Figure 3 shows the framework for the automatic generation of adapters. If two incompatible services, A and B, wish to communicate, at design-time an adaptor

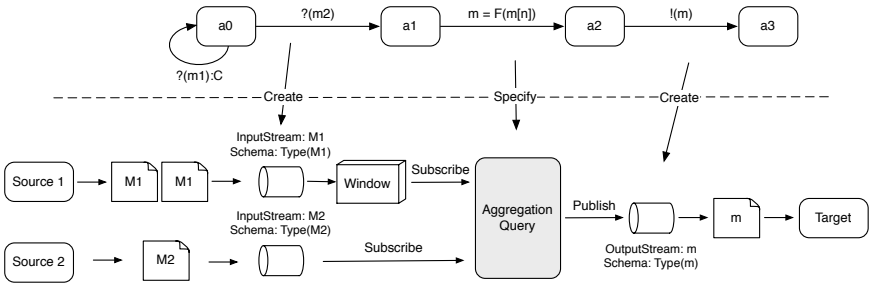


Fig. 4. Aggregate Translation

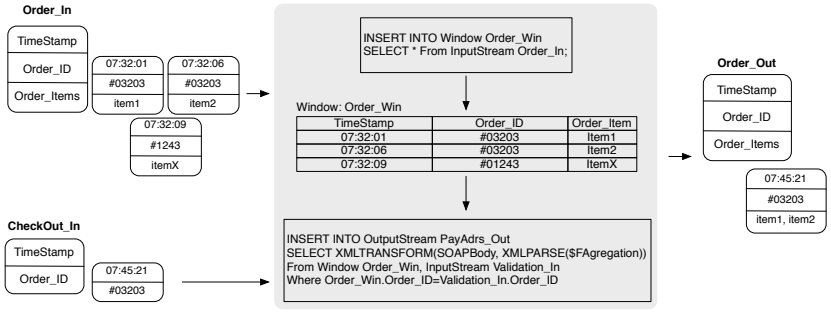


Fig. 5. Aggregate Flow

can be generated for the runtime CEP engine by classifying the incompatibilities between two service interfaces (using the method described in [9]) and using them to construct the adapter using *resolution operator templates* (described in Section 3). The resulting adapter is converted into the CEP engine's continuous computation language (CCL) that is deployed at run-time within a SOAP message interceptor to provide a message serialization/deserialization capability.

The **Design Time Environment** is used to instantiate the template operators described in Section 3 so they can be used in a specific Web Service protocol adaptation. The design-time tools can be used to develop strategies for dealing with complex adaptation situations by allowing the composition of the template operators. In these cases, the designers of the adaptation must identify what adaptations are required between two services and use a graphical user interface (the *Design Tool* shown) to wrap the corresponding composition of operators in a map. Maps are exported to the CCQ code generation tool that includes a compiler to produce a CEP execution-time module (i.e., the CCQ) which is then loaded into the CEP execution engine.

The **Run Time Environment** contains a CEP platform with a continuous query engine and a set of SOAP message integration layers to allow it to send and receive messages to and from Web Services. The continuous query engine

provides the capability for the system to receive, process, correlate and analyze SOAP messages against a CCQ. However, since Web services communicate through the use of SOAP messages, intermediate adapters are required to provide entry and exit points to the engine. These intermediate adapters are of two types: input and output wrappers. An input wrapper receives SOAP messages from the source's service interface and transforms it to the representation appropriate for the CEP engine and then sends it to the engine. Similarly, an output wrapper receives events produced by the engine and transforms it to a SOAP message before forwarding the message to the target service.

4.4 Demonstration & Experimentation

A demonstration of our prototype can be seen at: `http://www.youtube.com/watch?v=g05ciEPZ_Zc`.

5 Related Work

As [3] describes, there are many commercial tools to achieve Web Service *signature mediation* and solve signature incompatibilities, including: Microsoft's Biztalk mapper[2], Stylus Studio's XML Mapping tools[3], SAP's Exchange Infrastructure (XI) Mapping Editor[4] and Altova's MapForce[5]. Academic research also exists in resolving signature incompatibilities through the use of semantic web technology (i.e., OWL), such as that described in [5] that presents a "context-based mediation approach to [...] the semantic heterogeneities between composed Web services", and the Web Service Modeling Ontology (WSMO) specification [8] that provides a foundation for common descriptions of Web Service behavior and operations. This research does not attempt to resolve the associated problem of protocol incompatibility, however.

Active research is also being performed into the adaptation of web service protocols, although all work we have surveyed does not tackle both problems of signature and protocol incompatibility and all use different approaches to the CEP-based technique presented. For example, although [3] presents mediation patterns together with corresponding BPEL templates, a technique and engineering approach for semi-automatically identifying and resolving identifying protocol mismatches and a prototype implementation (the *Service Mediation Toolkit*), it does not solve the signature adaptation problem. Similarly, [2] "discusses the notion of *protocol compatibility* between Web Services" and [1] again only "focusses on the protocol mismatches, leaving data mismatches apart" — i.e., they present solutions to protocol mismatches and do not tackle the associated problem of signature incompatibility. Our chosen approach solves both signature and protocol incompatibilities.

[2] `http://www.microsoft.com/biztalk/en/us/default.aspx`
[3] `http://www.stylusstudio.com/xml_mapper.html`
[4] `http://www.sdn.sap.com/irj/sdn/nw-xi`
[5] `http://www.altova.com/mapforce/web-services-mapping.html`

6 Conclusion

Web service incompatibilities are found in either their message signatures or protocols. This paper presents an CEP approach to adapt Web Service interactions and resolve these conflicts. Using predefined operators represented as configurable automata allows us to automatically CEP generate adapters capable of intercepting incoming messages sent between services and adapting their structure, type and number into the desired output message(s). Our future work will be in two areas: (i) performing extensive testing on real services, and (ii) developing tools to assist service designers to generate adapters.

Acknowledgment: The research leading to these results has received funding from the European Community's Seventh Framework Program [FP7/2007–2013] under grant agreement 215482 (S-CUBE). We thank Marie-Christine Fauvet, Djamal Benslimane and Marlon Dumas for their comments and contributions on earlier stages of this work.

References

1. Ardissono, L., Furnari, R., Petrone, G., Segnan, M.: Interaction Protocol Mediation in Web Service Composition. International Journal of Web Engineering and Technology 6(1), 4–32 (2010)
2. Dumas, M., Benatallah, B., Nezhad, H.R.M.: Web Service Protocols: Compatibility and Adaptation. IEEE Data Engineering Bulletin 31, 40–44 (2008)
3. Li, X., Fan, Y., Madnick, S., Sheng, Q.Z.: A Pattern-Based Approach to Protocol Mediation for Web Services Composition. Information & Software Technology 52(3), 304–323 (2010)
4. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman (2001)
5. Mrissa, M., Ghedira, C., Benslimane, D., Maamar, Z., Rosenberg, F., Dustdar, S.: A Context-Based Mediation Approach to Compose Semantic Web Services. ACM Transactions on Internet Technology (TOIT) 8(1), 1–23 (2008)
6. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F.: Semi-Automated Adaptation of Service Interactions. In: Proceedings of the 16th international conference on World Wide Web. pp. 993–1002 (2007)
7. Papazoglou, M.: Web Services: Principles & Technology. Pearson Education (2008)
8. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology 1(1), 77–106 (2005)
9. Taher, Y., Aït-Bachir, A., Fauvet, M.C., Benslimane, D.: Diagnosing Incompatibilities in Web Service Interactions for Automatic Generation of Adapters. In: Proceedings of the 23rd International Conference on Advanced Information Networking and Applications (AINA-09). pp. 652–659 (2009)
10. Taher, Y., Marie-Christine, F., Dumas, M., Benslimane, D.: Using CEP TEchnology to Adapt Messages Exchanged by Web Services. In: Proceedings of the 17th International Conference on the World Wide Web (WWW 2008). pp. 1231–1232. Beijing, China (April 2008)
11. Taher, Y., Nguyen, D.K., van den Heuvel, W.J., Ait-Bachir, A.: Enabling Interoperability for SOA-Based SaaS Applications Using Continuous Computational Language. In: Proceedings of the 3rd European ServiceWave Conference. pp. 222–224. Ghent, Belgium (December 2010)

Appendix G

A Classification of BPEL Extensions

A Classification of BPEL Extensions

Oliver Kopp, Katharina Görlach, Dimka Karastoyanova, Frank Leymann, Michael Reiter, David Schumm, Mirko Sonntag, Steve Strauch, Tobias Unger, Matthias Wieland
Institute of Architecture of Application Systems, University of Stuttgart
{lastname}@iaas.uni-stuttgart.de

Rania Khalaf
IBM TJ Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA
rkhalaf@us.ibm.com

Abstract: *The Business Process Execution Language (BPEL) has emerged as de-facto standard for business processes implementation. This language is designed to be extensible for including additional valuable features in a standardized manner. There are a number of BPEL extensions available. They are, however, neither classified nor evaluated with respect to their compliance to the BPEL standard. This article fills this gap by providing a framework for classifying BPEL extensions, a classification of existing extensions, and a guideline for designing BPEL extensions.*

Key words: BPEL Extension, Classification of Extensions, Extension Guidelines

1. Introduction

Originally, the Business Process Execution Language (BPEL) has been designed for the implementation of business processes using Web service technology. The Web service technology is the de-facto standard used to implement a service-oriented architecture [77]. Nowadays, BPEL is used for implementing business processes in numerous different scenarios: for automating scientific simulations, for provisioning software as a service (SaaS) applications and as exchange format for business processes (i.e., BPEL as description language for business protocols). The requirements of the usage scenarios differ and the desired functionality is not always shipped out of the box, i.e., it is not supported using standard language constructs. For instance, sub-processes are a demand that the BPEL specification [59] and consequently standard-conform implementations do not cover. As a result, BPEL is frequently extended for supporting desired functionality that is not available in standard BPEL. Depending on the particular purpose, an extension may improve efficiency, increase flexibility, ensure better performance, or add more functionality. However, an extension also has disadvantages. Firstly, the whole toolset that is used for business process management (BPM) needs to support the extension. Common components of this toolset are applications for modeling, adapting, executing, monitoring, and analyzing the processes. Secondly, if business partners exchange (parts of) their processes, their toolsets need to understand and support the extensions as well.

In this paper, we provide a classification of existing BPEL extensions and provide guidelines to develop extensions. This might support a developer to search for existing extensions and to develop a new extension in case a new one is necessary. Consequently, the paper is structured as follows: Sect. 2 provides the technical background that describes the typical environment for BPEL processes as well as the associated components and technologies. Sect. 3 introduces a classification framework for extensions including standard-conformity, distinction between modeling and runtime extensions, as well as different purposes. Building on this, Sect. 4 presents requirements on extensions to be standard-conformant to BPEL. Sect. 5 presents approaches to realize a BPEL extension and the related BPEL environment. Sect. 6 introduces an extension development guideline that helps in the course of implementing an extension. The classification is applied to 62 existing BPEL extensions in Sect. 7. The paper finishes with a conclusion in Sect. 6.

2. Background

In the following we describe the environment that is common for using workflows (cf. Fig. 1). Workflows are the implementation of business processes [51]. The environment also applies to environments for BPEL processes.

The components in the upper part of the figure represent the modeling part of the environment. It consists of three components. The process modeling tool is used for the (typically graphical) specification of process models. The process analysis tool refers to static verification, deadlock analysis and other checks that can be performed at design time. It is often already integrated in the process modeling tool. Finally, the process repository serves as a means for efficient storage and retrieval of process models.

The components in the lower part of Fig. 1 represent the runtime environment. The central component for runtime is the process engine. At process deployment time, a process model is passed to this component, which compiles the process model into an internal format and offers the deployed process as a service to the outside. A so-called navigator, a subcomponent of the process engine, manages the status of process instances, traverses workflow graphs, triggers activity implementation execution, and takes care of directing incoming messages to the intended recipients, i.e., to particular process instances using correlation [8]. The process engine communicates with services via the enterprise service bus [12]. The ESB allows for abstracting from communication details, such as the used transport protocol and message format. Note that an ESB is an abstract concept which may be implemented using a specific component (which is generally referred to as ESB, too) or in other ways, such as embedded into the process engine (cf.[49]). The services represent the actual functions that are orchestrated in the workflow. The monitoring component registers, receives, and analyses execution events that are emitted by the process engine and the orchestrated services. For example, this component allows tracking the status of a particular instance of a process.

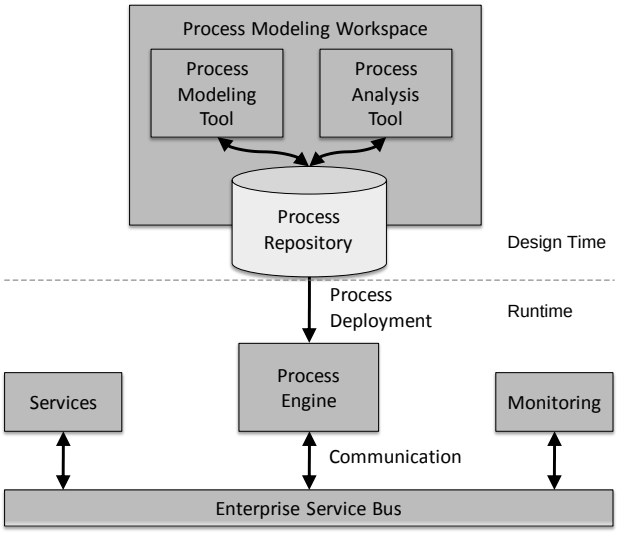


Fig. 1: Common Environment for Workflows

BPEL is a workflow language for specifying business process behavior based on Web Services [59]. It provides activities to exchange messages with Web Services and provides control-flow constructs to order these activities. BPEL requires the interfaces to be specified in WSDL 1.1 [15]. It is important that WSDL does not require the messages being exchanged using SOAP/http. Other bindings, such as SOAP over Java Messaging Service are available, too [2]. In BPEL, the connection to partner services is formed by a partner link, which specifies the port type required, offered, or both. An `invoke` activity is used to send a message to a specific operation of a Web Service. In its two-way form, it awaits a reply message back. A `receive` activity is used to receive a message by a given operation. A `pick` activity realizes a one-out-of-many choice of mutual exclusive incoming messages. A `wait` activity waits for a specified time or until a given date is reached. An `empty` activity does nothing. The `scope` activity enables fault-correcting behavior and event-handling. Faults are catched by fault handlers. A completed scope may be compensated. The compensation behavior is specified by a compensation handler. Event-handlers run in parallel to the activities in the scope and handle additional incoming messages and timeouts. The control-flow itself may be specified using block-structured and graph-based construct, which makes BPEL a hybrid workflow language [43]. The block-based constructs are `sequence`, `if`, `while`, `repeatUntil`, `forEach`, and `flow` without links. A `flow` with links enables modeling of a graph, where control-flow follows the specified links. A detailed summary is provided .by [51].

The first version of BPEL has been proposed in 2002 as “Business Process Execution Language for Web Services 1.0” (BPEL4WS). Subsequently, version 1.1 has been released in 2003. Here, minor

corrections and clarifications have been made. This version has been submitted to the Organization for the Advancement of Structured Information Standards (OASIS). In 2007, OASIS has completed the standardization process and has published the revised version as *WS-BPEL 2.0*. Important changes have been made with respect to the extensibility of the language. For example, designated concepts such as an `extensionActivity` element or `extensionAttributes` have been added (cf. Sect. 4). A detailed comparison of the BPEL versions 1.1 and 2.0 is provided by [66]. The work we present in the following focuses on the current language specification WS-BPEL 2.0 (BPEL), and the extensibility mechanisms specified therein. Where appropriate, we point out properties of BPEL 1.1.

In order to refer to the components affected by an extension, we present an exemplary architecture of a BPEL engine. We implemented a prototype of a BPEL engine (called Stuttgart's Workflow Machine, SWoM) at our institute[1]. The architecture of SWoM distinguishes all major components existing in a BPEL engine and thus can be used to illustrate them. The internal architecture of the SWoM is illustrated in Fig. 2. It consists of four main modules namely Gateway, Process Execution, Persistence, and Administration. The Gateway deals with Web Service invocations and handles incoming messages. The Process Execution is responsible for process instance creation and execution. The Persistence consists of databases for storing auditing events (*Audit Database*), data about deployed BPEL process models with appropriate WSDLs and deployment descriptors (*Buildtime Database*), and information about process instances (*Runtime Database*). The Administration contains an interface and functionality for human users to supervise process execution. The arrows in the figure indicate communication dependencies. Message queues and topics are used to decouple modules. Components with a black box at the top expose their functionality as Web service. After giving a short overview regarding the main modules in the following, we describe their inner structure.

The *Administration Interface* enables human access to core functionality of the engine. The *Import Export Handler* is used to import process models into the engine, to statically validate process models, and to delete and export uploaded process models. The *Process Deployment Manager* is responsible for deployment and undeployment of imported process models. With the help of the *Supervision* an administrator can activate or deactivate the auditing of process models. Furthermore, audited events of process models can be inspected. The *Systems Management* allows viewing and deleting errors occurred in the SWoM, forced termination of running process instances and their deletion from the SWoM as well as user management. The *Administration Infrastructure Provider* is an interface to access the databases and to put messages into the Manager topic (indicated by an "MT").

The *Service Provider* component exposes deployed process models as Web services. Web service clients can invoke processes by sending a SOAP message to the engine. In case of a synchronous request/response operation the Service Provider additionally sends the reply back to the client. The *Invocation Handler* is responsible for the invocation of Web services following the blocking request/response pattern or the unblocking one-way pattern.

The *Navigator* interprets process model logic, supervises control and data flow, and executes activity implementations. It makes use of the navigation queue (indicated by an "N") to send and receive navigation events. For each `invoke` activity it puts a Web service invocation message into the invocation queue (indicated by an "I") to be performed by the Invocation Handler. In case of a reply activity it inserts a reply message into the reply queue (indicated by an "R") to be sent back to the invoking client by the Service Provider. The *Data Manager* provides Runtime database access to the Navigator and caches process models to prevent from extensive Buildtime database accesses during process execution. Steering of *Data Managers* can be done over the Manager topic, e.g., to force process model state changes. The *Auditing* persistently stores information about the life of a process for analysis or legal reasons. The *Process Instance Creator* is used by the Navigator to build new process model instances in the Runtime database. The *Correlation Manager* correlates incoming and outgoing messages to corresponding process instances.

[1] *Institute of Architecture of Application Systems (IAAS), http://www.iaas.uni-stuttgart.de/institut/*

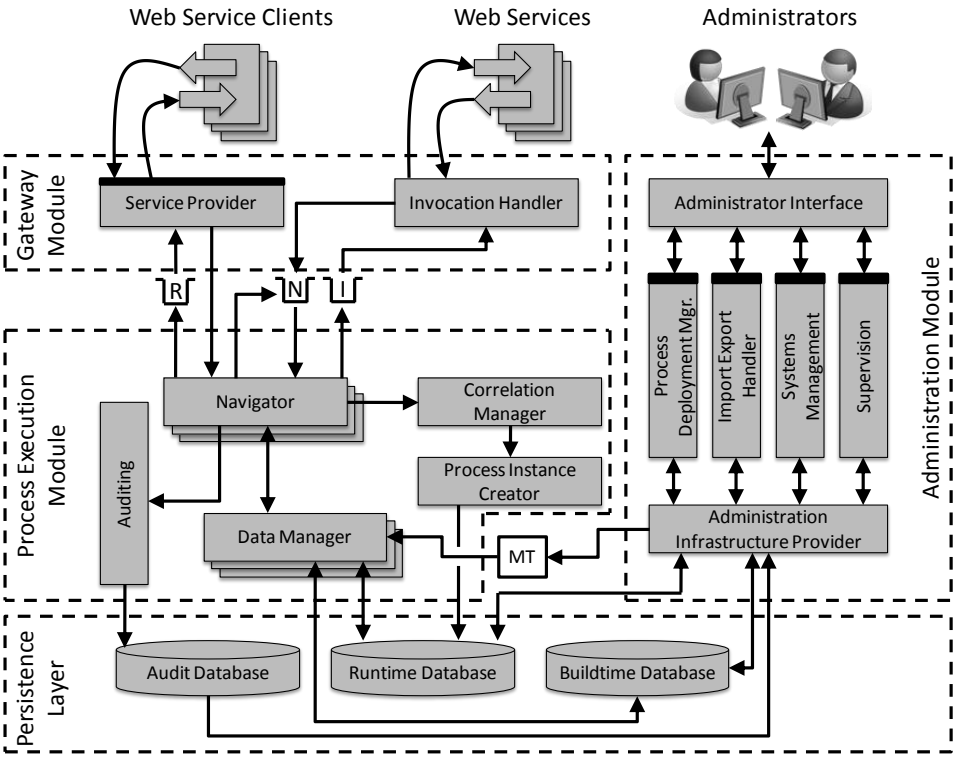


Fig. 2: Architecture of the BPEL Engine SWoM

Extending a process language has profound impact on all components of its supporting infrastructure, most important on the modeling tool and the process engine. Furthermore, the other components involved, such as tools for process analysis and monitoring, have to be adapted accordingly. Our evaluation of current approaches for extending BPEL in Sect. 7 shows that most extensions cover modeling tool and runtime extensions only.

3. Classification Framework

The follwoing definition defines the term “BPEL extension” and is referred to throughout the paper. The definition follows the definition of a software extension in the field of computer science [44].

> ***Definition 1:*** *A standard-conform BPEL extension is an enhancement of functionality of the Web Services Business Process Execution Language specified in the OASIS WSBPEL 2.0 standard by following the extension proceedings defined in the standard. On its own, the BPEL extension is not useful or functional.*

To be standard-conformant, extensions must not contradict the semantics of any element or attribute defined by the WS-BPEL specification. The concrete guidelines defined in the WS-BPEL 2.0 standard [59] are summarized in Sect. 4. The essence of these guidelines is presented in Tab. 1. In this table we provide a checklist for classifying a given extension with respect to its standard conformity. The table shows a characteristic, its standard conformity, and an identifier as a shortcut. The shortcut is used in Sect. 7 as reference for a classification. BPEL 1.1 does not explicitly define an extension mechanism, but allows for adding elements of other namespaces into the process model. BPEL 2.0 explicitly specifies the extension mechanism of BPEL. This has impact on the standard-conformity of an extension. As a consequence, we show the BPEL version in the column “Standard-conform Language Extension”. In case several characteristics are applicable to an extension, an extension has to be standard-conformant regarding all characteristics. Tab. 2 provides a classification into design time and runtime extensions. The runtime components listed in Tab. 2 are components illustrated in Fig. 2 which were extended by the extensions presented in Sect. 4. Note that an extension can be both a design time extension and a runtime extension. “n/a” denotes “not applicable”. This is the case if an extension is not a BPEL extension the sense of Definition 1. For instance, in case an extension changes the behavior of an invocation handler only, it is not an extension in the sense of Definition 1. For a standard-conform runtime extension at least the navigator has to be extended.

