Grant Agreement N° 215483

Title:	*Taxonomy of Adaptation Principles and Mechanisms*
Authors:	*CITY, FBK, INRIA, Polimi, SZTAKI, Tilburg, UCBL, UniDue*
Editors:	*Julia Hielscher (UniDue), Andreas Metzger (UniDue), Raman Kazhamiakin (FBK)*
Reviewers:	*Vasilios Andrikopoulos (Tilburg)*
	Branimir Wetzstein (USTUTT)
Identifier:	*Deliverable # CD-JRA-1.2.2*
Type:	*Contractual Deliverable*
Version:	*1.0*
Date:	*16 March 2009*
Status:	*Final*
Class:	*External*

Management Summary

The deliverable presents the vision on the adaptation and monitoring research highlighting the research challenges, objectives, and an integrated adaptation and monitoring framework adopted within this workpackage. Starting from this framework, the refined conceptual models and taxonomies of SBA monitoring and adaptation are provided. The deliverable also demonstrates how the presented taxonomies are instantiated across functional SBA layers and involved research disciplines.

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 today's 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/

Contents

1 Introduction **3**
1.1 Aims and Focus of the Deliverable 3
1.2 Relationships with other S-Cube Deliverables 4
1.3 Methodology 5

2 Vision **6**
2.1 Conceptual Adaptation and Monitoring Framework 7
2.1.1 Elements of the Framework 7
2.1.2 Usages of the Framework 9
2.1.3 User Perspective 10

3 Taxonomy of Monitoring Principles and Mechanisms **12**
3.1 Conceptual Model 12
3.2 Monitoring Taxonomy 13
3.2.1 Taxonomy Dimension: Why? 14
3.2.2 Taxonomy Dimension: Who? 14
3.2.3 Taxonomy Dimension: What? 16
3.2.4 Taxonomy Dimension: How? 17
3.3 Monitoring in Relevant Areas and Domains 18
3.3.1 Monitoring in Business Process Management 18
3.3.2 Monitoring in Service-Oriented Architectures 22
3.3.3 User and HCI Aspects in Monitoring 25
3.3.4 Monitoring in Grid Computing 28
3.3.5 Monitoring in Component-Based Systems 32
3.4 From Monitoring to Adaptation 36
3.4.1 Lack of conformance 37
3.4.2 Requirements changing 38

4 Taxonomy of Adaptation Principles and Mechanisms **39**
4.1 Conceptual Model 39
4.2 Adaptation Taxonomy 40
4.2.1 Taxonomy Dimension: Why? 40
4.2.2 Taxonomy Dimension: Who? 42
4.2.3 Taxonomy Dimension: What? 42
4.2.4 Taxonomy Dimension: How? 42
4.3 Adaptation in Relevant Areas and Domains 45
4.3.1 Adaptation at Business Process Management 45
4.3.2 Adaptation in Service-Oriented Architectures 48
4.3.3 User and HCI Aspects in Adaptation 53
4.3.4 Adaptation in Grid computing 54

4.3.5 Adaptation in Component-Based Systems 56

5 Conclusion **59**

Chapter 1

Introduction

1.1 Aims and Focus of the Deliverable

This document is part of the deliverable series of S-Cube, the Software Services and Systems Network. The goal of this deliverable is twofold. First, it aims to provide the research vision on the research problems and challenges to be addressed by the S-Cube consortium with respect to SBA adaptation and monitoring. Second, it aims to resolve the diversity in terminology and interpretation of related terms between different communities identified in the previous deliverable [21], and to provide a comprehensive and holistic overview of the knowledge and concepts in the field of adaptation and monitoring.

In order to achieve these goals, the deliverable first presents the research problems and challenges for SBA adaptation and monitoring and a first version of the integrated adaptation and monitoring framework (Chapter 2). This integrated framework (depicted by the abstract conceptual model in Figure 2.1) provides a high level view of the key logical elements needed for adaptation and monitoring of SBAs and shows the basic dependencies between them. The framework not only provides a basis for the alignment of different research communities and their terminology, but also shows a way to resolve one of the key research challenges of this workpackages – to provide an integrated, holistic, and comprehensive adaptation and monitoring principles, techniques, and methodologies.