Tab. 1: Standard Conformity

Characteristic	Standard-conformant Language Extension	Shortcut
New activity without nesting in an extension activity	No (2.0) / Yes (1.1)	$\overline{s}$ 1
New construct/element in BPEL namespace	No (1.1/2.0)	$\overline{s}$ 2
New attribute in BPEL namespace	No (1.1/2.0)	$\overline{s}$ 3
Contradiction with BPEL semantics	No (1.1/2.0)	$\overline{s}$ 4
Defining something out of scope of the BPEL specification (not using <process> as root element, …)	No (1.1/2.0)	$\overline{s}$ 5
No extension declaration specified	No (2.0) / Yes (1.1)	$\overline{s}$ 6
New extension activity	Yes (2.0)	s7
New extension attribute	Yes (1.1/2.0)	s8
New extension construct/element	Yes (2.0)	s9
New extension assign operation	Yes (2.0)	s10

Tab. 2: Extension Type

Extension	Type	Shortcut
Modeling tool extension		
BPEL Extension can be transformed to standard BPEL	Modeling	M
BPEL Extension cannot be transformed to standard BPEL	Modeling	M
Modeling tool offers different rendering	n/a	
Process engine extension		
Deployment mechanism extension	n/a	
Invocation handler extension	n/a	
Correlation manager extension	n/a	
Navigator extension	Runtime	R
Auditing extension	n/a	

Extensions can be further characterized, independent of their standard-conformity and particular type. We use the extension purpose, the extension subject, the workflow dimension, and the placement in the business process management (BPM) life cycle as additional characterizations. The *extension purpose* criterion lists different intentions of an extension, such as the improvement of reusability of processes. The *extension subject* addresses the language constructs and mechanisms which are affected by an extension. According to [50] a workflow has three independent dimensions (IT infrastructure, process logic, and organization). We use these *workflow dimensions* as one criterion to characterize an extension. Finally, we use the *placement in the BPM life cycle* as criterion. The life cycle starts with modeling a business process. This business process has then to be refined to an executable process model (IT refinement). Static analysis and verification makes sure that the process model conforms to given constraints (e.g., freeness of deadlocks). Subsequently, the process model is deployed on a process engine, where the process is executed. In the monitoring phase the execution of single processes or process groups is observed. The results of monitoring are analyzed and may lead to redesign and optimization, which is again conducted in the modeling phase closing the loop.

These extension characteristics are listed in Tab. 3. We have derived the criteria and appropriate characteristics from the evaluated extensions (see Sect. 4). This list may be further extended when discussing novel extensions. The characteristics are sorted alphabetically, except the life cycle characteristics, which are sorted according to the order in the life cycle. “Occurrence” shows the total number of extensions matching the respective characteristic.

Tab. 3: Extension characteristics

Criterion	Characteristic	Shortcut	Occurrence
Purpose	Ability to outsource	C1.1	3
	Flexibility	C1.2	13
	Functionality	C1.3	28
	Maintainability	C1.4	13
	Performance	C1.5	6
	Reusability	C1.6	9
	Robustness	C1.7	11
	Usability	C1.8	12
Subject	Control flow	C2.1	25
	Data integration	C2.2	10
	Expressions/assign statements	C2.3	3
	Handling of large data	C2.4	2
	Other	C2.5	9
	Service binding	C2.6	5
	Service invocation	C2.7	22
	Variable access	C2.8	3
Workflow dimension	IT infrastructure	C3.1	29
	Process logic	C3.2	36
	Organization	C3.3	2
Placement in the BPM life cycle	Modeling	C4.1	47
	IT refinement	C4.2	2
	Static analysis/verification	C4.3	0
	Deployment	C4.4	10
	Execution	C4.5	45
	Monitoring	C4.6	3

Based on Definition 1, we can *exclude* particular changes on the BPEL language and give a list of approaches, which are *not* a BPEL extension. BPEL offers the possibility to model abstract processes, which need not to be executable but address different use cases. An abstract process profile specifies the semantics of an abstract process. It furthermore describes how to get an executable process starting from the abstract one, called “executable completion”. The BPEL specification itself provides two profiles: A profile for observable behavior and a profile for process templates. Abstract processes following the abstract process profile for observable behavior describe the public visible behavior of a process. Abstract processes following the template profile serve as process templates, where activities required for execution have to be put in at fixed places.[41] introduce the Abstract Process Profile for Globally Observable Behavior, which enhances the profile for observable behavior by providing more flexibility for the executable completion. Describing a new Abstract BPEL process profile is not an extension as it is just a restriction that defines, which constructs are allowed in a process model.

Approaches that redefine the semantics of existing BPEL constructs are not standard-conform and thus not an extension in the meaning of Definition 1. The specification does not provide information about the event model a process engine should support. Hence, a modification or extension of an existing event model, such as defined by Karastoyanova et al. [31], is out of scope of the specification and thus not a BPEL extension.

BPEL itself does not specify any rendering of the process model. Since the rendering is not standardized, any specific rendering is not a BPEL extension. This includes graphical renderings in BPMN [66];[76]) or a script syntax such as BPELscript [9].

4. Requirements for Standard-conform Extensions

In BPEL 2.0, the extensibility of BPEL is standardized. Extensions are declared in the `extensions` element. Each extension is associated with a namespace and takes a Boolean attribute

`mustUnderstand` [59], Sect. 14. In case the value is set to "`yes`", a process engine has to reject the process model if it does not support the extension. The specification does not state anything about the modeling tool. A value of "`no`" denotes that the extension is optional. In case an engine is not aware of the extension, the each respective `extensionActivity` is replaced by an `empty` activity, extension assignments are ignored, and all other XML attributes and XML elements are ignored.

The BPEL standard offers following possibilities to extend the language:

- Introduce new activity types, called `extensionActivity` ([59], Sect. 10.9
- Include new data manipulation operations ([59], Sect. 8.4)
- Specify individual query and expression languages ([59], Sect. 8.2)
- Allow namespace-qualified attributes and elements from other namespaces ([59], Sect. 5.3) and apply extension semantics for all BPEL constructs in the syntax sub-tree ([59], Chapter 14)

The standard requires that an extension does not cause any change to the semantics of a BPEL process ([59], Sect. 5.3). If an `extensionActivity` is a start activity or contains a start activity, the namespace of the `extensionActivity` child element must be declared as `mustUnderstand="yes"` ([59], Sect. 10.4). In the old version of BPEL, namely BPEL 1.1, an extension is simply made by adding XML attributes and XML elements in another namespace into the BPEL process. In case a workflow engine is not aware of the namespace, the behavior is not specified by the BPEL 1.1 specification. This version of the specification does not impose any restrictions on extensions. The fact that the execution semantics of the extension has to be described is implicitly required by all versions of the specification.

5. Possibilities to Realize an Extension

We distinguish between two different options for the realization of an extension in terms of Definition 1: (A1) Extended modeling tool and extended engine and (A2) extended modeling tool and model transformation.

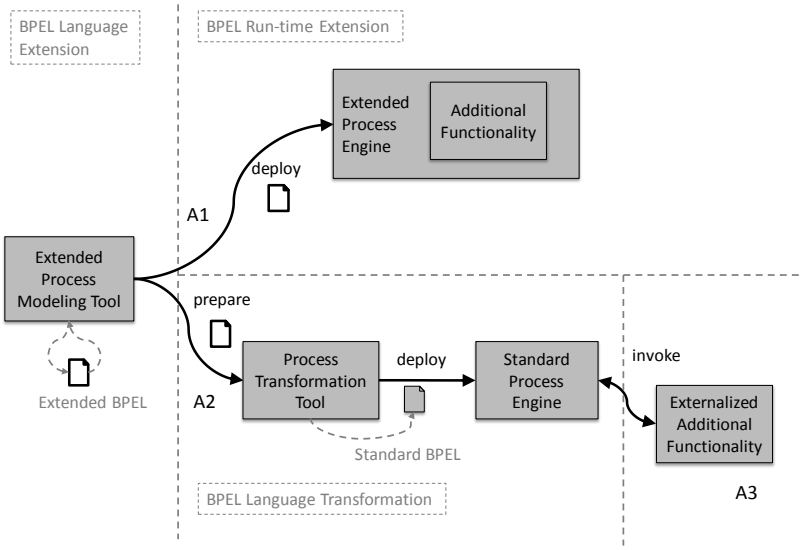


Fig. 3: Runtime Extension versus Model Transformation

The first option A1 "BPEL Runtime Extension" is represented by the upper branch in Fig. 3. Extended BPEL code is created in a modeling tool which supports this kind of extension. The BPEL code and its extension are deployed onto a process engine that supports the additional functionality. That means, the process engine has to be modified for this option.

The second option A2 "BPEL Language Transformation" is represented by the lower branch in Fig. 3. Extended BPEL code is created in an extended modeling tool as well. The significant difference to A1 is the employment of model transformations [69]. This technique translates the extension constructs into standard BPEL language constructs. Standard BPEL code is thereby generated that can be deployed on a process engine that is not aware of any extension. Note that this paper does not discuss transformations of other business process modeling languages to BPEL. A discussion of that aspect is given by [70].

In addition to these two options, there is the possibility to separate the desired functionality in an external service. In case an extension uses this approach, it is not a valid extension according to Definition 1. This option A3 “Dedicated Service” is shown in the lower right corner in Fig. 3. The external service can be invoked from the process engine with standard language constructs. That means, a language extension is not required per se, but typically provides more comfort. In this setting, the modeling tool may be extended to support different renderings of the dedicated services or may be kept as is.

The runtime extension approach (A1) envisages extending both the language (including the modeling tool) and the execution engine that supports the execution of the new constructs. This may also require an adaption of the monitoring components, as they may need to distinguish standard and extended activities and monitor them differently. The consequential changes may reach up to the dashboard. A prominent example for the runtime approach is the extension BPELJ [10], which extends BPEL with the possibility to use Java code snippets as an activity. The BPEL language is extended with an according `extensionActivity`, the modeling tool is extended for support of entering Java code, and also the process engine is extended for actually executing the Java code.

In the model transformation approach (A2) basically higher level constructs are introduced. This is, however, only possible if an extension is expressible with a set of standard constructs. For illustrating this approach we take a fictive BPEL extension, which we call “Delayed Execution”. Listing 1 shows the code for an `invoke` activity that uses the “Delayed Execution”, which delays the execution for 3 days and 10 hours counted from the point of the activation of the `invoke`.

```
<invoke name="refreshValue" ext:delay="P3DT10H" .../>
```

Listing 1: Invoke Activity Extended for Delayed Execution

A model transformation tool has to processes all constructs that carry an extension attribute for the delay. Each identified construct is split up into a `wait` activity and the actual activity (here: an `invoke` activity) that should be executed (cf. Listing 2).

```
<sequence …>
  <wait name="refreshValueDelay" for="P3DT10H" />
  <invoke name="refreshValue" …/>
</sequence>
```

Listing 2: Extended Invoke Activity Transformed to Standard Constructs

For some cases, functionality can be externalized as a service (A3). This approach is easy to implement, offers high reusability (even outside of BPEL processes), and does not hamper portability of the processes. A major issue is that the require functionality may need the current state of a process instance such as the state of activities and variable content, which is difficult to pass to the externalized service. This limits the applicability of this approach. The approach may, for instance, be applied for extending BPEL with business rules, discussed in Sect. 7.2.1.

When comparing the different extension options A1 and A2, the model transformation approach (A2) has one significant advantage: Compatibility and thus portability of the process models to another toolset is preserved. It also has a significant disadvantage: The original activity is replaced by a set of new activities, variable definitions, and other constructs that do not represent the work that was actually intended. This circumstance impacts monitoring and debugging instruments that will register the execution of activities that are not contained in the original process model. To ease monitoring, an additional transformation step of monitoring information into the former process model format is required. The transformation approach is, however, not applicable in all cases. If an extension cannot be expressed with standard constructs, an extension of the engine is inevitable. The advantage of the runtime extension approach is the holistic and consistent integration of the extension in the modeling tool and workflow engine. The user gets what he modeled. Moreover, this solution promises the highest engine performance due to an optimized workflow model (as no additional elements are generated) and a reduced communication overhead. The disadvantage of the runtime extension approach is that it requires a huge development effort. Note that the approaches can be combined: In case a process engine supports the extension, it can be executed natively. If it does not support the extension, then a model transformation step needs to take place in advance. The option to externalize

the new functionality into a distinct service that offers the functionality is only possible if the new functionality does not affect actual engine components, such as the navigator.

6. Extension Development Guideline

If new functionality is required for the development of business processes, one has to balance how and where to integrate this functionality. This section provides the reader a means at hand to decide whether a BPEL extension is an adequate solution. For supporting the decision making, we present in Sect. 6.1 different aspects that should be considered when planning BPEL extensions and give recommendations how to achieve the planned goal. Due to its high development effort, the runtime extension approach (A1) should be avoided if possible. If no reason is found for an A1 extension, the enhanced functionality should be implemented in other ways as described in Sect. 5. For instance, the enhanced functionality may be realized as Web service called by a workflow, as functionality in the ESB-infrastructure, as design time extension in the modeling tool, or as transformation. If it turns out that the A1 approach is needed, there are different possibilities how to implement it. Sect. 6.2 discusses three possibilities: As a commercial solution, as a self-implemented solution based on open source software, or as a hosted solution.

6.1 Recommendations for the Choice of Extensions

When deciding about the need for an extension, different aspects of the extension have to be thought of, which we present in the following. We discuss the aspects and give recommendations for design and implementation of extensions. Note that the considered aspects are arranged in an unordered list.

Implementation of the functionality in other components of the infrastructure. The infrastructure offers components such as an enterprise service bus or application server (cf. Sect. 2). It may be possible to implement the needed functionality in a component other than the engine. For example, retrying service invocation or replacing a service with an equivalent service is a typical task for an ESB [12]; [49]. Thus, this functionality is not implemented in the BPEL engine but in the integration layer. If the functionality can be realized by modifying infrastructure components other than the engine (e.g., the ESB), we recommend this approach. In case the planned extension needs to be reflected in the workflow logic, it should be implemented in the workflow engine.

Visibility of the extension in the workflow model required. A BPEL extension is visible in the workflow model if it is explicitly declared as `extension` and either embedded in an `extensionActivity/extensionAssignOperation` element or implemented as an extension attribute or extension element. This allows identifying usage of the extension easily. In case visibility of the extension is necessary for process users and/or developers, the extension should be designed according to the standard mechanism (cf. Sect. 5) using the A1 or A2 approach. If visibility can be neglected, we recommend the dedicated service solution (A3), which is easier to implement.

Visibility of the extension in the audit trail required. Typically, a BPEL engine logs state changes of activities in the audit trail. The planned extension may need to be accounted for in the audit trail. When realizing the extension with approach A2, the process model is transformed into a standard BPEL process model where the extension is not visible anymore. We recommend solving this problem with a two-directional mapping between the modeling tool extension and the representing standard BPEL elements. The mapping can be used to conciliate the displayed auditing information and the process model. It may happen that the backward mapping (transformed model to extended design time model) is complex or even ambiguous, e.g., in the case a service is used by multiple extensions. In this case we recommend realizing the extension in the workflow engine (A1).

Detailed internal execution information of the extension in the audit trail required. It may be required to add information beyond standard state changes of an extension activity to the audit trail, which may be the progress of execution or the selected user for instance. In this case an engine extension is inevitable (A1). Furthermore, the audit trail has to be capable of storing this additional information and may also need to be extended.

Execution performance importance. If the runtime of the extended functionality is a major issue, we recommend implementing the extension directly in the workflow engine (A1). This solution is characterized by the possibility of optimized code (compared to A2) and by a reduced communication overhead (compared to A3).

Based on the decision taken at each aspect, it can be decided whether a BPEL extension is needed. The decision depends on the concrete problem statement. Thus, a general answer cannot be given and has to be made on a per-case basis. The different alternatives, their advantages and

shortcomings are presented in Sect. 5. In case the decision is to create a BPEL extension, the next step is to decide how to realize the extension.

6.2 Solution Possibilities for Implementing a BPEL Extension in a BPEL Engine

After deciding for realizing an extension in the modeling tool and the engine (approach A1 from Sect.5, the extension has to be implemented both in the modeling tool and in the engine. In case the approaches A2 or A3, the modeling tool has to be changed to support the extension. The BPEL engine stays unchanged.

In this section, we describe how extensions can be added to existing systems by providing concrete examples. The discussion is structured around several key considerations: The level of extension support in the system and the ability to modify the system itself. Subsequently, we address the additional issues arising from implementing extensions in hosted BPM systems, e.g., "BPM as a service", which is an emerging trend.

The first consideration is whether the system (modeling tool and engine) has some or full support for extensibility. In the case that it does have support, the developer simply uses the extension support – provided that it can handle the requirements of the target extension. Examples for this case are the Eclipse BPEL designer[2] and the Apache ODE engine, where plug points for extensions are available. The "Pluggable Framework for Enabling the Execution of Extended BPEL Behavior" (described in Sect. 7.3.2) also allows for changing the behavior of BPEL and thus offers an alternative way to extend BPEL engines.

In the case that the system does not have adequate support, one must first enable it. This can only be done if the source code is accessible and can be modified, which is the case with the Eclipse BPEL designer and the Apache ODE engine, for instance. Consider a developer having an extension that introduces data references in BPEL during runtime execution [79]. The Eclipse BPEL Designer nor the Apache ODE engine supports this out of the box. Thus, the support has to be added to them by a programmer.

In all cases, commercial products are always a solution. Thus, the first decision to make is a make-or-buy decision [27].

Commercial Solution: With most commercial workflow systems it is not possible to implement BPEL extensions, because their source code is not available and they do not provide an extension interface. Thus, only the usage of extensions provided by the vendor is possible. Nevertheless, a custom development of an extension by the vendor may be triggered.

Open Source Solution: The alternative is to implement a BPEL extension using an open source workflow engine. Compared to the hosted solution, this approach has the advantage that the developer has the full control over the development of the extension. Extensions are not restricted to defined extension points. If the system is running on a private server, execution of the extension can be observed and the data that is used in the workflow is secure (as long as critical data is not sent to external services). There are open source workflow engines available that can be used as development basis. As described in Sections 2 and 5, a modeling tool is also an essential part of the system and therefore has to be extended accordingly.

One of the goals of the standardization of BPEL has been the removal of all dependencies between process definition files, their process modeling tool, and the engines running those workflows. The modeling tool and the engine can be regarded as loosely coupled as they are replaceable by other systems that are implementing the BPEL 2.0 standard. This interchangeability breaks when a new extension activity is introduced. An extension activity typically enhances the set of BPEL activities and adds dependencies between the process engine and process modeling tool, as both have to understand how to handle these extension activities. Both systems (engine and modeling tool) have to care about the syntax of the extensions and the developer has to ensure that both systems rely on the same version of the extension activity. The engine needs to know what to do when it reaches the extension activity within a workflow model (semantics) and the modeling tool needs to know how to visualize, serialize and deserialize the activity to and from XML. Thus, both systems are not loosely coupled anymore. When creating a new extension activity, the workflow engine has to be extended via its extension API (if available). In addition, the modeling tool with its (mostly different) extension API has to be extended independently which leads to two extension implementations: one for the engine and one for the modeling tool. The developers have to take care that both versions do not differ in syntax and semantics.

[2] *http://www.eclipse.org/bpel/*

For avoiding double implementation we developed a system design that allows using the same data model for the Eclipse BPEL Designer and Apache ODE [20]. This approach allows for implementing an extension by using a single shared Java class. The modeling tool and the engine use the corresponding parts of this class relevant for them: the modeling tool uses the layouting and XML serialization parts; the engine uses the execution code and the serialization code. This has the advantage that less inconsistencies, e.g., in the serialization or naming of the developed extensions, occur.

Hosted Solution: There is a current trend towards hosted "BPM as a service" systems, which are "Software as a Service" solutions targeting Business Process Management. As such, they provide a hosted system (accessible simply with a Web browser) for the end-to-end BPM lifecycle including design, execution, and monitoring. Additionally, such systems can enable collaboration between developers and designers. With nothing to install, this lowers the barrier to entry but does require a continuous connection to the Internet while working. In such systems, additional concerns arise for providing extensions. Referring back to the previous concerns, a developer has no access to modify the source and thus one must rely on supported extensibility. Thus, we focus on a concrete BPM as a service system presented by Curbera et al.[16]. It consists of a visual modeling tool backed by the Bite workflow runtime [37] and an extension catalog [68]. This system supports extensions and also enables collaboration around extension activities: Developers and designers can use the catalog to download, use, comment on, and rate extensions. The extension considerations highlighted in this section are the same for Bite and BPEL, because Bite's control flow semantics are a subset of BPEL's.

First, consider how the Bite runtime identifies and executes an extension activity: An extension is recognized upon encountering an unknown XML element in the process. The engine looks up a corresponding extension implementation module in an extension registry and associates it with the parsed activity. An extension implementation module may be written either in Java or in any of a set of supported scripting languages. When the extension activity is reached and activated in a process instance, the implementation module is called and handed the XML definition of the activity and required process instance data. The extension activity may only write data to the activity's output variable. It does not have the ability to read or modify process navigational state. Once the implementation module completes, its output is stored in the output variable of the extension activity and the activity completes.

The extension enablement considerations for a hosted system include ensuring that the implementation artifact can reach the runtime, be registered in a catalog for use and looked up by the runtime and by other designers, and be able to be rendered by the design tool. In the system, a developer wanting to create an extension must provide basic activity metadata along with code implementing the extension. The meta-data is used by (a) the modeling tool, in order to provide the user with a meaningful display of the desired inputs for the extension and (b) the catalog, in order to provide a description and tags for users browsing the catalog. The implementation module is placed in a shared repository.

Developers upload the extensions either via a plug-in to their development environment or via a simple Web form. The extensions become immediately available to logged-in users. Once a user selects to use an extension activity in a workflow, its implementation module is pulled from the repository, the extension is registered with the engine and the module is bundled with the workflow application.

One key concern around extensions in a BPM as a service system is that it requires strong policing of the quality and integrity of the extension implementation code due to the fact that the environment is shared among many users and that the hosting entity may be liable for malicious extension code and potentially missed Service Level Agreements. This concern may be addressed by applying trust and reputation systems such as rating and ranking, third-party certification, and the ability to upload only by those with explicitly granted privileges.

7. Extensions in Practice

This section lists 62 commercially available extensions and scientifically published extensions. We apply the classification provided in Sect. 3. An extension may cover only the design time, the design time and the runtime, or only the runtime environment. The following is structured accordingly and additionally subdivided into vendor and research extensions.

7.1 Design Time only Extensions

This section presents approaches that make use of the transformation approach (A2) or that invoke dedicated services in order to integrate additional features (A3). It is also possible to combine both ways, as shown by Oracle's extensions presented in the following section.

7.1.1. Design Time only Extensions by Vendors

Oracle's Human Task [61] activity is used to integrate human behavior into business processes (C1.8, C2.7). There are several configuration options, for example to route a task to a second approver or to execute a number of human tasks in parallel. Tasks can be assigned to humans by specifying concrete users or user groups. Depending on the chosen configuration human task activities are realized by `scope`, `assign`, `invoke`, `receive`, and `switch` activities. Oracle's Process Manager provides a dedicated human task Web service that is called by the `invoke` activity (C3.1, A3). A GUI enables the assigned user to handle the task (C4.1). The outcome of the task is sent back to the process. The extension mechanism used is an extension element that annotates the activities realizing the human task (s9).

Oracle's notification service [61] is a collective term for five different notification mechanisms, namely email, fax, pager, SMS, and voice messages (C1.8, C2.7). Each is reflected by a single activity on a component palette in the process modeling tool (C4.1). Configuration of the activities is type-dependent. For example, the email activity provides parameters for target email addresses, a subject, and email body. The code underlying a notification activity is BPEL compliant: the activity is transformed into a `scope` with input, output, and fault variables, an `assign` to copy the user's parameter values to the input variable, an `invoke` activity to call a dedicated notification service, and a fault handler to deal with possibly occurring failures. An extension element annotates the `scope` to mark it as notification activity (s9). The appropriate notification service is provided by Oracle's Process Manager (C3.1, A3) that routes notifications to particular servers (email server, SMS server, etc.).

7.1.2. Design Time only Extensions by Research

BPEL4Chor is extending BPEL with a unique ID which is used for identifying message activities and `onMessage` branches. Decker et al. [17] present *BPEL4Chor* as an extension of BPEL for modeling choreographies. A choreography describes the message exchange between multiple participants ([64]; C1.1, C4.1). BPEL4Chor uses BPEL to describe the behavior of each participant in a choreography (C2.1, C3.2). The BPEL4Chor topology lists the participants and the connection between them in the form of message links. A unique ID is used to identify the activities and `onMessage` branches, which are referenced in a message link. The ID is stored in the attribute `wsu:id` (s8). The `name` attribute is not used since it is not possible to put a `name` attribute on an `onMessage` construct. Each participant behavior description is transformed to an abstract BPEL process following the abstract process profile for observable behavior. This model does not contain any extensions any more. The model is then manually refined to an executable BPEL process without addition of any BPEL4Chor related extensions (C4.2). This makes BPEL4Chor a design time only extension.

The *ID attribute* is a general extension where a unique identifier may be put to each element in the BPEL processes (s8). The identifier is mainly used in modeling, such as for referencing particular constructs (C1.4, C2.5, C3.2, C4.1). The modeling extension does not need to be understood by the engine (`mustUnderstand="no"`) since there is no runtime behavior of the identifiers.

BPEL process templates [30] are abstract, reusable units of BPEL code stored in a separate XML file (`*.template`). Usually, a template solves a general, recurring problem that can be used to avoid process modeling from scratch and reinventing the wheel (C1.2, C1.6). Templates are abstracted with the help of parameters that hide certain details (e.g., variables, partner links, port types). At buildtime, parameters can be mapped on concrete values provided by the process modeler. Templates can be referenced from within BPEL processes by a `tRef` element in BPEL namespace (C2.5, $\overline{s}$ 2). Using such references in templates allows recursive template definition (C3.2, C4.1). Since processes pointing to templates are not executable, transformation steps need to be performed in order to make them executable. Template parameters are thereby substituted by concrete values, template references by the actual template code (A2, C4.4).

"SWRL for BPEL" [81] defines how constraints between BPEL activities can be encoded in the BPEL process using the Semantic Web Rule Language (SWRL). This enables another way of modeling process models (C1.4, C2.1, C3.2, C4.1). The extended BPEL process is transformed to a standard-

conform process following the given constraints (A2). The extension declares new extensions elements (s9).

BPEL fragments [53] are introduced as modeling construct to enable reuse of process parts across different processes (C1.6, C2.1, C3.2, C4.1). The approach does not use BPEL's extension mechanisms, but declares a new namespace and uses `fragment` instead of `process` as root element ($\overline{s}$ 5).

BPEL-D [36] replaces variables by explicit data links in BPEL 1.1 (C2.8, C3.2, C4.1). In general, there are two ways to propagate data between activities in business processes: the blackboard approach and explicit data flow [5], p. 266. In the case of the blackboard approach, variables are used to share data. BPEL implements the blackboard approach, whereas BPEL-D realizes explicit data flow. Thus, BPEL-D contradicts with the BPEL semantics ($\overline{s}$ 4). The motivation of BPEL-D is enabling business process outsourcing (C1.1): A BPEL-D process is used as input for an algorithm splitting the process into several standards-conform BPEL processes, which maintain the operational semantics of the intended BPEL-D process [34]. Thus, BPEL-D is only used at design time. It is possible to transform one BPEL-D process into one standard BPEL process reassembling BPEL-D semantics by standard BPEL constructs.

BPEL data transitions (BPEL-DT) extend the BPEL language with data transitions for handling large amounts of data [21]. This is, for instance, required in ETL (extract, transform, load data) flows that are based on Web service orchestrations which are realized with BPEL. Such data intensive service applications can make only limited use of the "by value" semantics in BPEL, as otherwise massive data sets have to be transferred forth and back to the process engine. Other ways of specifying data flow are therefore necessary. In standard BPEL, data flow is implicitly contained by the access of activities to variables and their values, respectively. BPEL-DT seeks to make data flow explicit by extending the BPEL metamodel with data transitions (i.e.,data links; C1.3, C1.5, C1.8, C4.1). These links are transformed into an XML mapping specification (A2; MSL, [24]), which needs to be manually refined (C4.2). The engine then calls additional services to realize the given mapping specification (A3, C3.1, C4.5). This extension is not implemented in a standard-conform manner and contradicts the BPEL semantics, since a new kind of links is added ($\overline{s}$ 4). In BPEL-D, data-flow is still internal to the process, whereas BPEL-DT externalizes the data flow.

References in BPEL [79] also address handling large amounts of data by extending BPEL's data handling mechanism with pointers on data (C1.8, C2.2, C2.4). A BPEL `referenceVariable` element in BPEL namespace ($\overline{s}$ 2) is introduced that specifies variables containing a reference to externally stored data (C3.2). The attribute `referenceType` indicates whether a reference is resolved at `scope` activation, before each usage, periodically, or on behalf of an external partner (C2.8). Actual reference resolution is made by an external Reference Resolution Service (RRS) (C3.1, A3). Since "References in BPEL" is proposed as build time extension, a pre-deployment step needs to transform extended BPEL files into standard BPEL by replacing reference variables with BPEL variables, inserting partner links and interaction activities (depending on the reference type) (C4.1, C4.4, A2).

"*Activity failure and recovery*" is a BPEL extension proposed by Liu [52] which is intended to increase the reliability of processes and to relieve process modelers from the complexity of defining BPEL fault handlers. They therefore introduce four fault tolerance patterns (ignore fault, skip scope, retry scope, and alternative scope) that can be exploited during modeling of processes to express reactions on faults (C4.1). The specified patterns are not included in the designed process but are mapped on `scope`s by name. Each pattern consists of rules to transform a given process definition into a process that implements the particular fault tolerance mechanism (e.g., retry a scope a specified number of times) (C1.7, C2.1, C3.2).

"Activity failure and recovery" is also proposed by Modafferi and Conforti. [54]. Here, an annotated BPEL process is used as starting point. The annotations include setting variables by external messages (C1.2, C2.2, C3.2), specifying timeouts for service invocations (C1.7, C2.7, C3.2) and enabling redoing of an activity (C1.2, C1.3, C3.2). The annotated process is then transformed to a standard BPEL process (C4.1, A2). The extensions are put in the BPEL namespace ($\overline{s}$ 2).

xBPEL [11] is a BPEL extension for modeling mobile participants in workflows (C1.2, C1.3). Chakraborty et al. introduce the PerCollab system which executes xBPEL and allows mobile integration of people into BPEL workflows without constraining the users to their desktop PC. xBPEL allows modeling communication between people and between a process and people (C2.1, C3.3, C4.1, C4.5). The extensions are put into the BPEL namespace ($\overline{s}$ 2). An xBPEL process is transformed to standard BPEL process (A2) and services of the PerCollab environment (A3).

7.2 Design Time and Runtime Extensions

This section lists extensions, where the BPEL modeling tool and the BPEL runtime are extended.

7.2.1. Design Time and Runtime Extensions by Vendors

WS-BPEL Extension for People (*BPEL4People*) enables integration of human-based activities in BPEL [3]. This includes the possibility to define people's activities, people groups, tasks and notifications (C1.3, C2.1, C3.1, C3.2, C3.3, C4.1, C4.5). BPEL4People is building on WS-HumanTask (cf. Agrawal, 2007b). WS-HumanTask is used in BPEL4People for the actual implementation of a people activity. BPEL4People defines the `peopleActivity` as a basic activity type which uses human tasks as an implementation (C2.7, s7). The `peopleActivity` allows specifying tasks local to a process or use tasks defined outside of the process definition. To use BPEL4People the modelling tool and the process engine must be extended (A1, A3).

BPEL for Java (*BPELJ*) combines the programming languages BPEL and Java [10]. The intention is to provide a way for integrating pieces of Java code into a BPEL process definition. The main effect of this extension is a higher convenience when programming a BPEL process (C1.3, C1.5, C1.8). BPELJ allows using Java code to be included in BPEL process definitions. The according activity in BPELJ is named `snippet`. In a snippet, BPEL variables can be manipulated and those snippets can be used for instance in loop conditions, branching conditions (C2.1) and for variable initialization as well as variable manipulation (C2.2, C2.3, C2.8). To use BPELJ extended modeling tools and process engines must be implemented (A1). Since BPELJ allows the modification of variables in a transition condition, it is not conform to the BPEL execution semantics ($\overline{s}$ 4).

BPEL-SPE [39] is a BPEL 2.0 extension for sub-processes that aims at increasing legibility and reusability of processes (C1.4, C1.6, C1.8). Sub-processes are BPEL processes implementing a single request-response operation and are called using a `call` activity in BPEL namespace from within the parent process (C2.5, C2.7, C4.1, $\overline{s}$ 2). The life cycle of sub-processes is tied to the respective parent process (C1.3, C3.2). For instance, a fault in a sub-process needs to be propagated to the parent process. This is enforced by coordination messages employed BPEL engines need to understand (A1, C4.4, C4.5, C4.6). Sub-processes can be defined as standalone process (C1.1) and inline within a parent process (C3.2). An inline sub-process can access visible data (i.e.,data of the scope it is defined in) of its parent process and thus omit implementation details.

The *Execution as Subprocess* extension [25] is a variant of BPEL-SPE. The goal is to enable an execution as a subprocess in a declarative way instead of a `call` activity (C1.3, C2.1, C2.7, C3.1, C3.2, C4.1, C4.5). The partner link declaration is extended by the attribute `processTemplate` (s8). Here, the name of a BPEL process may be specified. If the execution engine finds that process at the runtime, the process is directly called by the BPEL engine and the life cycle of the process is tied to the caller (A1). That means, for example, that a fault on process level of the called process is communicated to the calling process.

The *Collaborative Scopes* approach [25] adds support for case handling [1] to BPEL processes (C1.3, C2.1, C3.1, C3.2, C4.5). A new `collaborativeScope` activity is introduced (C4.1, s7). Each activity in a collaborative scope may have an exit condition. It is possible to evaluate the exit condition on start or on completion of an activity, or both. In case the condition is evaluated at the start, the activity is skipped if the exit condition is met. In case the exit condition is evaluated at the completion of an activity and the exit condition evaluates to false, the activity is started again. The extension is included in the modeling tool and realized in the engine (A1).

The *Generalized Flo [25]* enables control links to connect activities arbitrarily (C1.3, C1.8, C2.1, C3.2, C4.1, C4.5, $\overline{s}$ 4). Standard BPEL allows links to form an acyclic graph only. In addition to arbitrary connections, fault links between two activities are introduced. If the source activity faults, the target activity is executed. The generalized flow has to consist of one start activity only and only one control link may be followed at each execution step. The approach requires an engine extension (A1).

ii4BPEL [23] integrates SQL statements into BPEL, connects processes directly to relational databases, and supports advanced ways of data exchange (C1.3, C1.5, C2.2). IBM implements ii4BPEL in the WebSphere Integration Developer as a Plugin. Based on BPEL 2.0 IBM extended the BPEL language and the tooling, e.g., the process engine, the deployment mechanism, the modeling tool (A1, A2, C3.1, C3.2, C4.1, C4.4, C4.5). Furthermore, a special data middleware is required (A3). ii4BPEL defines four new activities for data management (s7): `SQLSnippet` runs an SQL statement against database tables. `retrieveSet` load referenced data sets into BPEL-variables.

`atomicSQLSequence` join SQL snippets and retrieve sets in one activity. `informationServer` interacts with the IBM InfoSphere Information Server.

Non-compensatable scopes [25] introduces the attribute `compensatable` to a scope. In case the attribute is set to `yes`, a compensation of the scope leads to a fault (A1, $\overline{S}$ 4). The feature is used to improve performance of process execution: In case a scope is marked as non-compensateble, no snapshots of variables after the completion of the scope are needed (A1, C1.3, C2.5, C3.1, C4.1, C4.5).

Dedicated Administrator [25] enables the assignment of an administrator to a scope at the beginning of its life cycle. The administrator may do corrective changes to variables and has full control over the life cycle of the scope to ensure proper process execution (A1, C1.7, C2.5, C3.1, C4.1, C4.5, s8).

A *microflow* [25] is a new execution mode for business processes indicated by `wpc:executionMode="microflow"`. A microflow is a micro script which is executed in one transaction to speed up processing ([50] C1.5, C2.5). Due to the single transaction, the starting receive is the only receive allowed. Asynchronous invokes are always allowed, whereas synchronous invokes only in the case of synchronous bindings (C3.1, C4.1, C4.5).

Transaction boundaries[25] enable configuration of the internal behavior of the BPEL engine with respect to its internal atomic transactions (A1, $\overline{S}$ 4). The navigator of a BPEL engine usually starts a new transaction at the beginning of an activity and commits it at the end of the activity. This execution causes an overhead at the transaction manager. By configuring the transactions to span multiple activities, this overhead and hence the process execution time can be reduced (C1.5, C2.5, C3.1, C4.1, C4.5).

The Apache ODE group [6] proposes eight extensions to facilitate execution of BPEL processes. The specification of these extensions does not require declaration of the extensions. Besides adding new activities and attributes, the Apache ODE engine[3] offers support for XPath 2.0 as query language and adds *new XPath functions* reducing the coding effort. For instance, the function `insert-before` inserts a node as a sibling before a given node (C1.3, C2.5, C3.2, C4.1, C4.5).

Implicit correlations remove the need to add correlation sets in the case the BPEL process starts the interaction with a service [6]. By using implicit correlation, a unique session identifier is generated and put into the message (C1.3, C2.7, C3.1, C4.1, C4.5, $\overline{S}$ 4). The response of the service contains the same session identifier. The message router of the engine uses this identifier to route the message to the correct process instance. A concrete implementation is available for the SOAP/HTTP binding (A1).

Activity failure and recovery enables configuration of failure handling in the case of an invoke activity[6]. An example for a failure is an HTTP timeout. Default failure handling shows faults in the process instance management of Apache ODE and requires manual intervention. This behavior can be changed by a `failureHandling` element ($\overline{S}$ 4). It can be configured as follows: `retryFor` specifies the number of retries; `retryDelay` denotes the time between each retry; `faultOnFailure` causes the invoke activity to throw an `activityFailure` fault as BPEL standard fault in the case of a failure (A1, C1.7, C2.7, C3.1, C4.1, C4.5).

Headers handling enables the access to header fields in SOAP messages ([6] A1). For that purpose, the attribute `header` is introduced into the BPEL namespace at the `from` and `to` elements of a `copy` statement in an assign activity ($\overline{S}$ 3). In case the attribute is present, the context node of the XPath statement is set to the specified header element (C1.8, C2.1, C3.2, C4.1, C4.5). There is no explicit possibility to check for presence or absence of header fields.

The *iterable forEach* adds the element `sequenceValue` to the BPEL namespace below a BPEL `forEach` ([6]; A1, $\overline{S}$ 3). If the element is present, the `forEach` iterates on all elements contained in the given `xsd:sequence` element instead of using start and final counter value (C1.3, C2.1, C3.2, C4.1, C4.5, $\overline{S}$ 4).

The *auto complete copy destination* enables the attribute `insertMissingToData` in a `to` statement `copy` statement in an `assign` activity ([6]; $\overline{S}$ 3). If set to `yes`, the path to the element given in `to` element of a `copy` statement is automatically generated (C1.3, C1.4, C2.1, C3.2, C4.1, C4.5). For example, if `New York` is assigned to `$customer/address/city`, but the variable `$customer` is empty, the parent elements `address` and `city` are automatically generated.

[3] *http://ode.apache.org/*

To enable ignoring unavailable data the two attributes ignoreMissingFromData and ignoreUninitializedFromVariable are introduced to the copy statement of the assign activity ([6]; § 3). In the case of `ignoreMissingFromData` and a from-spec returning no XML information items, the `selectionFailure` fault is suppressed and no assignment done. In case of `ignoreUninitializedFromVariable` and the usage of an uninitialized variable in the from-spec, the `uninitializedVariable` fault is suppressed and no assignment is done (A1, C1.3, C1.4, C2.1, C3.2, C4.1, C4.5).

Process contexts are key value pairs allowing metadata in sent and received messages to be stored and accessed in processes (C1.3, C1.4, C2.1, C2.2, C3.2, C4.1). The contexts can be used in assign activities and in invoke activities (A1, § 4). Developers have to provide Java code to copy SOAP header information from and to context objects in Apache ODE. The Java code compiled and stored in the engine. The functionality is activated using `properties`-files and `deploy.xml`.

Resource-oriented BPEL is an approach to add support for providing and using REST services in BPEL. The Apache ODE group and Overdick[62] propose to add special REST attributes to the `invoke` activity, the `receive` activity and the event handler (C1.3, C1.4, C1.8, C2.1). That way, RESTful services are directly supported by BPEL instead of using a special HTTP binding in WSDL.

BPEL for REST is an approach shown in Pautasso[63]. Four activities (`get`, `put`, `post`, and `delete`) are used to invoke REST services (§ 2, § 4). RESTful resources can be offered via `onGet`, `onPut`, `onPost`, and `onDelete` handler (A1, C1.3, C2.1, C2.6, C3.1, C4.1, C4.5).

Continue on error [25] offers a similar behavior as activity failure and recovery. Each invoke activity gets the attribute `continueOnError` (s8). A human task for an administrator is generated in the case the invoke activity encounters a communication failure and the value of the attribute is `yes`. The assigned administrator is then privileged to do corrective actions. In the case of a `no`, the failure is converted into a fault and thrown into the BPEL process (C1.7, C2.7, C3.1, C4.1, C4.5).

7.2.2. Design Time and Runtime Extensions by Research

Retry scopes [19] extend BPEL with scope retrying behavior (C1.3, C1.7, C2.1, C3.2, C4.1, C4.5). The idea is similar to the idea presented by[52]. In [19], the issue of retrying is solved with an explicit `restart` activity and without an à priori rewriting step (A1, § 4). The `restart` activity may only be used in a fault handler and restarts the respective scope. By using an explicit activity, explicit repair behavior may be executed before restarting the scope.

BPEL/SQL [74] is a generic term for approaches to integrate SQL statements into BPEL with the aim to connect workflow engines directly to relational databases. [75] have presented an overview of BPEL/SQL implementations, which all share the properties of ii4BPEL described in Sect. 4.2.1: A1, A2, A3, C1.3, C1.5, C2.2, C3.1, C3.2, C4.1, C4.4, C4.5, s7.

Parameterized processes [30] is an extension that decouples BPEL's interaction activities from concrete port types and operations to improve reusability of (parts of) workflows (C1.6) and flexibility of selecting arbitrary services at runtime (C1.2). The new element `evaluate` is inserted under BPEL namespace into message sending activities to override the specified port type/operation pairs (C3.2, C4.1, § 2). The "evaluate" concept enables several strategies to provide an activity with a concrete port type/operation (static, prompt the user, query, and from variable) (C4.5, C4.6). The approach allows determining the interface of the service to invoke at runtime, taking different interfaces for different process instances, or handling faulty Web service invocations by default port type/operation pairs (C1.7, C2.7). In conjunction with the "evaluate" extension the `find_bind` element is introduced (in BPEL namespace) which can be used in message sending activities (C3.2, C4.1 § 2). It enables a deployment-independent specification of service selection policies even at runtime (C4.4), the runtime modification of such policies even for single process instances (C132) as well as a process instance repair if the service selection fails (C2.6). The parameterized processes approach extends both design time and runtime environments (A1).

Cross-process fault handling and transaction handling [43] enables grouping arbitrary activities of different participating processes together to form a logical transaction unit called choreography sphere (C1.3, C1.7, C2.1, C4.1). The grouping and additional handlers are specified outside the BPEL processes in the choreography. To execute the choreography sphere, an additional coordination infrastructure is needed (C3.1). Thus, the runtime semantics of BPEL is changed (C3.2, C4.5, § 4).