Starting from this general framework the subsequent parts (Chapter 3 and 4 for monitoring and adaptation respectively) present separate conceptual models that provide a refinement of the key logical elements of the framework by introducing important (however, generic) concepts and their relationships. The refinement from the A&M framework is depicted in these conceptual models by introducing packages which are named according to the logical elements in the A&M framework. The adaptation- and monitoring-specific conceptual models are further specialized and classified in the form of the corresponding adaptation and monitoring taxonomies. The classification within taxonomies is structured such as to answer the following questions about the corresponding concepts: “Why?”, “Who?”, “What?” and “How?”. In particular,

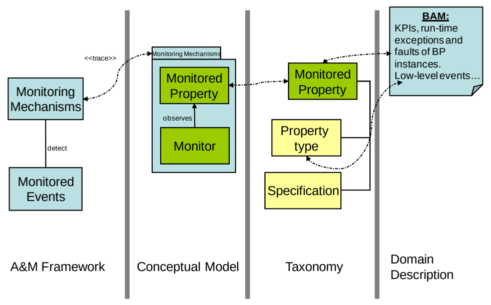
- the “Why?” dimension provides a description of the motivation for monitoring respectively adaptation;
- the “Who?” dimension characterizes the monitoring problem respectively adaptation problem from the user point of view;
- the “What?” dimension is used to classify the subject of monitoring respectively adaptation and the way it is described;
- the “How?” dimension describes the way the monitoring approach respectively adaptation approach is delivered.

These generic concepts and taxonomies are then related to individual domains and research challenges to show how those domains and research areas fit within these refined models. The representation of domains is given in a form of tables, which follow the taxonomy structure (i.e., the "Why?", "Who?", "What?", and "How?" dimensions) and include the key elements of that taxonomy. In this way, the tables are able to capture all the relevant aspects of the problem and its solution in a particular domain, without, however, going too much in details. In particular, the taxonomies aim to take into account the following research disciplines:

- Software Engineering (and, in particular, component-based software engineering),
- Business Process Management,
- Service-oriented Computing,
- Human Computer Interaction, and
- Grid Computing.

Based on those domain instantiations, the differing or similar usage of terms can be identified. In fact, key definitions for concepts (both domain-specific and generic) are provided and will be contributed to the S-Cube knowledge model.

The following figure illustrates the dependencies between the various model using the concepts from the SBA monitoring:



Finally, in Chapter 5 we conclude the deliverable with final remarks.

1.2 Relationships with other S-Cube Deliverables

Several of the topics addressed in this deliverable are closely related to the topics addressed in other workpackages of S-Cube, or are addressed – with a different focus – in those workpackages. The relationships of this deliverable with other S-Cube deliverables include:

- *PO-JRA-1.1.3* ('Codified HCI Knowledge and Context Factors'): In PO-JRA-1.1.3 HCI knowledge and context factors are documented in an explicit form. HCI knowledge and context factors are also relevant for monitoring and adaptation concerns.
- *CD-JRA-1.3.2* ('Quality reference model for SBA'): In that deliverable quality attributes that could be monitored are further described.
- *CD-IA-1.1.2* ('Separate knowledge models for functional layers'): The objective of CD-IA-1.1.1 is to build separate knowledge models for the three functional layers of service based applications. The taxonomies and definitions of terms made in this deliverable will be integrated, with the aim of supporting a common understanding and consistent use of terminology throughout the workpackages of S-Cube.

- *PO-JRA-1.2.1* ('State of the art report, gap analysis of knowledge on principles, techniques and methodologies for monitoring and adaptation of SBAs'): The analysis performed in PO-JRA-1.2.1 has identified a diversity in terminology and interpretation of terms between different communities addressing service-based applications. To address this diversity and consolidate and align the terminology on monitoring and adaptation of service-based applications, this deliverable provides a comprehensive taxonomy.

1.3 Methodology

In order to identify, consolidate, and evaluate the adaptation and monitoring framework, the conceptual models, and the corresponding taxonomies, we performed the following research activities:

1. Starting from the state-of-the-art overview of the research results on SBA adaptation and monitoring and using the initial classification of those results presented in Deliverable *PO-JRA-1.2.1*, we have been incrementally building the generalized representation of the most important A&M concepts and their relationships. This activity aimed to come up with a general and holistic conceptual model and its classification that is equally applicable to literally any adaptation- and monitoring-related problem within the research disciplines and areas the S-Cube project focuses on. As a result of this activity, we obtained the following models:
   - a general integrated adaptation and monitoring framework (Section 2.1). On the one hand, the framework is general enough to equally represent various domains and disciplines, and on the other hand, reflects the key concepts, the architecture and the process of the SBA adaptation and monitoring. The concepts of the framework are further refined into the models specific to adaptation and monitoring.
   - a refinement of the general A&M framework by means of individual conceptual models for monitoring (Section 3.1), adaptation (Section 4.1) and potential dependencies between monitoring and adaptation (Section 3.4).
   - a classification of the SBA monitoring and adaptation concepts in the form of corresponding monitoring (Section 3.2) and adaptation taxonomies (Section 4.2). This classification represent the evolution of the classification represented in the state-of-the-art deliverable PO-JRA-1.2.1. The elements of the taxonomies are sill generic and may be applied to various disciplines and domains. Having in mind that the presented taxonomies will be used within the project to derive novel holistic and integrated A&M solutions, the classification aims to define requirements for such solutions, and therefore is defined to answer the "Why?", "Who?", "What?", a "How?" questions.
2. In order to evaluate the presented models and classifications, we then demonstrated how various monitoring (Section 3.3) and adaptation problems (Section 4.3) identified in the literature can be captured with those models. For these purposes, we considered various problems and different research disciplines and interpreted those problems in terms of the presented concepts. The obtained instantiations not only show the usability of the defined taxonomies, but also provide a basis for the future integration of various state-of-the-art solutions as they uniformly represent different terminology and techniques.

Chapter 2

Vision

Service-Based Applications (SBA) run in dynamic business environments and address constantly evolving requirements. These applications should hence become drastically more flexible, as they should be able to adequately identify and react to various changes in the business requirements and application context. These challenges make monitoring and adaptation the key elements of modern SBA functionality.

The problem of monitoring and adaptation of various types of software system has gained a lot of interest both in the research community and in industry. In the recent years, these aspects have attracted more and more interest in the area of SBA and in Service-Oriented Computing (SOC). However, the results and directions are still insufficient. First, the proposed approaches are very *fragmented*; they address only specific problems, particular application domains, and particular types of applications and systems; the monitoring solutions are often isolated from the adaptation needs and approaches. Second, most of the approaches dealing with adaptation address the problem *reactively*: the solutions aim to define a way to recovery from the problem when it is already happened rather than to prevent it to happen. This is, indeed, insufficient in certain applications and domains. Third, as the applications, their users, and the settings where they operate become more and more dynamic, open, and unpredictable, the role of the *application context* (being a physical, business, or user-specific) becomes much more critical. These issues are often omitted by the state-of-the-art solutions both for monitoring and adaptation. Very relevant to this is also the role and participation of *various types of users* in the monitoring and adaptation process. The service-based applications are often designed to target final users, and, therefore, should be able to collect and properly exploit the information about the user in order to customize and personalize those applications as well as to let the users participate to the corresponding activities.

In this workpackage the work is devoted to the development of the novel principles, techniques, and mechanisms for SBA adaptation and monitoring focused on the following key research aspects and questions:

- **Comprehensive adaptation and monitoring framework**. The workpackage will concentrate on providing holistic framework for adaptation and monitoring principles, techniques, and methods, which will enable the integration of different, isolated, and fragmented solutions. In particular, the framework will allow for:
  - *Integrating solutions for monitoring with the solutions for adaptation* bridging the gap between their models, architectures, and realizations to achieve more efficient and focused support for SBA implementation and management.
  - *Cross layer integration of monitoring approaches*. This form of integration is crucial for modern SBA provisioning, as it provides a way to properly locate and evaluate the source of the problem and its impact.
  - *Cross layer integration of adaptation approaches*. This form of integration is complementary to the previous one and will allow us to properly identify and propagate the necessary adapta-

tion activities in different elements of the SBA architecture. As well as in case of monitoring, new solutions will integrate isolated adaptation mechanisms available at different functional layers (and investigated in the corresponding workpackages of JRA-2) into the holistic cross layer approaches.

  – *Cross boundary integration* of monitoring and adaptation techniques. Here the focus on identifying the role and the impact of various monitored events and adaptation actions on the different participants of the system and its environment, as well as on distributing the information and the actions across those participants accordingly.
  – *Cross life-cycle integration* of monitoring and adaptation techniques. Here the goal is to exploit the knowledge and mechanisms available at different phases of the life-cycle (e.g., design-time or post-operational information) in order to devise, e.g., new monitoring approaches (e.g., exploiting post-mortem process analysis for prediction) or adaptation decision mechanisms (e.g., explore previous decisions and adaptation effects to select proper adaptation strategy).