The *E4X extension for BPEL* [48] enables the usage of ECMAScript for XML [26] instead of XSLT and XPath in the case of variable manipulation. E4X extends JavaScript with support for XML-based data

manipulation (C1.4, C2.3, C3.2, C4.1, C4.5). The extension defines an `extensionAssignOperation` and an `extensionActivity`, where JavaScript code may be used (s7, s10).

Context4BPEL [80] allows the definition of context-aware workflows (C1.3). Such workflows may be used to create context-aware applications or to apply workflow technology in manufacturing production processes, for example (C4.6). Context4BPEL provides several extensions in a `c4b` namespace to implement three concepts for explicitly making use of context information from within workflows. First, the workflow can handle context events by particular activities that register (`c4b:registerSpatialEvent`), deregister (`c4b:deregisterSpatialEvent`) and update (`c4b:updateSpatialEvent`) events (C2.1, C2.7, C3.2, s9). Context events can be received by any incoming message activity with certain message types. Second, context data can be queried by a `c4b:queryContext` activity that stores the result of the request in a variable with well-defined type. Third, transition conditions can be evaluated based on workflow internal or external context data (C2.2, C2.3). New XPath functions are specified that facilitate dealing with context information, e.g., the `c4b:within(area, location)` function. Context4BPEL extends both design and runtime environment (C4.1, C4.5, A1).

BPEL4Grid [18] combines workflow and grid technology. The extensions help to invoke stateful Grid services (C1.3, C1.4, C2.2, C2.7, C3.1, C3.2, C4.1, C4.5). BPEL4Grid defines three new activities: `GridInvoke`, `GridCreateResourceInvoke`, `GridDestroyResourceInvoke`. Since BPEL4Grid introduces an additional way to communicate with services, it is not standards compliant ($\overline{s}$ 2). BPEL4Grid includes an extended modeling tool and an extended process engine (A1). A similar approach is presented by Zhang [82] where a `GrsService` activity is used to call a stateful Grid service.

BPEL^{light} [57] is an extension of BPEL 2.0 that decouples process logic from WSDL 1.1 interface definitions to improve reusability of process models and to enable workflow modeling without WSDL knowledge (C1.2, C1.6, C1.8, C4.1). BPEL^{light} introduces a novel interaction model with the help of BPEL's extension activity mechanism (C1.3, C3.2, C4.5): The WSDL-less `bl:interactionActivity` emulates the behavior of `receive`, `reply`, and `invoke` activities (C2.7). WSDL-less `bl:pick` and `bl:eventHandlers` replace their BPEL counterparts. BPEL's partner link concept is split to BPEL^{light} `bl:partners`, containers for partner endpoint references (EPRs), and `bl:conversations`, message exchanges that can involve several messages and partners ($\overline{s}$ 4 – contradicts BPEL's communication paradigm). Interaction activities can be arbitrarily bound to synchronous or asynchronous services (C1.2, C2.6, C4.4). BPEL^{light} results in an extension of design time and runtime environment (A1).

BPEL for Semantic Web Services (BPEL4SWS) by [57] proposes WSDL-less BPEL by removing these artifacts and thereby increasing the flexibility of business processes. In contrast to BPEL^{light}, BPEL4SWS uses semantic web technology, whereas BPEL^{light} uses straight-forward communication paradigms. BPEL4SWS uses a set of composable standards and specifications and is independent of any Semantic Web Service framework. It can be used to compose Semantic Web Services, traditional Web Services and a mix of them (A1, $\overline{s}$ 4, C1.2, C1.3, C1.6, C1.8, C2.6, C2.7, C3.2, C4.1, C4.4, C4.5).

OWL for BPEL integrates semantics in the form of OWL to BPEL [46]. Messaging activities are replaced by generic `ontcaf:service` element, which directly specifies its input and output data formats ($\overline{s}$ 4). The integrated OWL information is used to find a matching service for each specified service (A1, C1.2, C2.1, C3.1, C4.1, C4.5).

WS-BPEL Extensions for Versioning [29] addresses the problem of versioning BPEL processes and partner links (C1.4, C2.5, C3.1). The extension introduces new activities such as `versionHandlers` and adds attributes to existing activities such as `invoke`, `receive`, `import`, or `onMessage` in the BPEL namespace ($\overline{s}$ 2). It also extends the partner links concept at different levels of versioning. To use BPEL for Versioning the modeling tool, the process engine and the deployment mechanism must be upgraded (A1, C4.4).

"*BPEL for pervasive computing*" [22] introduces a multicast and publish/subscribe mechanism in BPEL 1.1 (C1.3). The aim is to make BPEL usable in pervasive and mobile computing scenarios where peers can enter or leave the network at any time and hence the number of message recipients is unknown at design time (C1.8). A new `ext:partnerGroup` construct works as list of endpoint references (EPRs). Management of this list is realized by `ext:add` and `ext:remove` activities to insert or delete EPRs, respectively. The `ext:reply` activity can exploit a partner group to send

messages to all contained partners, eventually realizing a multicast (C2.7, C3.2, C4.1, C4.5). Since several partners communicate with the process over one and the same partner link, there is a need to explicitly unbind a partner link (`ext:unbind` activity) and close its connection (`ext:close` activity) ($\overline{s}$ 4 – contradicts BPEL's communication paradigm). The approach requires a design time and runtime extension (A1).

T-BPEL [71] stands for "Transactional BPEL" and allows for attaching transaction requirements to a BPEL process and transaction capabilities to Web services. This enables a BPEL process to initiate distributed atomic transactions as well as compensation based transactions (C1.3, C2.7, C3.1, C4.1, C4.5). The extension is fully BPEL 1.1 compliant as it uses a separate namespace for its attributes (s8) and does not change the behavior of the BPEL engine.

7.3 Runtime only Extensions

In the case of a runtime only extension, the process model itself stays unchanged but other artifacts are touched, e.g., the deployment descriptor is modified. Runtime only approaches are not an extension in terms of Definition 1. We show them to emphasize the difference between a language extension and other forms of modifications and use the term "extension" for consistency with the terminology of the workflow community.

Runtime only extensions involve particular new components, but they have no impact on the modeling tool. It is possible, however, that such an extension offers other modeling tools for their particular purpose, different from BPEL modeling tools.

7.3.1. Runtime only Extensions by Vendors

"Business rules integration" is presented in Oracle [60]. Here, a business rule engine can be used within a BPEL process by using the `invoke` activity which calls a dedicated service (A3) for processing business rules (C1.2, C1.4, C1.6, C2.1, C3.2, C4.1, C4.5). This service interacts with a rules engine, which again is integrated with a rule authoring tool and a rules repository. For evaluation of a rule all required parameters are passed in the actual service call. The result of the business rule service invocation can then be used in further processing, e.g., as `transitionCondition` on a control link.

7.3.2. Runtime only Extensions by Research

BPEL'n'Aspects [32] is an approach of applying the aspect-oriented programming (AOP) paradigm [38] to BPEL processes to facilitate adaptations of running service compositions (C1.2). It enables to insert (or weave) aspects into processes without touching these processes themselves. Aspects are described by WS-Policy[72]. They contain a pointcut (i.e.,the place in the process to weave the aspect in) and an advice (i.e.,the functionality to weave in). Possible pointcuts are described by joinpoints that can currently be activities and transition conditions. In BPEL'n'Aspects, an advice is always a Web service invocation (C2.7). There are three advice types that denote whether the invocation ought to be carried out before, instead, or after a BPEL construct. Aspects are weaved into processes with the help of the WS-Policy Attachment mechanism [73]. BPEL'n'Aspects enables to insert aspects into single process instances, process instance groups, or all process instances of a process model (C1.4, C1.6, C3.2). The actual weaving can be done at runtime by the BPEL engine itself or by an external component (i.e.,the weaver) on basis of appropriate events created during workflow execution (C4.5). Since the engine itself is not aware of the executed aspects, the auditing needs to be extended in order to provide compensation capabilities.

AO4BPEL [13] is an approach similar to BPEL'n'Aspects, but enables BPEL snippets to be weaved into (running) processes (C1.2, C1.4, C1.6). Aspects are expressed as BPEL extension in BPEL namespace with an `aspect` element ($\overline{s}$ 2, $\overline{s}$ 4). Pointcuts are XPath expressions contained in a `pointcut` element. Each BPEL activity is thereby a possible joinpoint (C3.2). Advices are BPEL snippets nested in an `advice` element (C2.1, C2.7). An AO4BPEL implementation foresees an extended aspect-aware BPEL process engine and an aspect manager which execute activated aspects (C4.5).

A variant of *activity failure and recovery* is presented in [28] and [78]. They propose to change the way of service invocation to support handling of unavailable services by retrying invocation or replacing the called service (C1.7, C2.7, C3.1, C4.5). Both assume that the service is idempotent and that each operation implements an in/out operation. Both add a new deployment artifact which specifies a policy for handling a network fault.

A second variant of *activity and failure recovery* is presented in [33]. There, a transformation of a BPEL process is proposed. Each `invoke` in the input BPEL process is surrounded by a fault handler. In the case of a transportation fault, a service registry is invoked. The registry returns compatible services (C1.7, C2.7, C3.1, C4.5). Each service of the list is tried to be invoked subsequently until an invocation succeeds. The original BPEL process does not need to be modified. The generated BPEL process requires a service registry. Thus, we treat the extension as a runtime only extension, although the behavior of the transformed BPEL process does not rely on an extended BPEL engine.

SH-BPEL is a variant of "activity failure and recovery" [55] shows an enhancement of the invocation handler of a BPEL engine to support failure handling in the engine. Such failure handling includes replacing a service or to trigger human involvement. This extension is not an extension in our sense, since the runtime of BPEL is changed without any change of the BPEL process (C1.7, C2.7, C3.1, C4.5).

"*Extended WS-RM*" [13] also deals with reliability. In their case, they extend WS-Reliable Messaging [54] to support multi-party conversations specified in BPEL (C1.3, C2.7, C3.1, C4.5). WS-RM is a standard used to realize reliable messaging requirements on a SOAP level [77]. The extension is implemented in the invocation handler. The behavior of the invocation handler is configured by the deployment descriptor.

The *"Pluggable Framework for Enabling the Execution of Extended BPEL Behavior"* [34] offers a systematic mechanism to instrument BPEL engines so that behavior can be injected into a process (C1.2, C2.1, C2.2). The framework is based on a generic event model which can be mapped to lifecycle events of particular BPEL engine. These events are forwarded to a custom controller (C3.1), which can execute arbitrary behavior (e.g., require by an extension). The event may be a 'blocking event', in which case navigation is suspended on the respective path in the process until it receives an unblocking notification from the controller. Data in this notification may potentially affect how the navigation in the process proceeds. The additional behavior is effective during the execution of a process (C4.5).

"A Management Framework for WS-BPEL" [47] has the same aim as the pluggable framework (C1.2, C2.1, C2.2, C3.1, C4.5). In contrast to rely on events, it renders the activities of the BPEL process as resources and thus offers a uniform access scheme.

"*Business Rules Integration in BPEL*" [65] makes use of interceptors to trigger business rule checks. Interceptors can be attached before or after message sending/receiving activities. This mapping of interceptors on BPEL activities is provided by the person who models the process. That way, business rule definitions are separated from process logic (C1.2, C1.4, C1.8). An extended enterprise service bus (ESB) interprets the mapping and executes the business rules (C4.5). Negative evaluated rules cause the respective activity to be skipped (C2.1, C3.2). A transformation engine for message mediation and a rule broker allow the integration of different rule engines.

7.4 Summary

We discussed a huge variety of extensions addressing different aspects of the BPEL environment (Figures 1 and 2). Tab. 4 presents an overview of the extensions discussed including a characterization in terms of the criteria introduced in Sect. 3. The table has six columns: The column extension lists the name of the extension; Extd (Language Extended) states whether the BPEL language has been extended with any new construct; Conform states whether the extension is conform to Definition 1 using the standard conformity shortcuts of Tab. 1; A (Approach) lists the approaches that were applied (cf. Sect. 5); Type lists D or R denoting the type of the extension: D stands for a design time extension, R stands for a runtime extension; Characteristics lists the characteristics of the extension (referring to Tab. 3).

Our classification is based on a literature study. We did not interview the extensions authors to find out the thoughts behind their extension design. We assume that all the authors fulfilled their goals as their extensions are available. When the authors followed our extension development guidelines presented in Sect. 6, they would have possible chosen another way. For instance, for enabling a retry of failed calls, the invocation handler could be modified.

Tab. 4: Extension Overview

Extension	Extd	Conform	A	Type	Characteristics
A Management Framework for WS-BPEL [47]	No	n/a	n/a	R	C1.2, C2.1, C2.2, C3.1, C4.5
Activity failure and recovery [6]	Yes	No ($\overline{S}$ 4)	A1	D, R	C1.7, C2.7, C3.1, C4.1, C4.5
"Activity failure and recovery" [28],[78]	No	n/a	n/a	R	C1.7, C2.7, C3.1, C4.5
"Activity failure and recovery" [33]	No	n/a	n/a	R	C1.7, C2.7, C3.1, C4.5
"Activity failure and recovery" [52]	No	n/a	n/a	D	C1.7, C2.1, C3.2, C4.1
"Activity failure and recovery" [54]	Yes	No ($\overline{S}$ 2)	A2	D	C1.2, C1.3, C2.2, C3.2, C4.1
AO4BPEL [13]	Yes	No ($\overline{S}$ 2, $\overline{S}$ 4)	n/a	R	C1.2, C1.4, C1.6, C2.1, C2.7, C3.2, C4.5
Auto complete copy destination [6]	Yes	No ($\overline{S}$ 3, $\overline{S}$ 4)	A1	D, R	C1.3, C1.4, C2.1, C3.2, C4.1, C4.5
"BPEL for Pervasive Computing" [22]	Yes	No ($\overline{S}$ 4)		D, R	C1.3, C1.8, C2.7, C3.2, C4.1, C4.5
BPEL for REST [63]	Yes	No ($\overline{S}$ 2, $\overline{S}$ 4)	A1	D, R	C1.3, C2.1, C2.6, C3.1, C4.1, C4.5
BPEL fragments [53]	Yes	No ($\overline{S}$ 5)		D	C1.6, C2.1, C3.2, C4.1
BPEL process templates [30]	Yes	No ($\overline{S}$ 2)	n/a	D	C1.2, C1.6, C2.5, C3.2, C4.1, C4.4
BPEL/SQL [74]	Yes	Yes (s7)	A1, A2, A3	D, R	C1.3, C1.5, C2.2, C3.1, C3.2, C4.1, C4.4, C4.5
BPEL'n'Aspects [32]	No	n/a	n/a	R	C1.2, C1.4, C1.6, C2.7, C3.2, C4.5
BPEL4Chor [17]	Yes	Yes (s8)	A2	D	C1.1, C2.1, C3.2, C4.1, C4.2
BPEL4Grid [18], [82]	Yes	No ($\overline{S}$ 2)	A1	D, R	C1.3, C1.4, C2.2, C2.7, C3.1, C3.2, C4.1, C4.5
BPEL4People [3]	Yes	Yes (s7)	A1, A3	D, R	C1.3, C2.1, C2.7, C3.1, C3.2, C3.3, C4.1, C4.5
BPEL4SWS [57]	Yes	No ($\overline{S}$ 4)	A1	D, R	C1.2, C1.3, C1.6, C1.8, C2.6, C2.7, C3.2, C4.1, C4.4, C4.5
BPEL-D [36]	Yes	No ($\overline{S}$ 4)	n/a	D	C1.1, C2.8, C3.2, C4.1
BPEL data transitions [21]	Yes	No ($\overline{S}$ 4)	A2, A3	D, R	C1.3, C1.5, C1.8, C2.4, C3.1, C4.1, C4.2, C4.5
BPELJ [10]	Yes	No ($\overline{S}$ 4)	A1	D, R	C1.3, C1.5, C1.8, C2.1, C2.2, C2.3, C2.8, C3.2, C4.1, C4.5
BPEL^{light} [56]	Yes	No ($\overline{S}$ 4)	A1	D, R	C1.2, C1.3, C1.6, C1.8, C2.6, C2.7, C3.2, C4.1, C4.4, C4.5
BPEL-SPE [40]	Yes	No ($\overline{S}$ 2)	A1	D, R	C1.1, C1.3, C1.4, C1.6, C1.8, C2.5, C2.7, C3.2, C4.1, C4.4, C4.5, C4.6
Business Rules Integration in BPEL [65]	No	n/a	n/a	R	C1.2, C1.4, C1.8, C2.1, C3.2, C4.4, C4.5
Business Rules Integration [60]	No	n/a	A3	R	C1.2, C1.4, C1.6, C2.1, C3.2, C4.1, C4.5
Collaborative Scopes [25]	Yes	Yes	A1	D, R	C1.3, C2.1, C3.1, C3.2, C4.1, C4.5
Continue on error [25]	Yes	Yes (s8)	A1	D,R	C1.7, C2.7, C3.1, C4.1, C4.5
Context4BPEL [80]	Yes	Yes (s9)	A1	D, R	C1.3, C2.1, C2.2, C2.3, C2.7, C3.2, C4.1, C4.5, C4.6
Cross-process fault handling [42]	No	No ($\overline{S}$ 4)	n/a	D, R	C1.3, C1.7, C2.1, C3.1, C3.2, C4.1, C4.5

Extension	Extd	Conform	A	Type	Characteristics
Dedicated Administrator [25]	Yes	Yes (s8)	A1	D, R	C1.7, C2.5, C3.1, C4.1, C4.5
E4X extension for BPEL [48]	Yes	Yes (s7, s10)	A1	D, R	C1.4, C2.3, C3.2, C4.1, C4.5
Execution as Subprocess [25]	Yes	Yes (s8)	A1	D, R	C1.3, C2.1, C2.7, C3.1, C3.2, C4.1, C4.5
Extended WS-RM" [14]	No	n/a	n/a	n/a	C1.3, C2.7, C3.1, C4.5
Generalized Flow [25]	Yes	No ($\overline{s}$ 4)	A1	D, R	C1.3, C1.8, C2.1, C3.2, C4.1, C4.5
Headers handling [6]	Yes	No ($\overline{s}$ 3)	A1	D, R	C1.8, C2.1, C3.2, C4.1, C4.5
id attribute	Yes	Yes	n/a	D	C1.4, C2.5, C3.2, C4.1
Ignore unavailable data [6]	Yes	No ($\overline{s}$ 3, $\overline{s}$ 4)	A1	D, R	C1.3, C1.4, C2.1, C3.2, C4.1, C4.5
ii4BPEL [23]	Yes	Yes (s7)	A1, A2, A3	D, R	C1.3, C1.5, C2.2, C3.1, C3.2, C4.1, C4.4, C4.5
Implicit correlations [6]	Yes	No ($\overline{s}$ 4)	A1	D, R	C1.3, C2.7, C3.1, C4.1, C4.5
Iterable forEach [6])	Yes	No ($\overline{s}$ 3, $\overline{s}$ 4)	A1	D, R	C1.3, C2.1, C3.2, C4.1, C4.5
Java Snippets [25]	Yes	Yes	A1	D, R	
"Retry or alternative service" [28],[78]	No	n/a		R	C1.7, C2.6, C2.7, C3.1, C4.4, C4.5
Microflows [25]	Yes	Yes	A1	D, R	C1.5, C2.5, C3.1, C4.1, C4.5
New XPath functions, e.g. [6]	No	Yes		D, R	C1.3, C2.5, C3.2, C4.1, C4.5
Non-compensatable scopes [25])	Yes	No ($\overline{s}$ 4)	A1	D, R	C1.3, C2.5, C3.1, C4.1, C4.5
Oracle Human Task [61]	Yes	Yes (s9)	A3	D	C1.8, C2.7, C3.1, C4.1
Oracle Notification Service [61]	Yes	Yes (s9)	A3	D	C1.8, C2.7, C3.1, C4.1
"OWL for BPEL" [46]	Yes	No ($\overline{s}$ 4)	A1	D, R	C1.2, C2.1, C3.1, C4.1, C4.5
Parameterized Processes [30]	Yes	No ($\overline{s}$ 2)	A1	D, R	C1.2, C1.6, C1.7, C2.6, C2.7, C3.2, C4.1, C4.4, C4.5, C4.6
Pluggable Framwork for Enabling the Execution of Extended BPEL Behavior [35]	No	n/a	n/a	R	C1.2, C2.1, C2.2, C3.1, C4.5
Process context [6]	Yes	No ($\overline{s}$ 4)	A1	D, R	C1.3, C1.4, C2.1, C2.2, C3.2, C4.1
References in BPEL [79]	Yes	No ($\overline{s}$ 2)	A2, A3	D	C1.8, C2.2, C2.8, C2.4, C3.1, C3.2, C4.1, C4.4
Resource-oriented BPEL [6];[62]	Yes	No ($\overline{s}$ 4)	A1	D,R	
Retry Scopes [19]	Yes	No ($\overline{s}$ 4)	A1	D, R	C1.3, C1.7, C2.1, C3.2, C4.1, C4.5
SH-BPEL [55]	No	n/a	n/a	R	C1.7, C2.7, C3.1, C4.5
SWRL for BPEL [81]	Yes	Yes (s9)	A2	D	C1.4, C2.1, C3.2, C4.1
T-BPEL [71]	Yes	Yes (s8)	A2	D, R	C1.3,C2.7,C3.1,C4.1,C4.5
Transaction boundaries [25]	Yes	No ($\overline{s}$ 4)	A1	D, R	C1.5, C2.5, C3.1, C4.1, C4.5
WS-BPEL extension for versioning [29]	Yes	No ($\overline{s}$ 2)	A1	D, R	C1.4, C2.5, C3.1
xBPEL [11]	Yes	No ($\overline{s}$ 2)	A2, A3	D	C1.2, C1.3, C2.1, C3.3, C4.1, C4.5

8. Conclusions

BPEL extensions are omnipresent in research and industry, but no comparison or classification was available, neither there are best practices and recommendations for design and implementation of

extensions. The only related research is a solution for architectural decision points [83], but there are no decision points defined specially for BPEL extensions, yet. Balko [7] regard extensibility as a property of a process model to be adaptable. This is in contrast to our definition, which regards the extensibility of the modeling language itself.

The main contribution of this paper is a comprehensive framework for understanding and classifying BPEL extensions, and a recommendation for developing BPEL extensions properly. For providing that knowledge, first the classification for BPEL extensions is given and based on that an overview of the state of the art of BPEL extensions is given. Furthermore as practical advice we give a design guideline that raises different questions for deciding wether a BPEL extension has to be implemented or the functionality can be realized in another way.

Interesting to note is that only around half of the discussed extensions are standard-conform BPEL extensions in terms of Definition 1. Standard-conform extensions have their advantage in being portable and re-usable across different BPEL environments. Needless to say, non-conforming extensions also have their justification. Thus, if an extension is not conforming to the BPEL standard, it does not imply that it is of no use or that it is realized in a wrong way. As we have shown, valid extensions to BPEL can include anything ranging from new attributes to new elements, to extended assign operations up to completely new activities. We have also shown that missing functionality can be implemented in different ways, for instance using standard language constructs or introducing extension attributes or extension activities. However, when talking about extensions, one has to be aware that the ways of extending BPEL foreseen in the specification are limited.

The presented discussion on possibilities to realize an extension remains valid in context of the Business Process Model and Notation (BPMN) language. As part of our future work, we will classify BPMN extensions according to the presented classification framework.

9. Acknowledgements

The work published in this paper was partially funded by the COMPAS project (contract no. FP7-215175) and the ALLOW project (contract no. FP7-213339) under the EU 7th Framework Programme Information and Communication Technologies Objective, the DFG Cluster of Excellence Simulation Technology (EXC310), the DFG project Nexus (SFB627), and the S-Cube project under the Network of Excellence (contract no. FP7-215483).

10. References

[1] van der AALST, W.M.P., WESKE, M. & GRÜNBAUER, D. (2004), Case handling: a new paradigm for business process support, *Data & Knowledge Engineering*, 53(2), pp. 129-162.

[2] ADAMS, P., EASTON, P., JOHNSON, ERIC, MERRICK, R. & PHILLIPS, M. (2010). SOAP over Java Message Service 1.0, W3C Working Draft 26 October 2010.

[3] AGRAWAL, A., AMEND, M., DAS, M., FORD, M., KELLER, C., KLOPPMANN, M., KÖNIG, D., LEYMANN, F., MÜLLER, R., PFAU, G., PLÖSSER, K., RANGASWAMY, R., RICKAYZEN, A., ROWLEY, M., SCHMIDT, P., TRICKOVIC, I., YIU, A. & ZELLER M. (2007a). WS-BPEL Extension for People (BPEL4People), Version 1.0, White Paper.

[4] AGRAWAL, A., AMEND, M., DAS, M., FORD, M., KELLER, C., KLOPPMANN, M., KÖNIG, D., LEYMANN, F., MÜLLER, R., PFAU, G., PLÖSSER, K., RANGASWAMY, R., RICKAYZEN, A., ROWLEY, M., SCHMIDT, P., TRICKOVIC, I., YIU, A. & ZELLER M. (2007b). Web Services Human Task (WS-HumanTask), Version 1.0, White Paper.

[5] ALONSO, G., CASATI F., KUNO, H. & MACHIRAJU, V. (2004), *Web Services*, Springer. 354 p.

[6] Apache ODE group (2009). BPEL Extensions. URL: http://ode.apache.org/bpel-extensions.html.

[7] BALKO, S., TER HOFSTEDE, A. H. M., BARROS, A. P. & ROSA, M. (2009). Controlled Flexibility and Lifecycle Management of Business Processes through Extensibility. *3rd International Workshop on Enterprise Modelling and Information Systems Architectures*, GI.

[8] BARROS, A., DECKER, G., DUMAS, M. & WEBER, F. (2007). Correlation Patterns in Service-Oriented Architectures. *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering* (FASE), Springer Verlag.

[9] BISCHOF, M, KOPP, O., VAN LESSEN, T. & LEYMANN, F. (2009). BPELscript: A Simplified Script Syntax for WS-BPEL 2.0. *35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2009).*

[10] BLOW, M., GOLAND, Y., KLOPPMANN, M., LEYMANN, F., PFAU, G., ROLLER, D. & ROWLEY, M. (2004). BPELJ: BPEL for Java, White Paper, BEA.

[11] CHAKRABORTY, D. & LEI, H. (2004) Pervasive Enablement of Business Processes, *Proc. of the Second IEEE Intl. Conf. on Pervasive Computing and Communications (PerCom 2004).*

[12] CHAPPELL, D. (2004). *Enterprise Service Bus: Theory in Practice,* O'Reilly Media.

[13] CHARFI, A. & MEZINI, M. (2004). Aspect-Oriented Web Service Composition with AO4BPEL. In: *European Conference on Web Services (ECOWS)*.

[14] CHARFI, A., SCHMELING, B. & MEZINI, M. (2006). Reliable Messaging for BPEL Processes. *IEEE International Conference on Web Services (ICWS)*, pp. 59-66.

[15] CHRISTENSEN, E., CURBERA, F., MEREDITH, G. & WEERAWARANA, S. (2001). *Web Services Description Language (WSDL) 1.1.* World Wide Web Consortium W3C (March 2001)

[16] CURBERA, F., DUFTLER, M., KHALAF, R. & LOVELL, D. Bite: Workflow Composition for the Web. *International Conference on Service Oriented Computing* LNCS, vol. 4749, pp.94-106, Springer Heidelberg *(2007)*.

[17] DECKER, G., KOPP, O., LEYMANN, F. & WESKE, M. (2009). Interacting services: from specification to execution, *Data & Knowledge Engineering*, doi:10.1016/j.datak.2009.04.003.

[18] DÖRNEMANN, T., FRIESE, T., HERDT, S., JUHNKE, E. & FREISLEBEN B. (2007). Grid Workflow Modelling Using Grid-Specific BPEL Extensions, *Proceedings of German e-Science Conference 2007*, pp. 1-9.

[19] EBERLE, H., KOPP, O., UNGER, T. & LEYMANN. (2009). F. Retry Scopes to Enable Robust Workflow Execution in Pervasive Environments, *2nd Workshop on Monitoring, Adaptation and Beyond (MONA+)*.

[20] FONDEN, C. (2009): Konzeption und Entwicklung von Kontexterweiterungen für Workflows, Diploma Thesis 2901, Institute of Architecture of Application Systems, University of Stuttgart.

[21] HABICH, D., RICHLY, S., PREISSLER, S., GRASSELT, M., LEHNER, W., MAIER, A. (2007) BPEL-DT - Data-aware Extension of BPEL to Support Data-Intensive Service Applications. In C. Pautasso and T. Gschwind, editors, *WEWST*, volume 313 of CEUR Workshop Proceedings. CEUR-WS.org.

[22] HACKMANN, G., GILL, C. & ROMAN, G.-C. (2007). Extending BPEL for Interoperable Pervasive Computing. *IEEE International Conference on Pervasive Computing*, pp. 204-213.

[23] IBM (2006). Adding an information service activity to a business process, URL: http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.is.bpel.help.doc/topics/accessdata.htm.

[24] IBM (2007). Mapping Specification Language. http://www.research.ibm.com/journal/sj/452/roth.html.

[25] IBM (2009). Working with BPEL extensions, URL: http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.wbit.620.help.bpel.ui.doc/concepts/cextent.html.

[26] International Organization for Standardization (2006). Information Technology — ECMAScript for XML (E4X) Specification. ISO/IEC 22537:2006.

[27] JÄGER, C. & WOLKE, C. (2008) *Make-or-Buy Decisions - A Transaction Cost Theoretical Approach to the Assessment of Outsourcing Activities*, Books on Demand GmbH.

[28] JUHNKE, E., DÖRNEMANN, T. & FREISLEBEN, B. (2009). Fault-Tolerant BPEL Workflow Execution via Cloud-Aware Recovery Policies. *Proceedings of the 35th EUROMICRO Conference on Software Engineering and Advanced Applications*.

[29] JURIC, M., SASA, A. & ROZMAN I. (2009). WS-BPEL Extensions for Versioning, *Information and Software Technology 51*, pp. 1261–1274.

[30] KARASTOYANOVA, D. (2006). Enhancing Flexibility and Reusability of Web Service Flows through Parameterization, *PhD thesis*, TU Darmstadt and Universität Stuttgart.

[31] KARASTOYANOVA, D., KHALAF, R. , SCHROTH, R., PALUSZEK, M. & LEYMANN, F. (2006). BPEL Event Model. *University of Stuttgart, Technical Report Computer Science No. 2006/10.*

[32] KARASTOYANOVA, D. & LEYMANN, F. (2009). BPEL'n'Aspects: Adapting Service Orchestration Logic, *Proceedings of 7th IEEE International Conference on Web Services (ICWS)*.

[33] KARELIOTIS, C., VASSILAKIS, C. & GEORGIADIS, P. (2007). Enhancing BPEL scenarios with Dynamic Relevance-Based Exception Handling. In: *Proceedings of IEEE International Conference on Web Services (ICWS)*., 2007, pp.751-758

[34] KHALAF, R. (2008). Supporting business process fragmentation while maintaining operational semantics: a BPEL perspective, Dissertation, University of Stuttgart, Germany.

[35] KHALAF, R., KARASTOYANOVA, D. & LEYMANN, F. (2007). Pluggable Framework for Enabling the Execution of Extended BPEL Behavior. *Proceedings of the 3rd International Workshop on Engineering Service-Oriented Application (WESOA'2007)*.

[36] KHALAF, R. & LEYMANN, F. (2006). Role-based Decomposition of Business Processes using BPEL. *International Conference on Web Services (ICWS)* pp. 770-780.

[37] KHALAF, R., SUBRAMANIAN, R., MIKALSEN, T., DUFTLER, M., DIAMENT, J. & SILVA-LEPE, I. Enabling Community Participation for Workflows through Extensibility and Sharing, *Workshop on Business Process Management and Social Software (BPMS2'09)*, Springer.

[38] KICZALES, G. (1997). Aspect-Oriented Programming, *Proceedings of ECOOP'97*.

[39] KLOPPMANN, M., KOENIG, D., LEYMANN, F., PFAU, G., RICKAYZEN, A., RIEGEN, C., SCHMIDT, P. & TRICKOVIC, I. (2005). WS-BPEL Extension for Sub-processes - BPEL-SPE, White Paper.

[40] KLOPPMANN, M., KÖNIG, D., LEYMANN, F., PFAU, G., RICKAYZEN, A., VON RIEGEN, C., SCHMIDT, P. & TRICKOVIC, I. (2005). WS-BPEL Extension for People – BPEL4People, White Paper.

[41] KÖNIG, D., LOHMANN, N., MOSER, S., STAHL, C. & WOLF, K. (2008): Extending the compatibility notion for abstract WS-BPEL processes. *WWW 2008*, 785-794.

[42] KOPP, O., MARTIN, D., WUTKE, D. & LEYMANN, F. (2009). The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *Enterprise Modelling and Information Systems. Vol. 4(1)*, pp.3-13.

[43] KOPP, O., WIELAND, M. & LEYMANN, F. (2009). Towards Choreography Transactions. *1st Central-European Workshop on Services and their Composition (ZEUS)*.

[44] LAEMMEL, R. & OSTERMANN, K. (2006). Software Extension and Integration with Type Classes, *Proceedings of the 5th international conference on Generative Programming and Component Engineering (GPCE '06)*.

[45] LAU, C., BEATON, M. (2004) Architecting on demand solutions, Part 3: Use BPEL to create business processes, *IBM developerWorks*. URL: http://www.ibm.com/developerworks/library/i-odoebp3/.

[46] DUY NGAN LE, NGOC SON NGUYEN, MOUS, K., KO, R.K.L., GOH, A.E.S., (2009). Generating Request Web Services from Annotated BPEL. *RIVF´09 International Conference on Computing and Communication Technologies*, Da Nang 2009, Print ISBN: 978-1-4244-4566-0

[47] van LESSEN, T., LEYMANN, F., MIETZNER, R., NITZSCHE, J. & SCHLEICHER, D. (2009). A Management Framework for WS-BPEL. *Proceedings of the 6th IEEE European Conference on Web Services*. IEEE Pervasive Computing, 8, 66–74, 2009.

[48] van LESSEN, T., NITZSCHE, J. & KARASTOYANOVA, D. (2009). Facilitating Rich Data Manipulation in BPEL using E4X. . *1st Central-European Workshop on Services and their Composition (ZEUS)*. http://CEUR-WS.org/Vol-438/paper16.pdf.

[49] LEYMANN, F. (2005). The (Service) Bus: Services Penetrate Everyday Life. Boualem Benatallah, Fabio Casati, Paolo Traverso (Eds.): *Service-Oriented Computing - ICSOC 2005, Third International Conference, Amsterdam,* Proceedings. Lecture Notes in Computer Science 3826 Springer 2005, ISBN 3-540-30817-2.

[50] LEYMANN, F. & ROLLER, D. (2000). *Production workflow: concepts and techniques,* Prentice Hall PTR.

[51] LEYMANN, F. & ROLLER, D. (2006). Modeling business processes with BPEL4WS. *Information Systems and e-Business Management (ISeB)*, Springer. 265-284.

[52] LIU, A., LI, Q., HUANG, L. & XIAO, M. (2007). A Declarative Approach to Enhancing the Reliability of BPEL Processes, *IEEE International Conference on Web Services (ICWS)*, pp. 272- 279.

[53] MA, Z. & LEYMANN, F. (2009), BPEL Fragments for Modularized Reuse in Modeling BPEL Process. *5th International Conference on Networking and Services (ICNS).*

[54] MODAFFERI, S. & CONFORTI, E. (2006). Methods for enabling recovery actions in ws-bpel. In *Proc. of Int. Conf. on Cooperative Information Systems (CoopIS)*.

[55] MODAFFERI, S., MUSSI, E. & PERNICI, B. (2006). SH-BPEL: a self-healing plug-in for Ws-BPEL engines. *Proceedings of the 1st Workshop on Middleware for Service Oriented Computing, MW4SOC*, ACM New York, 2006, ISBN:1-59593-425-1.

[56] NITZSCHE, J., van LESSEN, T., KARASTOYANOVA, D. & LEYMANN, F. (2007a). BPEL light. In: *Proceedings of the 5th International Conference on Business Process Management* (BPM 2007). Lecture Notes in Computer Science; 4714, pp. 214-229, Springer.

[57] NITZSCHE, J., VAN LESSEN, T., KARASTOYANOVA, D. & LEYMANN, F. (2007b). BPEL for Semantic Web Services (BPEL4SWS). In: *Proceedings of the 3rd International Workshop on Agents and Web Services in Distributed Environments* (AWeSome'07) - On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops. Lecture Notes in Computer Science; 4805, pp. 179-188.

[58] OASIS (2004). Web Services Reliable Messaging TC: WS-Reliability 1.1, *OASIS Standard*. http://docs.oasis-open.org/wsrm/ws-reliability/v1.1.

[59] OASIS (2007). Web Services Business Process Execution Language Version 2.0, *OASIS Standard*. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[60] Oracle (2006). BPEL Process Manager: BPEL + Business Rules. URL: http://www.oracle.com/technology/products/ias/bpel/pdf/bpelandbusinessrules.pdf.

[61] Oracle (2007). BPEL Process Manager Developer's Guide. Version 10g (10.1.3.1.0) B28981-03. URL: http://download.oracle.com/docs/cd/B31017_01/integrate.1013/b28981.pdf.

[62] OVERDICK, H. (2003). Towards resource-oriented BPEL. In *2nd ECOWS Workshop on Emerging Web Services Technology*. http://ceur-ws.org/Vol-313/paper8.pdf.

[63] PAUTASSO, C. (2008). BPEL for REST. *7th International Conference on Business Process Management*.

[64] PELTZ, C. (2003). Web Services Orchestration and Choreography, *IEEE Computer*, 36, pp.46-52.

[65] ROSENBERG, F. & DUSTDAR, S. (2005). Business Rule Integration in BPEL – A Service-Oriented Approach, *Proceedings of the 7th International IEEE Conference on E-Commerce Technology.*

[66] SCHUMM, D. (2007) A Graphical Tool for Modeling BPEL 2.0 Processes, *Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 2124*.

[67] SCHUMM, D., KARASTOYANOVA, D., LEYMANN, F. & NITZSCHE, J. (2009). On Visualizing and Modelling BPEL with BPMN, *Proceedings of the 4th International Workshop on Workflow Management (ICWM)*.

[68] SILVA-LEPE, I., SUBRAMANIAN, R., ROUVELLOU, I., MIKALSEN, T., DIAMENT, J. & IYEN-GAR, A. (2008). SOAlive Service Catalog:ASimplied Approach to Describing, Discovering and Composing Situational Enterprise Services. *Proceedings of the International Conference on Service-Oriented Computing (ICSOC)*.

[69] STAHL, T., VÖLTER, M. & CZARNECKI, K. (2006). Model-driven Software Development*: technology, engineering, management*. John Wiley & Sons.

[70] STEIN, S., KÜHNE, S. & IVANOV, K. (2009). Business to IT Transformations Revisited. In: *Business Process Management Workshop* : BPM 2008 Internation Workshops, Milano, 2008, Revised Papers, Nr. 17 Berlin: Springer (2009), pp. 176--187.

[71] TAI, S., MIKALSEN, T. A., WOHLSTADTER, E., DESAI, N. & ROUVELLOU, I. (2004). Transaction policies for service-oriented computing, *Data Knowl. Eng*, 51, pp. 59-79.

[72] VEDAMUTHU, A.S. et. al (2007a). Web Services Policy 1.5 – Framework, *W3C Recommendation 04 September 2007*, URL: http://www.w3.org/TR/ws-policy/.

[73] VEDAMUTHU, A.S. et al. (2007b). Web Services Policy 1.5 – Attachment, W3C Recommendation 04 September 2007, URL: http://www.w3.org/TR/ws-policy-attach/.

[74] VRHOVNIK, M., SCHWARZ, H., SUHRE, O., MITSCHANG, B., MARKL, V., MAIER, A. & KRAFT T. (2007). An approach to optimize data processing in business processes, *Proceedings of the 33rd international conference on Very large data bases* 2007, pp. 615-626.

[75] VRHOVNIK, M., SCHWARZ, H., RADESCHIITZ, S. & MITSCHANG, B. (2008). An Overview of SQL Support in Workflow Products, *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference,* pp.: 1287 - 1296 , ISBN: 978-1-4244-1836-7.

[76] WEIDLICH, M., DECKER, G., GROSSKOPF, A. & WESKE, M. (2008). BPEL to BPMN: The Myth of a Straight-Forward Mapping. *International Conference on Cooperative Information Systems (CoopIS)*.

[77] WEERAWARANA, S., CURBERA, F., LEYMANN, F., STOREY, T. & FERGUSON, D.F. (2005). *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*, Prentice Hall PTR.

[78] WEN, JIAJIA, CHEN, JUNLINAG, PENG, YONG & XU, MENG. (2006). A Multi-Policy Exception Handling System for BPEL Processes. *First International Conference on Communications and Networking in China*.

[79] WIELAND, M., GÖRLACH, K., SCHUMM, D. & LEYMANN, F. (2009). Towards Reference Passing in Web Service and Workflow-based Applications, *Proceedings of the 13th IEEE Enterprise Distributed Object Conference (EDOC 2009)*.

[80] WIELAND, M., KOPP, O., NICKLAS, D. & LEYMANN, F. (2007). Towards Context-aware Workflows, *CAiSE'07 Proceedings of the Workshops and Doctoral Consortium Vol 2*.

[81] WU, YUNZHOU & DOHSI, PRASHANT. (2008). Making BPEL Flexible – Adapting in the Context of Coordination Constraints Using WS-BPEL, *International Conference on Services Computing (SCC 2008).*

[82] ZHANG, HUAJIAN, FAN, XIAOLIANG, ZHANG, RUISHENG, LIN, JIAZAO, ZHAO, ZHILI & LI, LIAN (2008). Extending BPEL2.0 for Grid-Based Scientific Workflow Systems, *Asia-Pacific Services Computing Conference (APSCC '08).*

[83] ZIMMERMANN, O., KOEHLER, J., LEYMANN, F., POLLEY, R. & SCHUSTER N. (2009). Managing Architectural Decision Models with Dependency Relations, Integrity Constraints, and Production Rules. *Journal of Systems and Software,* Elsevier. 82(8), pages 1249-1267.

Appendix H

A Penalty-based Approach for QoS Dissatisfaction using Fuzzy Rules

A Penalty-based Approach for QoS Dissatisfaction using Fuzzy Rules

Barbara Pernici[1], S. Hossein Siadat[1], Salima Benbernou[2], and Mourad Ouziri[2]

[1] Politecnico di Milano, Italy
[2] LIPADE, Université Paris Descartes, France

Abstract. Quality of Service (QoS) guarantees are commonly defined in Service Level Agreements (SLAs) between provider and consumer of services. Such guarantees are often violated due to various reasons. QoS violation requires a service adaptation and penalties have to be associated when promises are not met. However, there is a lack of research in defining and assessing penalties according to the degree of violation. In this paper, we provide an approach based on fuzzy logic for modelling and measuring penalties with respect to the extent of QoS violation. Penalties are assigned by means of fuzzy rules.

Key words: QoS, service level agreement, penalty, fuzzy logic

1 Introduction

QoS guarantees defined in contracts may be violated due to various reasons. This situation needs to be handled through applying adaptation techniques not to bring dissatisfaction. The concept of penalty has been used in SLAs to compensate the conditions under which guarantee terms are not met [1]. Despite some research have been done on the description, negotiation and monitoring of SLAs, however there is not much work on the definition of penalty clauses. [4] studied on WS-Agreement specification to define penalties based on different types of violation. However, penalties are assigned to violation of a single property instead of assigning penalties to violation of overall QoS. Moreover, the approach introduces a method for measuring penalties which is for fixed predefined number of violations, instead of measuring the extent of violation and assigning penalties accordingly.

One main issue is how to determine the appropriate amount of penalties as compensations from providers to satisfy customers. As quality parameters can be satisfied partially, the assessment of penalties can be based on the degree of quality violation. Understanding the violation degree is a prerequisite for assessing penalties. However, measuring such violation is yet an open research challenge. In addition, the influencing factors in defining penalties need to be identified. A static amount of penalty (manual approaches) does not reflect the extent of violation at runtime. The amount and level of penalties are related to the degree of quality violation provided from the provider side. On the other

side, the customers characteristics may also affect the amount of penalties. For example a penalty to satisfy a gold/loyal customer is different with the one for an occasional customer. To the best of our knowledge, there is no formal relation between the assigned penalty and its influencing factors. Moreover, the extent and type of penalties are not clearly expressed in related work. However, understanding such relation and providing a mapping between them are complicated issues. We argue what is missing is a suitable mechanism for modelling penalties that takes into account both provider and consumer sides. Apart from the degree of violation, we also consider the state of customer and service provider with respect to their past history (e.g. whether the service has been penalised previously) in determining the right amount of penalties. However, as the relation between a given penalty and its influencing factors is not linear, conventional mathematical techniques are not applicable for modelling penalties.