- **Predictive SBA monitoring and proactive SBA adaptation**. This workpackage will concentrate on the problem of predicting the critical changes in SBA functioning (in collaboration with the workpackage JRA-1.3, end-to-end quality) in order to proactively prevent undesired situations. In particular, the workpackage will focus on new techniques and solutions for adapting the system based on the predicted quality values.
- **Exploiting contextual information and user aspects for SBA monitoring and adaptation**. The information about different types of the SBA context, as well as about the user and its settings, is crucial for the application logic. Novel approaches are necessary for being able to specify and observe this information and for driving the selection, realization, and enactment of the corresponding adaptation actions.

In this chapter we illustrate the novel vision on the SBA adaptation and monitoring that we have defined in S-Cube; this vision will provide a comprehensive, coherent framework for the existing challenges and for the different research lines undertaken by S-Cube and by the broader SOC research community. The vision will also place the adaptation and monitoring research within the global picture and objectives of the S-Cube project; will drive the identification of the competences – and gaps – of the consortium, and to define the research roadmap, which will be addressed by the WP participant and by the project as a whole.

2.1 Conceptual Adaptation and Monitoring Framework

At the high level of abstraction, the adaptation and monitoring framework can be described by the concepts represented in Figure 2.1. This figure identifies Monitoring Mechanisms, Monitored Events, Adaptation Requirements, Adaptation Strategies, Adaptation Mechanisms, and the relations between these concepts, as the key elements of the S-Cube A&M framework. It is important to remark that the significance of this conceptual framework is not in the figure itself – it describes a standard sensing/planning/actuating control chain. The significance is in the very broad meaning that we give to the different concepts, and to the capability of the chain to allow for a very general integration of a wide range of mechanisms, techniques and methodologies for monitoring and adaptation, as discussed in the following.

2.1.1 Elements of the Framework

A generic adaptation and monitoring framework consists of the following elements:

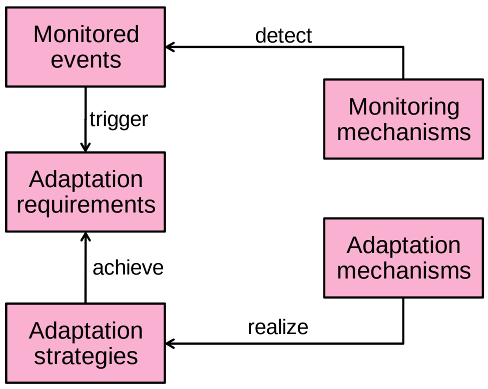


Figure 2.1: Conceptual A&M framework

- With *Monitoring Mechanism* we mean **any** mechanism that can be used to check whether the actual situation corresponds to the expected one. The meaning we give to the monitoring mechanisms is very broad; in this way, we refer not only to "classical" run-time monitoring facilities, but also to techniques such as post-mortem log analysis techniques, data mining, online and offline testing and even verification/validation, etc. Realization of monitoring mechanisms is provided by the corresponding monitoring engines built on top of the monitoring infrastructures.

- Monitoring mechanisms are used to detect *Monitored Events*, i.e., the events that deliver the relevant information about the application execution, evolution, and context. These events represent the fact that there is critical difference with respect to the expected SBA state, functionality, and environment. The monitored events result from observing monitoring properties, derived from the adaptation requirements as a specification of the expected state and functionality of the SBA and its environment. The notion of monitored events may be very broad ranging from basic failures, deviation of QoS parameters, to complex properties over many executions of SBA, certain trends in the SBA environment, changes in business rules, etc.

- Monitored events in turn trigger *Adaptation Requirements*, which represent the necessity to change the underlying SBA in order to remove the difference between the actual (or predicted) situation and the expected one. They may include dependability and functional correctness requirements, optimality, interoperability, usability, etc.

- In order to satisfy adaptation requirements, it is necessary to define *Adaptation Strategies*, which define the possible ways to achieve those requirements given the current situation. Note that it is possible to have a set of different adaptation strategies applicable in the same situation. In this case the process requires certain decision mechanisms that operate autonomously or involve humans.

- Finally, the adaptation strategies are realized by the *Adaptation Mechanisms* – the techniques and facilities provided by the underlying SBA or by the operation and management platform in different functional SBA layers that enable corresponding strategies. The adaptation may be also done "manually", i.e., by re-designing/re-engineering the application. In this case we should speak about application evolution as the permanent SBA changes are required that should be done via SBA re-design.