Recent approaches are dealing with the issue of partial satisfaction for quality commitments and different techniques were used such as applying soft constraint [6], fuzzy sets [3] and semantic policies [2]. Among them, [6] introduced the concept of penalties for unmet requirements. However, defining penalties and finding a relation between the assigned penalties and the violated guarantees are remained challenges in similar approaches. The goal of this paper is to apply an inference technique using fuzzy logic as a solution [5] to propose a penalty-based approach for compensating conditions in which quality guarantees are not respected. Fuzzy logic is well suited for describing QoS and measuring quality parameters [3]. We demonstrate a penalty inference model with a rule-based mechanism applying fuzzy set theory. Measuring an appropriate value for penalties with respect to the amount of violation is the main contribution of the paper.

In the following, we start by a motivating example in Section 2. In Section 3 we show the descriptions of penalties and in Section 4 we provide a rule-based system using fuzzy set theory for modelling and reasoning penalties. Section 5 shows some experiments in applying penalty for the problem of QoS dissatisfaction and we conclude the paper in Section 6.

2 Motivating Example

Let's assume that a user is wishing to use a food delivery service. Therefore, a contract is established between the user and the service provider. The contract defines non functional criteria such as delivery time, quality of the perceived service (the quality of food during the delivery service, for example the food is maintained at the ideal temperature), and availability of the delivery service. Therefore we define a list of parameters for our example as follows: time to delivery (t_d), quality of delivered food (q_d), availability of delivery service (a_d). These quality parameters together with a list of penalty terms are defined in a contract and illustrated in Table 1.

The delivery service will be penalized if it is not able to provide the quality ranges defined in the contract. An overall QoS violation will be calculated first

Quality Parameters	time to delivery	between 10 to 15 min
	quality of delivered food	between 0.8 to 1
	availability	between 90 to 100% of the time
Penalties:	Minor or Null penalties	
	Penalties on quality parameters	
	Extra Service penalties	
	Termination penalty	

Table 1: motivating example

and afterwards a penalty is assigned with respect to the extent of the violation. We also take into account customer and provider perspectives by considering parameters from both parties. Parameters such as history of a delivery service and state of a customer can be involved. The history of a service shows whether the service is penalized previously. This can influence the amount of given penalties for future. The current state of a customer presents the importance of the customer for service provider. For example, minor violation of service delivery can cause a major penalty for provider in case the customer is gold (with good history). In contrast, a normal customer (with ordinary history) will not be given any extra service if the quality of delivered food is not good.

3 Definition of Penalties

In order to provide a formal model of penalties and build a reasoning mechanism to handle the penalties in the contract, in the following we try to summarize the different types of penalties that can be applied. We categorize the penalties into two main classes:

1. **Numerical penalties**: They are related to measurable qualities of service. In other words, we have to handle and work with variables of the service (e.g *the availability* > 0.9, the $responsetime < 0.2ms$).

2. **Behavioural penalties**: They are related to the behaviour of either the customer or the service provider. Consider the following case: a merchant wishes to obtain a service for online payment by bank card. The financial institution offered a 25% off if the settlement proceeds within two days of the request. Beyond these two days, the penalty is such as the trader does not have the discount and will therefore be required to pay all fees.

A penalty clause in an SLA may be of the following types:

– Penalty Null and denoted by P_0: no penalty is triggered because all agreed QoS are satisfied or minor violation has occurred.
– Penalty on the QoS: a penalty should be triggered on one of the QoS parameters Q_j in the contract if Q_i is not fulfilled.
– Penalty on the penalty: a new penalty P_j should be triggered if the previous one P_i is unfulfilled. Such penalty will be handled through the long term contract validation. The reasoning on the time aspect of the contract is out of the scope of the paper.

– Extra service penalty: if a QoS is not fulfilled by the service provider, to penalize him, an extra service might be offered to the customer.
– Cancellation penalty: this is a dead penalty for the service provider. A service substitution occurs.

4 Modelling Penalties

We present a fuzzy model to express penalties in a rule-base system. Our fuzzy penalty model is defined by the couple $FP =< \mathcal{S}, \mathcal{R} >$, where $\mathcal{S}$ is a fuzzy set on penalties and $\mathcal{R}$ is a set of inference rules.

4.1 Fuzzy sets for penalties

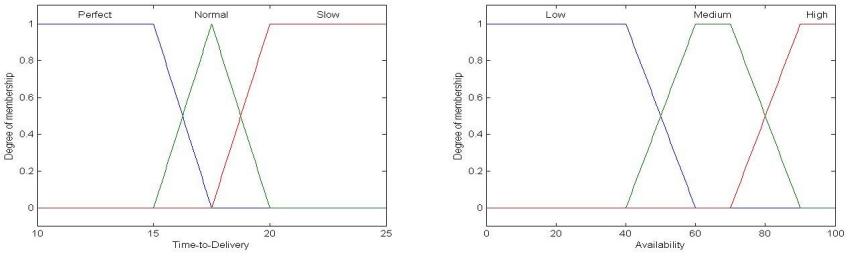
Our knowledge system includes linguistic variables defined by tuple $(\mathcal{Q}, \mathcal{C}, \mathcal{H}, \mathcal{P})$, where $\mathcal{Q}$ is a set of QoS parameters defined by fuzzy parameters as $\mathcal{Q} = \{t_d, q_d, a_d\}$ where t_d is the time to delivery, q_d is the quality of the delivered service and a_d is the availability of the delivery service. $\mathcal{C}$ is the current state of the customer, $\mathcal{H}$ is the history of the service to show whether the service is penalized previously and $\mathcal{P}$ is the set of penalties. We define these linguistic variables by fuzzy sets in the following.

The linguistic parameter of customer is defined by three fuzzy sets as in $\mathcal{C} = \{Normal, Silver, Gold\}$. We define two fuzzy sets to represent the state of service with respect to previous penalties as in $\mathcal{H} = \{Penalized, Not - penalized\}$. Finally penalties are described by five fuzzy sets to show the diverse range of penalties as in $\mathcal{P} = \{Null, Minor, Average, Major, Termination\}$, where *null* is no penalty, and *termination* is the situation in which the customer will terminate his contract with the delivery service. A fuzzy set represents the degree to which an element belongs to a set and it is characterized by membership function $\mu_{\tilde{A}}(x) : X \mapsto [0, 1]$. A fuzzy set $\tilde{A}$ in X is defined as a set of ordered pairs

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X, \mu_{\tilde{A}}(x) \in [0, 1]\} \tag{1}$$

where $\mu_{\tilde{A}}(x)$ is the membership function of x in $\tilde{A}$. Therefore, a membership function shows the degree of affiliation of each parameter by mapping its values to a membership value between 0 and 1.

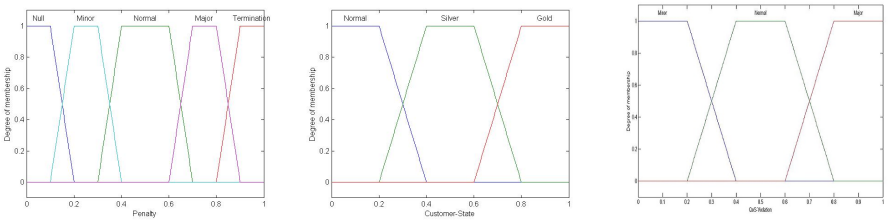
We associate membership functions to a given fuzzy set to define the appropriate membership value of linguistic variables. We start by providing membership functions for quality parameters from the motivating example. We take an approach that calculate an overall degree of violation with respect to the violation of each quality parameters. This way, we perform a trade-off mechanism and quality parameters are not treated independently. For each quality parameter a membership function is provided to show the degree of their satisfaction. We define three linguistic variables for each parameters such that t_d belongs to the set $\{Slow, Normal, Perfect\}$ and q_d is in the set $\{Unacceptable, Bad, Good\}$ and a_d is in the set $\{Low, Medium, High\}$. Figure 1 depicts the membership



(a) Time-to-Delivery membership function (b) Availability membership function

Fig. 1: Membership function for quality parameters

functions of time to delivery (a) and service availability (b). The functions are defined according to the contract and by an expert of the system. For example, the time to delivery between 10 to 15 min is *perfect*, between 15 to 20 min is *good* and more than 20 min is *slow*. Membership functions of penalty and customer state are shown in Figure 2 in (2a) and (2b) respectively.



(a) Penalty mf (b) Customer-state mf (c) QoS-violation mf

Fig. 2: Membership function for penalty ,state of the customer and QoS violation

4.2 Inference rules on penalties

The inference rules to trigger penalties are expressed as follows:

– $R_{\mathcal{Q}}$: QoS-based penalty rules. These are rules that reflect the violation of quality parameters. Penalties will be applied to a service if QoS guarantees stipulated in SLA are not fulfilled. It will be presented formally by $R_{\mathcal{Q}} : \mathcal{Q} \rightarrow \mathcal{P}$. For instance, in the SLA, the delivery service agreed with the customer: $10mns \leq$ *delivery time* $\leq 15mns$ and *good* quality of delivered food. If the QoS delivery time is not fulfilled (partially), then penalty p_{e1} (e.g. 10% discount) will be applied. Depending on the severity of the violation a *harder* penalty might be applied. For example, if both QoS are not fulfilled then penalty p_{e2} (e.g. 20% discount) will be applied. The fuzzy inference system gives us such degrees for penalties. Both cases are presented respectively below by rules:

  - R1 $(t_d = Slow) \wedge (q_d = Good) \rightarrow p_{e1}$
  - R2 $(t_d = Slow) \wedge (q_d = Bad) \rightarrow p_{e2}$
- $R_{\mathcal{P}}$: penalty on penalty rules. These rules reflect whether the service was given a penalty. If a service was penalized previously and again does not fulfil a QoS, then a penalty will be *harder*. It will be presented formally by $R_{\mathcal{P}} : \mathcal{Q} \times \mathcal{P} \rightarrow \mathcal{P}$ such that $R_{\mathcal{P}}(q, p_1) = p_2 \Rightarrow p_1 \prec p_2$.
  For instance, let us consider a service having a penalty p_{e1} w.r.t rule $R1$ and again provides a slow delivery time, then the penalty p_{e3} (e.g. 10% discount plus free delivery) will be applied. The rule can be presented as below:
  - R3 $(t_d = Slow) \wedge p_{e1} \rightarrow p_{e3}$
- $R_{\mathcal{C}}$: customer-related penalty rules. The rules defined here will be adapted according to a customer qualification. Such rules will be presented formally by $R_{\mathcal{C}} : \mathcal{Q} \times \mathcal{P} \times \mathcal{C} \rightarrow \mathcal{P}$.
  For instance, if the provided QoS is not fulfilled knowing that a penalty is assigned to the service, and if a customer is gold (has a good history), then extra service penalty p_{e4} (giving some extra service to the gold customer e.g. one movie ticket) will be harder than the one applied for normal customer p_{e3}. The rules can be presented as below:
  - R4 $(t_d = Slow) \wedge p_{e1} \wedge (C = Normal) \rightarrow p_{e3}$
  - R5 $(t_d = Slow) \wedge p_{e1} \wedge (C = Gold) \rightarrow p_{e4}$

5 Experiments and Implementation

We have simulated our approach in a simulator based on fuzzy inference system. Initial membership functions were designed based on the contract in the motivating example and fuzzy rules are defined by the expert of the system. Figure 2c illustrates membership function for QoS violation (see [3] for further details). Having defined the QoS violation, we measure the extent of penalties taken into account the state of customers and previously applied penalties for the same service. For this, fuzzy rules are defined considering all three influencing factors. Figure 3 depicts fuzzy rules for penalty based on QoS violations, customer's state and service status with respect to previous penalties which are defined by the *service-state* parameter represented by fuzzy set $\{Penalized, Not-penalized\}$.

For example rule no. 8 shows that a major penalty will be given to a silver customer if major violation occurs from defined QoS, while rule no. 7 will give a normal penalty (has lesser effect than major penalties) to the normal customer when the same amount of violation happens. The role of service-state can be seen in the rule, e.g. by comparing the rule no. 5 with the rule no. 14. In general, a harder penalty will be given to the service which is already penalized from the provider side.

The inference system calculates the degree of penalty by applying all the rules in a parallel approach for given input values of influencing factors. For example assume a QoS violation of 0.7 which has a membership degree of 0.5 for both *normal* and *major* fuzzy sets (according to their membership functions

5. If (QoS-Violation is Average) and (Customer-State is Silver) and (Service-State is Not-penalized) then (Penalty is Normal) (1)
6. If (QoS-Violation is Average) and (Customer-State is Gold) and (Service-State is Not-penalized) then (Penalty is Major) (1)
7. If (QoS-Violation is Major) and (Customer-State is Normal) and (Service-State is Not-penalized) then (Penalty is Normal) (1)
8. If (QoS-Violation is Major) and (Customer-State is Silver) and (Service-State is Not-penalized) then (Penalty is Major) (1)
9. If (QoS-Violation is Major) and (Customer-State is Gold) and (Service-State is Not-penalized) then (Penalty is Termination) (1)
10. If (QoS-Violation is Minor) and (Customer-State is Normal) and (Service-State is Penalized) then (Penalty is Minor) (1)
11. If (QoS-Violation is Minor) and (Customer-State is Silver) and (Service-State is Penalized) then (Penalty is Normal) (1)
12. If (QoS-Violation is Minor) and (Customer-State is Gold) and (Service-State is Penalized) then (Penalty is Major) (1)
13. If (QoS-Violation is Average) and (Customer-State is Normal) and (Service-State is Penalized) then (Penalty is Normal) (1)
14. If (QoS-Violation is Average) and (Customer-State is Silver) and (Service-State is Penalized) then (Penalty is Major) (1)
15. If (QoS-Violation is Average) and (Customer-State is Gold) and (Service-State is Penalized) then (Penalty is Termination) (1)
16. If (QoS-Violation is Major) and (Customer-State is Normal) and (Service-State is Penalized) then (Penalty is Major) (1)
17. If (QoS-Violation is Major) and (Customer-State is Silver) and (Service-State is Penalized) then (Penalty is Termination) (1)
18. If (QoS-Violation is Major) and (Customer-State is Gold) and (Service-State is Penalized) then (Penalty is Termination) (1)

Fig. 3: Fuzzy rules for penalty based on QoS violations, customer's state and previous penalties on the service

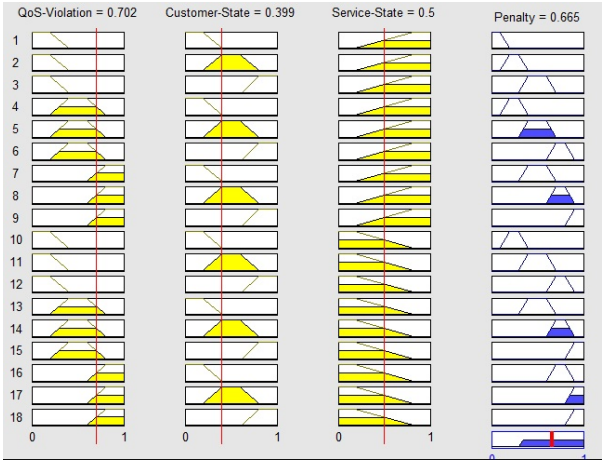


Fig. 4: A view of the inference system for applying penalties

presented in the figure 2c). Such a violation, can trigger all the rules that include normal and major QoS violations. Note that the result of each rule depends on the membership degrees of other linguistic variable. For this example, rules with *minor* QoS-violation are not triggered at all. This situation is demonstrated in Figure 4. The result of each rule is integrated with an aggregation method to include the effect of all the rules. Figure 5 depicts a plot showing the penalties regarding QoS violation and customer's state. The figure represents possible values for penalties after defuzzification for all values of QoS violation and customer's state. For example, for the QoS violation of 0.7 and customer-state of 0.4 the penalty degree is 0.66 which is shown in the figure. The relation between QoS violation and customer's state can also be seen in the figure.

6 Conclusions and Future Work

Applying penalties is a complex research issue in service oriented computing which has not been paid enough attention in the literature. In this work, we

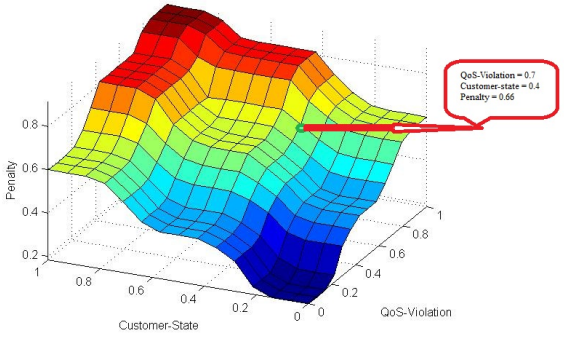


Fig. 5: The plot showing the penalties regarding QoS violation and customer's state

elaborated the concept of penalty and propose a mechanism for modelling and measuring penalties. Penalties are modelled using a fuzzy approach and applying fuzzy set theory. The relation between penalties and their influencing factor are defined by fuzzy rules through an inference method. We have demonstrated the proposed penalty model through a motivating example and performed some initial result in measuring penalties.

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).

References

1. A. Andrieux *et al.* Web Services Agreement Specification (WS-Agreement). Recommended standard, Open Grid Forum, March 2007.
2. Pei Li, Marco Comerio, Andrea Maurino, and Flavio De Paoli. Advanced non-functional property evaluation of web services.
3. Barbara Pernici and Seyed Hossein Siadat. A fuzzy service adaptation based on QoS satisfaction. In *CAiSE'11*, pages 48–61, 2011.
4. Omer Rana, Martijn Warnier, Thomas B. Quillinan, Frances Brazier, and Dana Cojocarasu. Managing violations in service level agreements, 2008.
5. Lotfali Askar Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
6. Mohamed Anis Zemni, Salima Benbernou, and Manuel Carro. A soft constraint-based approach to qos-aware service selection. In *ICSOC*, pages 596–602, 2010.

Appendix I

Design for Self-adaptation in Service-oriented Systems in the Cloud

CHAPTER EIGHT

DESIGN FOR SELF-ADAPTATION IN SERVICE-ORIENTED SYSTEMS IN THE CLOUD

ANTONIO BUCCHIARONE, CINZIA CAPPIELLO, ELISABETTA DI NITTO, SERGEI GORLATCH, DOMINIQUE MEILÄNDER, ANDREAS METZGER

Service-oriented systems are able to offer complex and flexible functionalities in widely distributed environments by composing different types of services. These systems have to be adaptable to unforeseen changes in the functionality offered by component services and to their unavailability or decreasing performances. Furthermore, when systems are made available to a high number of potential users, they should also be able to dynamically adapt to the current context of use as well as to specific requirements and needs of the specific users. In order to address these issues, mechanisms that enable adaptation should be introduced in the life-cycle of systems, both in the design and in the runtime phases.

In this chapter we will go through the life-cycle of a service-oriented system highlighting those activities that are needed to support adaptation. The adaptation activities can be performed at various layers of the service-oriented system. In particular, they can concern the layer where services are composed together or the layer of the executing infrastructure, typically, a cloud system. To exemplify the various steps and activities we use an example from the domain of real-time online interactive applications.

1. Introduction

Modern software technology has enabled us to build software systems with a high degree of flexibility. The most important development in this direction is the concept of *service* and the *Service-oriented Architecture (SOA)* (Josuttis 2007). A *service-oriented system* is built by composing software services (and is thus also called *service composition* or *composed service* in the literature).

Software services achieve a high degree of flexibility by separating 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 from remote through the service's interface. Ownership, maintenance and operation of the software remain with the service provider (Di Nitto et al. 2008).

One key principle of SOA is *loose coupling*, which means that a service only makes weak assumptions about its interactions with other services; e.g., instead of services being tightly coupled by means of a common data model, only simple data types are used. This principle allows a service to be (re-)used in many different service compositions. Another key principle of SOA is *late binding*. This implies that many services will be discovered and composed into a service-oriented system only at run-time. Service-oriented systems thus provide their functionality rather dynamically and in a loose fashion.

As recent studies reveal (Zheng et al. 2010), such distributed applications are extremely fragile, due to the execution of uncontrollable third-party services. Unexpected changes of third-party services or unpredicted network latencies, for example, can cause failures which negatively impact on the timely execution of the applications. To prevent such failures, self-adaptation capabilities are needed.

Self-adaptation is usually referred to as the ability of a system to autonomously address changes during its operation. The activities performed during adaptation typically follow the steps of the MAPE loop, which stands for Monitor-Analyze-Plan-Execute (Salehie et al. 2009).

Service-oriented systems operate on different layers to implement their functionality and to provide the expected quality. In a simplified view, we can identify two main layers, the one of the computational and storage resources used for executing the system (this is often called the *infrastructure layer*) and the *composition layer*, in which the services offered by the system are made available and composed together. The infrastructure layer can be organized in various ways and encompasses the

boundaries of various organizations. In the case when services are offered by various third parties we can have different cloud systems, either private or public, making available their resources to the system.

In this two-layers setting, self-adaptation can happen both at the infrastructure and at the service composition layer, depending on the nature of the situation to be accommodated. If, for instance, a failure of a computational resource occurs at the infrastructure layer it is quite natural to design such layer so that it is able to replace the failing resource with another. Similarly, if the failure concerns a specific service operating as part of the system, at the service composition layer, it is possible to replace the failing service with another without making the infrastructure layer aware of this change.

While in many cases it is possible to isolate self-adaptation in a single layer, there are cases in which this is not possible. In particular, adaptations on different layers might impact on each other and may even be conflicting. As an example, if an adaptation on the infrastructure layer leads to the migration of a service from one cloud to another, this could be conflicting with the decision on the service composition layer to choose a third-party service instead of the "own" service implementation deployed in the Cloud. In addition, there may be cases in which the joint adaptation on both layers provides opportunities to address problems, which isolated applications on single layers will not be able to remedy.

The need for scrutinizing such cross-layer adaptations has been elaborated in S-Cube, the EU Network of Excellence on Software Services and Systems (Papazoglou et al. 2010). Additionally, a recent survey amongst professionals from the Future Internet community, which has been performed during the Future Internet Assembly (FIA 2011) in Budapest, has confirmed the importance of such cross-layer adaptations (Metzger et al. 2011).

In this chapter we focus on two main aspects, that is, the kinds of changes that trigger self-adaptation in a service-oriented system and the strategies that can be adopted to deal with adaptation. We do so by highlighting the fact that some adaptation can be performed at the infrastructure layer while others have to be handled at the composition layer, and we will scrutinize design activities and decisions for applications that will be deployed in the cloud as the infrastructural target. We also provide a preliminary contribution to the systematic understanding of adaptation across layers.

To exemplify the approach presented in the chapter we refer to a specific system example focusing on the domain of Real-Time Online Interactive Applications (ROIA). This system has been developed by

exploiting a service-oriented approach and has been featured with a software component that deploys and manages it on the cloud. Among the other things, this component is particularly interesting for us as it is able to execute some cross-layer adaptations.

2. Kinds of self-adaptation needs

Self-adaptation of a service-oriented system is triggered by changes that may occur in three major areas: (1) the expectations that its users (or other stakeholders) have concerning some quality that the system should provide, (2) the world in which the system is executed, (3) the system itself.

The first area is the realm of *non-functional requirements*, the second one is traditionally called *context*, and the third one is often referred to as the *machine* (Jackson 2001).

The reader can notice that there is another important area where changes occur frequently and that demands for changes in software systems. This is the area of functional requirements. We do not consider it in this chapter as this concern cannot be addressed with self-adaptation, but, instead, requires software systems to undergo through a re-development cycle (often called evolution).

In general, all changes that can be addressed by performing some simple reasoning (e.g., "since the user is in downtown Milano, he/she is certainly interested in knowing about free parking spaces in that area"), or by modifying the bindings to services, can be addressed by a service-oriented system if it incorporates self-adaptation facilities (de Lemos et al. 2011). Vice versa, the changes that require redesign or reimplementation of the system typically have to be addressed by the intervention of human beings and are not considered in this chapter.

The area of non-functional requirements refers to Qualities of Service (QoS) such as the service response time or its availability. During the execution of a service it can happen that the expectation of users for a certain non-functional requirement, e.g., response time, changes. This change may be due to various reasons, for instance, to a new regulation that requires users to perform some operation in a shorter time. The change can be, to a certain extent, accommodated through the self-adaptation capabilities of the system, taking into account also its execution conditions and its ability to cope with an increase of the response speed for a certain user.

The term context derives from the Latin *cum* (with or together) and *texere* (to weave). It has been defined by Dey and Abowd (Dey and Abowd 2000) as "'any information that can be used to characterize the situation of entities (persons, places, objects) that are relevant to the interaction between a user and an application, including the user and the application themselves". According to Hofer et al. (Hofer et al. 2002), context can be *physical*, i.e., measured by some hardware sensor or *logical*, i.e., captured by monitoring user interaction. When it is physical it refers to location, movement, and any environmental information. When it is logical it refers to users' goals, emotional state, business processes, etc. In both cases, the context evolves during the execution of the service-oriented system. For instance, the user may change location, or the business processes in which he/she is involved can change. Self-adaptation capabilities of the system need, therefore, to be able to sense and to reason about these changes.

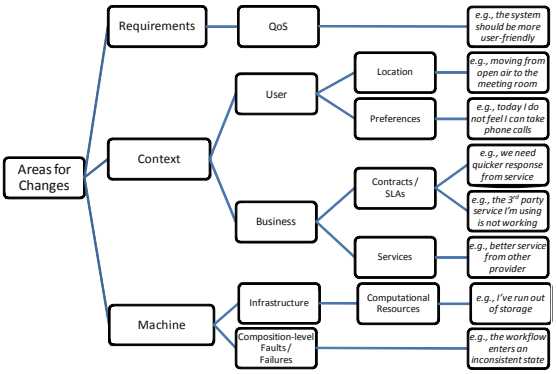


Fig. 8-1. Areas for changes and examples

Both changes in the non-functional requirements and in the context happen in the external environment that, therefore, has to be continuously monitored by the system. Some changes happen also internally to the system, both in its hardware and software resources. An example concerns the increase/decrease of the computational power available for the software system.

Other examples of changes that may occur and trigger the need for adapting the service-oriented system are shown in Figure 8-1.

Clearly, the examples of changes shown in the figure require different levels of intervention on the corresponding software system and its resources.

For instance, the lack of computational resources may be addressed at the infrastructure layer, without modifying the structure of the service-oriented system, e.g., by exploiting the flexibility offered by Cloud computing (Armbrust et al. 2010). The failure of a service is addressed at the service-composition layer through dynamic binding of alternative and compatible services. In this case, in fact, the substitution of one service for another service can occur at runtime without performing any reprogramming activity of the software (Moser et al. 2008).

A classification of various kinds of adaptation needs can be found in (Metzger et al. 2011). Among others, the authors identify *reactive adaptation*, that is, the ability of the system to react to changes when they occur, and *proactive adaptation*, that is, the ability of the system to anticipate the need for adaptation. Orthogonally to this classification, the authors also comment on the importance of managing adaptation across all layers of a software system. We also highlight that self-adaptation that copes with problems within the machine can be classified as *self-healing*, i.e., the ability to repair the system or *self-optimization*, i.e., the ability of improving the performances of the system.

To enable adaptation needs to trigger self-adaptation, the service-oriented system has to be properly built and deployed. At design time the mechanisms that enable self-adaptation have to be prepared. At runtime a monitoring infrastructure has to ensure that the adaptation needs are captured and that proper adaptation strategies are executed. The life cycle presented in the following section focuses on both the design time and the runtime aspects and highlights the main activities and artifacts to be produced in each phase.

3. The S-Cube lifecycle and self-adaptation

As discussed before, at design time, functional and non-functional requirements, context and machine characteristics have to be analyzed in order to identify the types of changes that trigger self-adaptation and consequently define the needed mechanisms to monitor the environment and the system behavior. In this phase, the strategies for self-adaptation have to be also developed. The life cycle proposed in the S-Cube project (see Figure 8-2) addresses all these aspects (Bucchiarone et al., 2009) and (Bucchiarone et al., 2010). Such life cycle is composed of two cycles: (i)

the *evolution* cycle that leads to the explicit re-design of an application, (ii) the *adaptation* cycle that is performed at run time when the application needs to be adapted on-the-fly. The two cycles coexist and support each other during the lifetime of the application.

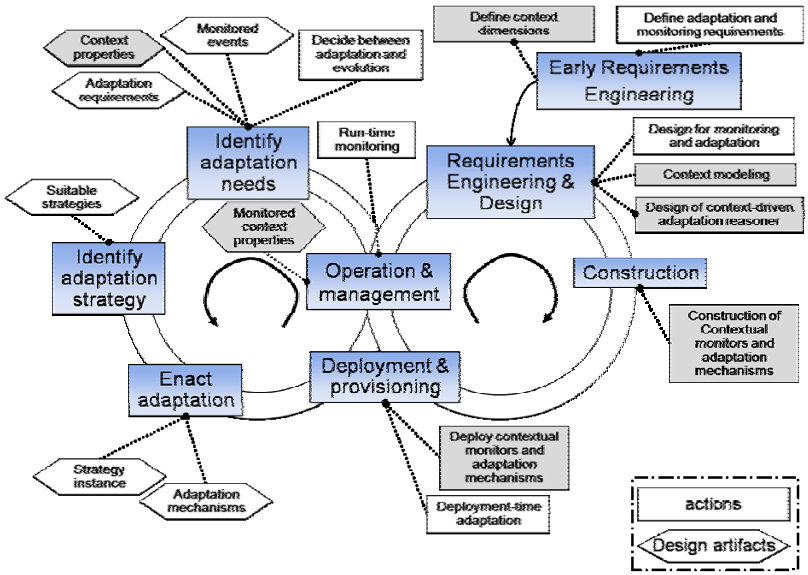


Fig. 8-2. Life-cycle for adaptable service-oriented systems

The initial phase of the life cycle is the *(Early) Requirement Engineering and Design* in which the adaptation and monitoring requirements are elicited on the basis of the context dimensions and the application and system characteristics that are considered relevant for the adaptation and the evolution of the considered system. Thus, this phase is responsible for the definition of the types of changes that trigger self-adaptation. Successively, during the *System Construction* phase, the corresponding monitors and the adaptation mechanisms are designed, developed and then refined until the *Deployment* phase. When the system is running (*Operation and Management* phase), continuous monitoring is executed supporting the detection of the relevant context and system changes. If changes occur, it is possible to proceed in two different directions: executing evolution or adaptation of the system. Evolution is performed if the system needs to be redesigned and thus it requires the reiteration of the described cycle starting from the requirements engineering and design phase.

If changes require adaptation, the *Adaptation Needs* that can be triggered from monitored events and adaptation requirements have to be identified. An adaptation need can be defined as a specific problem-situation that demands for adaptation. Each adaptation need has to be associated with the different A*daptation Strategies* able to satisfy the corresponding adaptation requirements. Finally, on the basis of the current situation, the knowledge obtained from previous executions, and the available adaptation mechanisms, a reasoner selects the most suitable adaptation strategy that will be automatically or manually performed in the *Enactment* phase.

Figure 8-2 highlights the various adaptation- and monitoring-specific actions (boxes) carried out throughout the life cycle of a service-oriented system, the main design artifacts that are exploited to perform adaptation (hexagons), and the phases where they are used (dotted lines).

While the life cycle can be applied when reasoning at different layers of a service-oriented system, self-adaptation triggers and strategies relevant for each layer are different. In the following two sections we discuss about those that are relevant to the composition layer and to the infrastructure layer. Moreover, we discuss about the interplay of adaptation actions performed at these two layers.

4. Possible adaptation triggers and actions at the composition layer

In order to offer efficient and reliable service-oriented systems, it is necessary to guarantee that the service components are always aligned with the changing world around them. As we have seen in the previous section, at design time we identify the possible needs that trigger adaptation, and that from now on we call *triggers*. Also, we define the possible alternative strategies to support adaptation.

To do this, we need to have a concrete model of the service composition context and of those aspects of the requirements and of the machine that are relevant for triggering the adaptation or evolution of a service composition, and that enable the identification and the collection of the proper context information. As we have seen in Figure 8-2, starting from the requirements analysis phase where, in parallel to the precise definition of requirements, a proper *context model* has to be defined.

Once the context model and its relation with the application model are defined, it is necessary to properly capture and define the adaptation aspects. In particular, it is important to define when the contextual changes

are critical for the service composition functioning (i.e., adaptation triggers) and what should be done or achieved when these changes take place (adaptation strategies).

Depending on the context dimension/element and on the specific requirements of a service composition, the definition of adaptation trigger may vary. As a result, the corresponding context monitors needed to detect those trigger will have different, sometimes application-specific, forms and realizations.

Furthermore, in certain cases the adaptation trigger does not correspond to some value of a particular single context element, but is characterized by the complex combination of different context dimensions/factors. In the simplest case, such situations may be directly encoded using the capabilities provided by existing monitoring frameworks as Dynamo (Baresi et al. 2005). In other cases, a sophisticated reasoner may be necessary. For instance, in (Kazhamiakin et al 2008), a context represents the combination of users personal assets (agenda, location, social relations, and money), and the critical context changes characterize some critical combination of those assets. A dedicated analysis mechanism is used to identify those situations (asset conflicts) and to trigger adaptation solutions.

As for triggers associated to the component services, considering the context, we can define adaptation actions to align the service composition with its context changes. Considering the contextual aspects, it is necessary to take into account that adaptations may be performed (i) to *customize* the composition in order to fit to the situation in which it currently operates, (ii) to *optimize* the composition in order to improve certain (usually QoS) issues and characteristics of the composition, or (iii) to *prevent and avoid* future faults or undesirable situations in the execution of the system. The customization is often driven by changes in the composition context and especially by time, ambient, user, business, and service context dimensions. Optimization is more related to the service and computing context dimensions but can be also triggered by changes in the users' preferences. Finally prevention is mostly related to the service and computing context since changes of the execution environment might increase the risk of failures and the need for prevention. Some of these aspects may be also interleaved. For example, if a user moves to a new location (i.e., change in the user context), new set of services may be available (i.e., change in the business context) with different bandwidth (i.e., change in the machine).

As represented in Table 8-1, examples of triggers relevant for the composition layer include changes in the time in which the system

operates, in the ambient of the user, in the user preferences (indicated in the table simply as "User"), in the services being exploited, in the business processes and in the machine. Possible adaptation strategies include the possibility to replace one or more services with some others, the re-execution of some services, the transformation of the composition itself, e.g., by introducing new paths of service calls, the complete failure of the composition, the possibility to keep the service bindings abstract till runtime, the possibility to renegotiate some Service Level Agreement (SLA), the possibility to compensate failing transactions, and the possibility to leave the adaptation loop to enter into the evolution loop.

Table 8-1. Suitability of adaptation strategies to react to context changes

Adaptation Strategy	Adaptation Triggers					
	Time	Ambient	User	Service	Machine	Business
Service substitution	X	X	X	X	X	X
Re-execution				X	X	
Re-composition				X	X	X
Fail			X	X	X	X
Service concretization	X	X	X	X	X	X
Re-negotiation			X		X	X
Compensation				X	X	
Trigger Evolution		X	X	X	X	X

As shown in Table 8-1, each trigger can be associated with a set of adaptation strategies that are suitable to re-align the application within the system and/or context requirements. In order to select the adaptation strategy to apply, it is necessary to consider that adaptation triggers may be associated with other requirements that are important for designing and performing adaptation, in particular: the *scope of the change*, i.e., whether the change affects only a single running instance of the service composition or influences the whole model and, the *impact of the change*, i.e., the possibility of the application to accomplish its current task. Depending on these parameters different strategies may apply. For example, when the scope of the change concerns the whole application model "trigger evolution" strategy applies. As for the impact, such strategies as "re-execution" or "substitution" may apply when the service composition state did not change and the task still can be accomplished.

On the other hand, “compensation”, “fail”, or “trigger evolution” apply when there is no way to complete the current task.

While the life cycle we have introduced so far can be applicable to both the composition and the infrastructure layer, the kinds of adaptation triggers as well as the corresponding strategies offered by the infrastructure layer are mainly focused on what is happening within the machine.

5. Possible adaptation triggers and actions at the infrastructure layer

More in detail, adaptation at the infrastructure layer is often triggered by variations in the load of the system (either measured in terms of number of requests per time unit or in terms of CPU cycles needed to complete the jobs in the system queues). In these cases, the infrastructure layer may decide to replicate the system on new machines, for instance, by creating clones of the currently used virtual machines (VMs from now on) or to balance the load among the existing virtual machines.

Another relevant trigger for adaptation concerns the availability of new resources that were not accessible before. In this case, the infrastructure layer may decide to use them for replacing the available ones, if they are more convenient, or to use them together with the existing ones.

Some triggers can also be external to the machine. An example is a change in the QoS users require. Such a change can be accommodated directly by the infrastructure layer in case it enables an elastic use of resources.

Table 8-2 shows some possible triggers and strategies. The ‘X’ symbols highlight the suitability of a certain strategy for a specific trigger. In many cases different strategies have to be combined together. For instance, when the load is too high for the system, new machines can be started, but this action alone does not solve the problem. We have also to move or clone some VMs on the newly started machines in order to exploit their capabilities.

Table 8-2. Adaptation triggers and strategies at the infrastructure layer

Adaptation strategy	Adaptation trigger				
	Load is too low / too high	New computing resources are available	One or more computing resources become unavailable	A new cloud becomes more convenient	QoS requirements change
Clone a VM	X				X
Move a VM		X	X	X	
Balance load among VMs	X				
Switch machines on/off	X				X
Move data storage			X	X	
Migrate from a cloud to the other	X	X		X	X

6. Coordinating cross-layer adaptation

As introduced above, service-oriented systems operate on different layers to implement their functionality and to provide their expected quality. We have identified and discussed two main layers: (1) the *infrastructure layer*, on which computational and storage resources used for executing the system reside (see the Section on possible adaptation triggers and actions at the infrastructure layer); (2) the *composition layer*, in which the services offered by the system are made available and composed (see the Section on possible adaptation triggers and actions at the composition layer). In this two-layers setting, self-adaptation can happen both at the infrastructure and at the service composition layer.

However, if adaptations on those layers are not coordinated, this can lead to the following two important issues: First, conflicting adaptations can arise, e.g., in case where the individual adaptations cancel each other or interact in an undesired way. Secondly, there may be cases in which isolated applications on single layers will not be enough to resolve the

problem; e.g., a problem can only be resolved if the composition of the system is changed while at the same time more computing resources are made available in the cloud.

We elaborate on both of these issues in the following sub-sections.

6.1 Avoiding Conflicting Adaptations

In (Bucchiarone et al 2010b) a general framework has been proposed to address the harmful side-effects resulting from independently acting adaptations. In (Kazhamiakin et al 2008b) general, adaptation-specific failures and possible quality assurance techniques to uncover those failures have been introduced. Finally, in (Guinea et al 2011), a more concrete framework that integrates layer specific monitoring and adaptation techniques, and enables multi-layered control loops has been proposed for service-based systems. All these contributions lend themselves naturally to discuss the problems that may arise when considering adaptations which are independently performed on different *layers*.

The following important classes of problems can arise in the presence of independently acting adaptations:

- **Conflicts:** A conflict arises if each layer reacts to a relevant change by issuing contradictory adaptations concurrently. Contradictory adaptations are adaptations that cannot be applied in combination since they would violate the requirements of the system or would not be able to solve the observed deviation from requirements. In certain situations the timing of events may be the source of another such kind of conflict. For example, it might happen that changes occurring on the infrastructure layer are so fast, that although they can be addressed efficiently on that layer, the impact they have on the composition layer leads to severe problems, as the execution of adaptation activities on the composition layer is not quick enough to follow. This may mean that new events continuously trigger new adaptations, leading to an "adaptation stack overflow", possibly delaying other important adaptations or bringing the application to a halt completely.

- **Oscillations:** An oscillation can occur if conflicting adaptations are issued in sequence. For example, the composition layer may receive an event that triggers an adaptation which in turn leads to a critical change that leads to an adaptation on the infrastructure layer. It might happen that the adaptation of the infrastructure layer in turn leads to

critical event on the composition layer, necessitating yet another adaptation. Obviously, if this undesired behaviour is not observed and controlled, this can lead to a long series of mutual adaptations without ever reaching a stable system configuration, i.e., this can lead to an "adaptation livelock".

- **Race conditions:** Race conditions occur if the two layers apply adaptations and the final outcome depends on the order in which these adaptations are performed and completed. As an example, an adaptation on the composition layer may require the deployment of a new service on the cloud. Yet, if the adaptation of the cloud needed to provide additional resources for that new service is executed after the deployment of the new service, it might happen that the deployment will not succeed due to limited computational resources. Yet, if the resources would have been reserved before deployment, this adaptation would have been successful.

There are basically two complementary ways of avoiding the above problems: (1) one can check whether those problems may actually occur by using targeted quality assurance techniques as proposed by (Cheng et al. 2009, Kazhamiakin et al 2008b); (2) one can restrict the freedom of each individual adaptation by coordinating their adaptation actions as proposed by (Guinea et al. 2011, Zengin et al. 2011, Bucchiarone et al 2010b, Popescu et al. 2010, Cheng et al. 2009, Vidackovic et al. 2009).

In the remainder of this chapter, we will follow the approach number 2 and will provide a concrete example for how such a coordination of adaptation actions can be realized.

6.2 Exploiting Synergy of Adaptations

Of course, the problems that can arise due to uncoordinated adaptations are numerous. Thus, one might be tempted to restrict adaptation to one single layer only in order to avoid these problems. However, there may be situations in which only the execution of concurrent adaptations on more than one layer can address the problems faced.

In an example pointed out by (Schmieders et al. 2011) the coordination of adaptations on the infrastructure and the composition layer can increase the chance that performance violations of the application can be avoided. For instance, a slow service on the composition layer may be replaced by a faster alternative service. However, this might not be enough to achieve the expected response time of the application. Thus, additionally an

adaptation on the infrastructure layer may be used to reserve more computing resources, thereby increasing the responsiveness of services. As this example shows, there may be situations in which a coordinated combination of adaptations can lead to more situations in which requirements violations can be avoided.

This chapter will not further scrutinize this specific aspect of cross-layer adaptation. However, we acknowledge the fact that understanding how to perform such kinds of adaptation should certainly be part of future research.

7. Use case

In this section, we describe how self-adaptation can be implemented in the life-cycle of challenging Real-Time Online Interactive Applications (ROIA) in Cloud environments. Examples of ROIA are multi-player online computer games, interactive e-learning and training applications and high-performance simulations in virtual environments. We illustrate the specific features of ROIA and express their major design and execution aspects in the context of the S-Cube Lifecycle model. Our use case focuses on the execution of ROIA on computational resources and, thus, illustrates adaptation scenarios applied mostly at the infrastructure layer, as described in the previous sections.

In ROIA, there are typically multiple users who access a common application state and interact with each other concurrently within one virtual environment. The users connect to the application from different client machines and interact with other users: e.g., in online games users move their avatars through a virtual environment and interact with other users' avatars or computer-controlled characters (*entities*). Since ROIA usually have very high performance requirements, the application processing is performed on multiple servers. Hence, ROIA are highly distributed applications with challenging QoS demands, such as: short response time to user actions (about 0.1-1.5 s), high update rate of the application (up to 50 Hz), large and frequently changing number of users in a single application instance (up to 10^4 simultaneously). To address these demands, application developers need to implement suitable adaptation mechanisms. In our previous work we study how the Lifecycle model and self-adaptation are applied to the challenging domain of ROIA (Meiländer et al. 2010).

A major problem for an efficient ROIA execution is the economical utilization of server resources, which is difficult due to a variable and continuously changing number of users. This leads to expensive up-front

investments to build a suitable server pool which is able to handle peak user numbers but will be underutilized most of the time when the load is below the peak.

Cloud Computing offers new opportunities for ROIA execution and promises a potentially unlimited scalability by distributing application processing on an arbitrary number of resources given suitable adaptation mechanisms. Clouds allow for adding/removing resources on demand. This opens for ROIA an opportunity to serve very high numbers of users and still comply with QoS demands. Despite a variable number of users, Cloud resources can be used efficiently if the application supports adding/removing resources during run-time. Hence, using Cloud Computing for resource provision and the Lifecycle model for implementing adaptable ROIA complement each other.