An important aspect of these conceptual elements is the necessity to define and implement the corresponding *decision mechanisms*, which correspond to the four arrows in the picture in Figure 2.1 and coordinate the work of the framework and realize the relations among them. In particular,

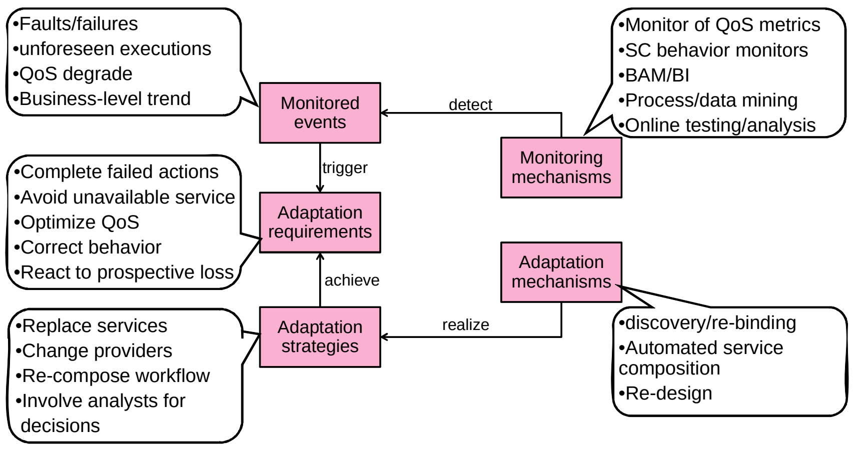


Figure 2.2: Realization of conceptual elements

- *Monitoring properties* allow us to analyze the variety of SBA information observed during its execution and evolution, and to extract and report those events and situations that are critical from the point of view of the monitoring.
- *Adaptation decision mechanisms* relate the monitoring activities with the adaptation activities: they regulate when a particular monitored event corresponds to a situation in which the system should be changed.
- *Strategy decision mechanisms* define the way a particular adaptation strategy is chosen based on the adaptation needs, SBA state, history of previous adaptations, etc. In particular, these mechanisms will provide a way to resolve conflicts among different adaptation requirements.
- *Realization mechanisms* define how a particular strategy is realized, when there is a wide range of available options (e.g., many services to bind in place of failed one).

Note that the realization of these decision mechanisms may be done automatically or may require user involvement. In the latter case we speak about the human-in-the-loop adaptation: the users (with different roles) may decide whether the adaptation is needed, which strategy to choose, and even participate to its realization (e.g., manual ad-hoc SBA adaptation through re-design).

2.1.2 Usages of the Framework

The role of the picture in Figure 2.2 is threefold: to provide an integrated model of the A&M framework, to define a conceptual architecture of such a framework, and to identify an overall adaptation process.

As an **integrated model** for the A&M framework it defines key concepts for monitoring and adaptation which are by design very general. This allows us to capture any adaptation approach in a uniform way, independently from the problem or application domain, discipline, functional layer, or type of the problem addressed. This also provides a basis for the integration of different solutions within a single approach, and re-use of existing solutions for various purposes.

Each conceptual element may be instantiated in a variety of ways (see Figure 2.2):

- different mechanisms and techniques may be exploited for the SBA monitoring such as run-time monitoring tools, online testing, process log analysis.
- a variety of different events may be observed for the same application such as various faults, QoS degrade and violation of SLAs, deviation from the expected behavior. These event may refer to a

particular instance (or execution) of an application or to all the instances; they may also correspond to different functional SBA layers.

- the list of adaptation requirements, strategies, and their implementations may be also very broad (e.g., re-execution of a particular service or changing a provider, modifying the composition or even re-design of the application) and may also refer to a particular instance or to the whole application model.

Furthermore, these instances may be further combined in a variety of ways within different approaches, and then applied to the same SBA.

Almost all the existing approaches covered in the state of the art can be mapped into this model by suitable instantiations of the conceptual elements. For instance,

- Approaches with dynamic re-binding: the service composition is monitored, service faults are detected, a re-execution strategy is realized through discovering, re-binding, and invoking alternative service.
- Provider reputation-based adaptation: QoS metrics statistics is collected, SLA violations are detected, a reputation management strategy is applied, the provider is added to the black list and replaced.
- SBA evolution: the composition is monitored, an undesired execution is recorded, a behavior correction is triggered which is achieved through the evolution of the SBA and the design of a new composition workflow.