In order to support ROIA development and adaptation on Clouds, we develop the RTF-RMS resource management system (Meiländer et al. 2011) on top of the Real-Time Framework (RTF) (RTF 2011). RTF-RMS implements the following mechanisms for ROIA development on Clouds:

- *Monitoring* of application-specific data, e.g., update rate, number of entities, etc.
- *Distribution handling* for the dynamic adaptation of running application sessions by adding/removing Cloud resources on demand using particular adaptation strategies (described below).
- *Application profiles* that allow developers to specify application-specific adaptation triggers.
- High-level *development support* for communication handling and application state distribution.

In Figure 8-3, we illustrate how RTF-RMS supports the developer in designing adaptable ROIA according to the different phases of the Lifecycle model; the phases are numbered in the figure.

In the "*Requirement Engineering*" phase of the Lifecycle model, the application developer must identify suitable adaptation requirements for his application. In the case of ROIA, the mechanisms for monitoring and adaptation should be non-intrusive, i.e., take place in parallel with the application execution, such that users are not aware of changes inside the application.

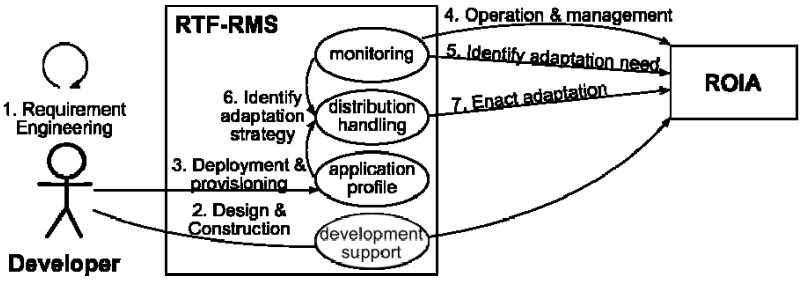


Fig. 8-3. Mechanisms of RTF-RMS for ROIA development according to the Lifecycle model

For the "*Construction*" phase, RTF-RMS provides the developer with a C++ library of high-level functions for optimized communication handling (client-server and inter-server) and efficient application state distribution in a multi-server environment. By using RTF-RMS for communication and distribution handling, monitoring is automatically integrated in the application processing. Monitoring data is used in the next phase of the Lifecycle to implement adaptation triggers, as described in the previous sections. For ROIA, we distinguish between the following adaptation triggers:

- Change in QoS, e.g., caused by unreliable hoster resources;
- Change in the machine, e.g., caused by increasing user interactions, making computation of state updates more expensive;
- Change in the business context, e.g., more users connect to the application due to changing user preferences.

In the "*Deployment and provisioning*" phase, trigger rules are defined in the RTF-RMS *application profile* for each adaptation trigger, see Figure 8-4 for an example. In the application profile, the developer specifies thresholds for the monitoring data provided by RTF-RMS. Adaptation is triggered if corresponding monitoring values pass these thresholds. In the example, adaptation is triggered if the update rate drops below 25 Hz or exceeds 50 Hz. The application developer can find suitable values for all parameters in the application profile by considering the runtime behaviour of the application on physical resources and calculating the virtualization overhead, or by conducting benchmark experiments in a Cloud environment.

```
<appProfileData>
  <metric>
    <name>UpdateRate</name>
    <addResourceThreshold>25</addResourceThreshold>
    <removeResourceThreshold>50</removeResourceThreshold>
  </metric>
</appProfileData>
```

Fig. 8-4: Example application profile for a fast-paced action game.

In the "*Operation and management*" phase, the application is running and monitoring data are checked continuously against the trigger rules to detect changes in the context or in the system that could require adaptation.

In the "*Identify adaptation need*" phase, RTF-RMS detects a violation of trigger rules which indicates, e.g., an overload of the application session: the update rate drops below 25 Hz.

In the "*Identify adaptation strategy*" phase, RTF-RMS analyzes the number of application servers and their current workload to choose an adaptation strategy (Meiländer et al., 2011; the cited paper refers to adaptation strategies as load-balancing actions). The adaptation strategies are based on the partitioning of the virtual environment among servers. In particular, the processing of entities inside a particular area (zone) is assigned to different servers. RTF-RMS implements currently the following three adaptation strategies:

- *User migration:* Users are migrated from the overloaded server to an underutilized server that is processing the same zone. For this purpose, user connections are switched from one server to another. RTF-RMS distributes users by default equally between the application server for a particular zone. User migration is the preferred action if the load of an overloaded server can be compensated by currently running resources.
- *Replication enactment:* New application servers are added in order to provide more computation power to the highly frequented zone. This strategy is called replication: each application server keeps a complete copy of the application state, but each server is responsible for computing a disjoint subset of entities. After enacting replication, RTF-RMS migrates a number of users to the new replica in order to balance the load. Replication implies an additional inter-server communication, so its scalability is limited.

If the number of active replicas for a particular zone is below the maximum number of replicas specified by the application profile, replication is used to add new resources; otherwise the resource substitution strategy (described next) is preferred.

- *Resource substitution:* an existing resource in the application processing is substituted by a more powerful resource in order to increase the computation power for highly frequented zones. For this purpose, RTF-RMS replicates the targeted zone on the new resource and migrates all clients from the substituted server to the new server. The substituted server is then shut down.

In the "*Enact adaptation*" phase, RTF-RMS enacts the chosen adaptation strategy and changes the distribution of the application processing accordingly.

An important factor that may limit the performance of applications on Clouds is the long startup time of Cloud resources which may take up to several minutes. Since ROIA are highly responsive applications, finding a compensation for long startup times is an important task. For this purpose, RTF-RMS implements a resource buffer to which a predefined number of Cloud resources are moved in advance, i.e. before they are demanded by the application. The number of buffered resources is configured in the application profile and has to be chosen carefully since resource buffering generates additional costs. A more detailed discussion on how to choose the size of the resource buffer can be found in (Meiländer et al. 2011).

Another critical issue for the cost-efficient ROIA provision on Clouds is the consideration of leasing periods. In commercial Cloud systems, resources are typically leased and paid per hour or some longer leasing period. Since ROIA have dynamically changing user numbers, the time after which Cloud resources become dispensable is very variable. However, resources will not be used cost-efficiently if they are shut down before the end of their leasing period. Hence, RTF-RMS removes the resources that have become dispensable from the application processing and moves them to the resource buffer. Cloud resources in the buffer are shut down at the end of their leasing period or they are integrated in the application processing again if required.

In order to demonstrate the influence of adaptation using RTF-RMS on the application performance, we present experimental results of the user migration adaptation strategy using an example of a multi-player action game called RTFDemo (RTF, 2011). In order to provide a seamless gaming experience, users should not receive less than 25 updates per second over a longer time period. Hence, we defined an adaptation trigger rule with 25 updates per second as the lower threshold.

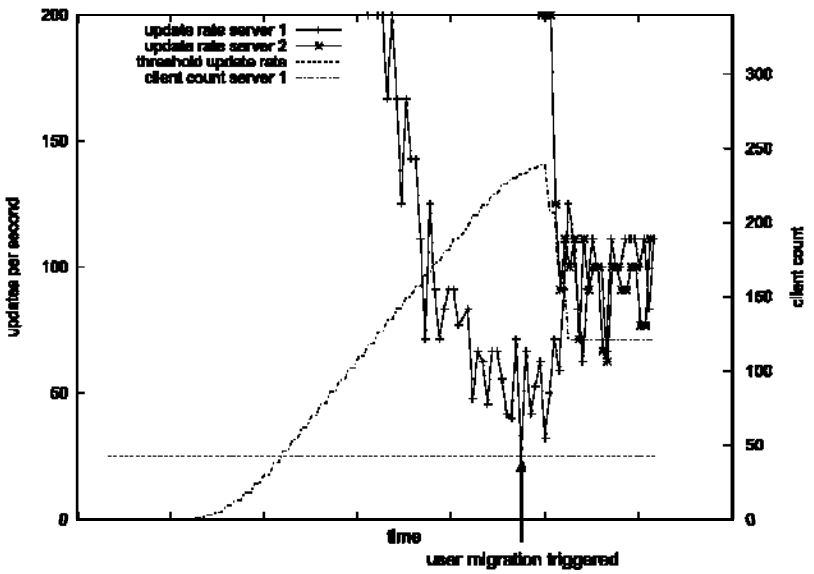


Fig.8-5. Adaptation using user migration.

For our experiments, we use a private Cloud environment with the Eucalyptus framework (version 2.0.2) (Nurmi et al., 2009). We have started two application servers on Cloud resources for replication of a single zone. All clients are initially connected to server 1, which implies an imbalanced load on application servers. We apply adaptation at the infrastructure level by balancing load among virtual machines using user migration. The load is balanced by assigning the processing of half of the users that are connected to server 1 to server 2. The corresponding user connections are switched between these Cloud resources accordingly. Figure 8-5 shows that the update rate of server 1 initially drops with the growing number of connected clients. When the update rate reaches the threshold of 25 Hz for 240 users, RTF-RMS migrates half of the users (i.e., 120 users) from server 1 to server 2. This adaptation strategy increases the update rate of server 1 from 25 Hz to about 100 Hz.

8. Related work

The literature contains several approaches for the definition of a life cycle for the design and management of service-based applications. Earlier approaches such as ASTRO (Trainotti et al. 2005), focused on the possibility to monitor and intervene on SBAs in order to recovery from unwanted and unexpected behavior, assume human interventions. More recent work tried to handle this issue proposing methods for developing self-adaptive applications. As suggested in (Ardagna and Pernici 2007), adaptation mechanisms can either be embedded in the description of the adaptable SBAs or implicit in its structure. Various frameworks can be found in the literature with the objective to support adaptation of SBAs. In (Linner et al. 2008), authors propose a life-cycle supporting self-adaptation of the service-based applications even if they lack of explicit guidelines for their design. Anyway, most of the literature contributions concern built-in adaptation, i.e., the adaptation logic is completely specified at design time. They focus on the specification of adaptation mechanisms and adaptable applications, exploiting different tools. For instance, the specification may be performed by extending standard notations (i.e., BPEL) with adaptation specific tools (Karastoyanova et al., 2005), event conditions actions rules (Baresi et al., 2007), variability modeling (Hallerbach et al, 2008) or aspect-oriented approaches (Kongdenfha et al., 2006).

In the literature there are, however, proposals of frameworks for dynamic adaptation, all featuring an adaptation manager separated from the application. Notably, all these approaches are in the service-oriented field. In (Spanoudakis et al., 2005) the authors consider the problem of adapting the application by replacing malfunctioning services at runtime. In this approach, adaptation strategies are triggered as a consequence of a requirement violation. In fact, the adaptation rule is fixed at design time, but it is dynamically applied by a manager component that monitors functional and non-functional properties, creates queries for discovering malfunctioning services and replaces them with dynamically discovered replacements. In other approaches, adaptation is triggered by application constraints. For example, Narendra et al. (Narendra et al., 2007) propose an aspect-oriented approach for runtime optimization of non-functional QoS measures. QoS constraints are also the basis for dynamic reconfiguration of processes in the METEOR-S framework (Verma et al., 2005). Reconfiguration is performed essentially at deployment-time. Some more general frameworks are also available. An example is SCENE (Colombo et al, 2006) where event-condition-action rules are used for

defining different kinds of self-adaptation rules dealing with runtime replacement of services, service re-execution, renegotiation of SLAs, replacement of parts of a service composition, etc. As in other cases, in this approach rules are triggered during the execution as a result of a monitoring activity and they can trigger various dynamic actions that change the service composition depending on many variables, including the status of the context. While a set of basic rules is defined at design time, new rules can also be added during the execution of the system as a result of a learning activity.

PAWS (Ardagna et al., 2007) is a framework for flexible and adaptive execution of web service based applications. At design-time, flexibility is achieved through a number of mechanisms, i.e., identifying a set of candidate services for each process task, negotiating QoS, specifying quality constraints, and identifying mapping rules for invoking services with different interfaces. The runtime engine exploits the design-time mechanisms to support adaptation during process execution, in terms of selecting the best set of services to execute the process, reacting to a service failure, or preserving the execution when a context change occurs. Finally the Vienna Runtime Environment for Service-oriented Computing (VRESCo) (Hummer et al., 2011) is a framework implemented to address issues like dynamic selection, binding and invocation of services.

In general, adaptable systems change their behavior, reconfigure their structure and evolve over time reacting to changes in the operating conditions, so to always meet users' expectations.

Nowadays, recent literature contributions started to address adaptation issues in the contexts of Grid or Cloud Computing. All the existing contributions focus on specific aspects. In (Brandic et al. 2009) authors considers cloud computing stating that such environment require adaptable services that can cope with system failures and environmental change minimizing human intervention. In particular, this chapter mainly proposes model for service negotiation and SLA mapping. (Lim et al. 2009) instead considers "Elastic cloud computing APIs" as the natural opportunity for designing controllers able to automate an adaptive service and resource provisioning, and many recent works have explored feedback control policies for a variety of network services under various assumptions. The approach presented in this chapter aims at providing a wider support in the design and adaptation of service based applications executed in cloud computing contexts.

9. Conclusion

In this chapter we have discussed about the needs that trigger self-adaptation in service-oriented systems and we have seen that adaptation can occur at different layers of the system architecture, and, in some cases, it can also encompass different layers. The life cycle offered by the S-Cube project can be a helpful tool to guide developers in the identification of those self-adaptation triggers and strategies that are relevant for the specific system under development and to allow providers to offer the proper runtime monitoring and reasoning mechanisms needed to execute the defined adaptation strategies.

Acknowledgements

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

Ardagna D. and Pernici B. 2007. “Adaptive Service Composition in Flexible Processes”. IEEE Trans. Software Eng. 33(6), (pp. 369-384).

Ardagna D., Comuzzi M., Mussi E., Pernici B., and Plebani P. 2007. "PAWS: a framework for processes with adaptive web services", *IEEE Software* 24(6), pp. 39-46.

Armbrust, M., Fox A., Griffith R., Joseph A. D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I., and Zaharia M. 2010. “A view of cloud computing”. *Communication ACM*, 53:50-58.

Baresi Luciano and Guinea Sam. 2005. “Dynamo: Dynamic Monitoring of WS-BPEL Processes”. ICSOC 2005: 478-483

Baresi L., Guinea S., and Pasquale L. 2007. „Self-healing BPEL processes with Dynamo and the JBoss rule engine“. In: Proc. of ESSPE 2007, pp. 11–20. ACM Press, New York

Brandic I., Music D., and Dustdar S. 2009. „Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services“. In Proceedings of the 6th international conference industry session on Grids meets autonomic computing (GMAC '09).

Bucchiarone A., Cappiello C., Di Nitto E., Kazhamiakin R., Mazza V., Pistore M.2009.„Design for Adaptation of Service-Based Applications: Main Issues and Requirements“. Proc. of ICSOC/ServiceWave Workshops, pp. 467-476.

Bucchiarone A., Kazhamiakin R., Cappiello C., Di Nitto E., and Mazza V. 2010. „A context-driven adaptation process for service-based applications“. In Proc. of PESOS 2010.

Bucchiarone A., Marconi A., Pistore M., Föll S., Herrmann K., Hiesinger C., Marinovic S. 2010b. “An Overall Process for Self-Adaptive Pervasive Systems”, in Proceedings of the Second International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE 2010), Elsevier.

Cheng B. et al. 2009. “Software engineering for self-adaptive systems: A research roadmap”. In B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, Software Engineering for Self-Adaptive Systems, ser. LNCS, no. 5525. Springer.

Colombo M., Di Nitto E. and Mauri M. 2006. „SCENE: A service composition execution environment supporting dynamic changes disciplined through rules“. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 191–202. Springer, Heidelberg.

De Lemos R. et al. 2011. “Software Engineering for Self-Adpaptive Systems: A second Research Roadmap”. In R. de Lemos, H. Giese, H. Müller, and M. Shaw, editors, Software Engineering for Self-Adaptive Systems, number 10431 in Dagstuhl Seminar Proceedings, Dagstuhl, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Dey A. K. and Abowd G. D. 2000. “Towards a Better Understanding of Context and Context-Awareness”. In Workshop on The What, Who, Where, When, and How of Context-Awareness.

Di Nitto E., Ghezzi C., Metzger A., Papazoglou M., and Pohl V. 2008.„A journey to highly dynamic, self-adaptive service-based applications”. *Automated Software Engineering.*

Guinea S., Kecskemeti, G., Marconi A., and Wetzstein B. 2011. “Multi-layered Monitoring and Adaptation”. In ICSOC 2011, LNCS. Springer.

Hallerbach A., Bauer T. and Reichert M. 2008. „Managing process variants in the process life cycle“. In: Proc.ICEIS, vol. 3-2,pp.154–161.

Hofer T., Schwinger W., Pichler M., Leonhartsberger G., and Altmann J. 2002. “Context-awareness on mobile devices: the Hydrogen approach.” In 36th Annual Hawaii International Conference on System Sciences, pages 292-302.

Hummer W., Leitner Ph., Michlmayr A., Rosenberg F., and Dustdar S. 2011. "VRESCo - Vienna Runtime Environment for Service-oriented

Computing" (invited) in: "Service Engineering: European Research Results", S. Dustdar, F. Li (ed.); Springer, 299 - 324.

Jackson M. 2001. "*Problem Frames: Analysing and Structuring Software Development Problems"*. Addison-Wesley, New York.

Josuttis N. 2007. "*SOA in Practice: The Art of Distributed System Design*". O'Reilly Media, 2007.

Karastoyanova D., Houspanossian A., Cilia M., Leymann F., Buchmann, A.P. 2005. „Extending BPEL for run time adaptability". In: Proc. of EDOC 2005, pp. 15–26. IEEE Press, Los Alamitos.

Kazhamiakin R., Bertoli P., Paolucci M., Pistore M., and Wagner M.. 2008. "Having Services "YourWay!": Towards User-Centric Composition of Mobile Services". FIS 2008: 94-106.

Kazhamiakin R., Metzger A. and Pistore M. 2008b, "Towards correctness assurance in adaptive service-based applications," in ServiceWave 2008, ser. LNCS, no. 5377. Springer, 2008.

Kongdenfha W., Saint-Paul R., Benatallah B., and Casati F. 2006. "An aspect-oriented framework for service adaptation". In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 15–26. Springer, Heidelberg.

Lim H. C., Babu S., Chase J. S., and Parekh S. S. 2009. „Automated control in cloud computing: challenges and opportunities". In Proceedings of the 1st workshop on Automated control for datacenters and clouds (ACDC '09). ACM, New York, NY, USA, 13-18.

Linner D., Pfeffer H. Radusch I., and Steglich S.. 2007. "Biology as Inspiration Towards a Novel Service Life-Cycle". In ATC, pp. 94–102.

Meiländer D., Gorlatch S., Cappiello C., Mazza V., Kazhamiakin R., and Bucchiarone A. 2010. „Using a Lifecycle Model for Developing and Executing Adaptable Interactive Distributed Applications". In Towards a Service-Based Internet. pp. 175-186, Springer.

Meiländer D., Ploss A., Glinka F., and Gorlatch S. 2011. „A Dynamic Resource Management System for Real-Time Online Applications on Clouds". Lecture Notes in Computer Science, Springer.

Metzger A. and Cassales Marquezan C., "Future Internet Apps: The next wave of adaptive service-oriented systems?" in ServiceWave 2011, ser. LNCS. Springer, 2011.

Moser O., Rosenberg F., and Dustdar S. 2008. "Non-intrusive monitoring and service adaptation for ws-bpel". In Proceeding of the 17th international conference on World Wide Web, WWW '08, pages 815-824, New York, ACM.

Narendra N., Ponnalagu K. Krishnamurthy J., and Ramkumar R. 2007. „Run-Time Adaptation of Non-functional Properties of Composite

Web Services Using Aspect-Oriented Programming“ in the Proceedings of International Conference on Service-Oriented Computing – ICSOC 2007, Krämer, B., Lin, K., Narasimhan, P. eds., Lecture Notes in Computer Science, vol. 4749, pp. 546 - 557, Springer Berlin / Heidelberg.

Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L., Zagorodnov D. 2009. „The Eucalyptus Open-Source Cloud-Computing System“. In 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 124-131. IEEE Computer Society.

Papazoglou M., Pohl K., Parkin M., and Metzger A., Eds. 2010. “*Service Research Challenges and Solutions for the Future Internet: S-Cube – Towards Mechanisms and Methods for Engineering, Managing, and Adapting Service-Based Systems*”. Heidelberg, Germany: Springer.

Popescu R., Staikopoulos A., Liu P., Brogi A., and Clarke S. 2010. “Taxonomy-driven adaptation of multi-layer applications using templates”. In: SASO.

Real-Time-Framework. 2011, http://www.real-time-framework.com

Salehie M. and Tahvildari L. 2009. “Self-adaptive software: Landscape and research challenges”. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2).

Schmieders E., Micsik A., Oriol M., Mahbub K., and Kazhamiakin R. 2011. “Combining SLA Prediction and Cross Layer Adaptation for Preventing SLA Violations”. 2nd WOSS Workshop. 2011, to appear in Scalable Computing: Practice and Experience, December 2011.

Spanoudakis G., Zisman A., and Kozlenkov A. 2005. „A Service Discovery Framework for Service Centric Systems“. Services Computing, IEEE International Conference on, 1:251–259.

Trainotti M., Pistore M., Calabrese G., Zacco G., Lucchese G., Barbon F., Bertoli P., and Traverso P. 2005. “ASTRO: Supporting Composition and Execution of Web Services”. In ICSOC 2005.

Verma K., Gomadam K., Sheth A. P., Miller J. A., and Wu Z.. 2005. „The METEOR-S Approach for Configuring and Executing Dynamic Web Processes“. Technical report, University of Georgia, Athens.

Vidackovic K., Weiner N., Kett H. and Renner T. 2009. “Towards business-oriented monitoring and adaptation of distributed service-based applications from a process owner's viewpoint”. In: ICSOC/ServiceWave Workshops, Springer.

Zengin A., Marconi A. and Pistore M. 2011. “CLAM: Cross-layer Adaptation Manager for Service-Based Applications”. In: QASBA Workshop @ ECOWS 2011, Lugano, CH.

Zheng, Z., Zhang, Y., and Lyu, M.R. 2010. “Distributed QoS evaluation for real-world web services”. In: Proceedings of the 2010 IEEE International Conference on Web Services. pp. 83--90. ICWS '10, IEEE Computer Society, Washington