More important, novel approaches are being defined as part of the S-Cube research activities by new instantiations of this model.

Second, this picture defines a **conceptual architecture** of a comprehensive adaptation framework. The modularization of these concepts allows us to define key components of the A&M tools, to identify the interfaces between these components and to abstract from any specific realization of these components. On the contrary, within the same component different mechanisms and techniques may be applied in combination. In this way, one can obtain more flexible, customizable, and powerful adaptation and monitoring solutions.

Third, the picture identifies an **overall adaptation process**, where: (i) the relevant information is collected through the monitoring mechanisms; (ii) critical events are recognized; (iii) the need for adaptation is identified; (iv) an appropriate way to perform the adaptation is identified, i.e., an adaptation strategy is selected; and (v) the adaptation is realized by exploiting the available adaptation mechanisms. This adaptation approach is aligned with the SBA life-cycle investigated in workpackage JRA-1.1 and is represented in Figure 2.3.

2.1.3 User Perspective

An important perspective of the introduced vision concerns the involvement of different user roles in the adaptation- and monitoring-related activities across the overall A&M process and across the activities of the SBA life-cycle (Figure 2.3). This perspective introduces an additional dimension of the problem, which makes the corresponding approaches range from completely autonomous (self-* approaches) to interactive and manual (human-in-the-loop approaches). We can distinguish the involvement of the users according to the participation to the life-cycle of the adaptable SBA and to the adaptation and monitoring process.

In case of participation to the life-cycle activities one can identify the roles of requirements engineers, designers, and adaptation engineers. A *Requirements Engineer* defines the application requirements and, therefore, identifies and derives the adaptation and monitoring requirements. A *Designer* (besides designing the SBA itself) may perform manual or semi-automatic design-time adaptation of the application

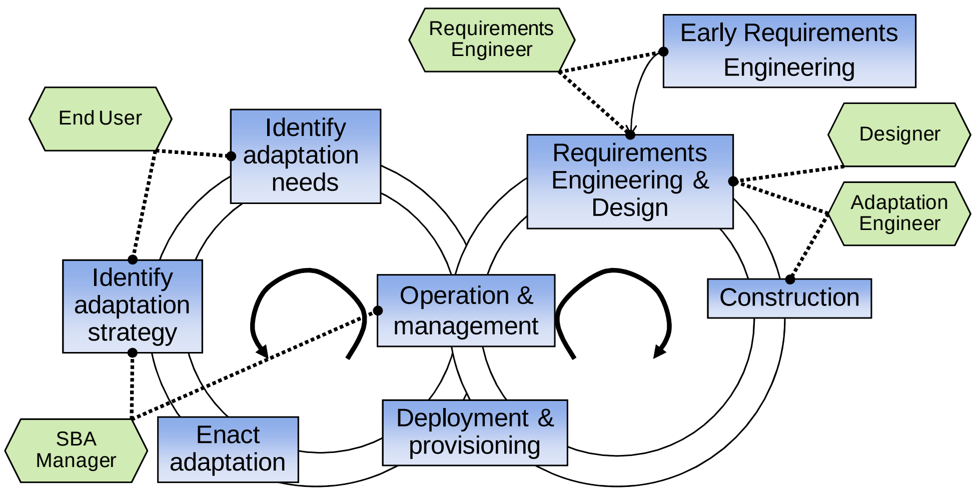

Requirements Engineer
Early Requirements Engineering
End User
Identify adaptation needs
Requirements Engineering & Design
Designer
Adaptation Engineer
Identify adaptation strategy
Operation & management
Construction
SBA Manager
Enact adaptation
Deployment & provisioning

Figure 2.3: User perspective on the Adaptation and Monitoring across SBA life-cycle

based on the information and requests for changes triggered at the operation and management phase of the SBA life-cycle. An *Adaptation Engineer* performs specific activities that target design for monitoring and adaptation, i.e., definition and specification of monitored properties and adaptation strategies, and possibly engineering of novel A&M techniques and mechanisms.

In case of participation to the adaptation process, specific roles may also be identified. Given the conceptual model, these roles correspond to the participation of the user in the realization of various decision mechanisms (to define whether adaptation is needed, which strategy to use and how to realize it). This includes *SBA Manager* (or Integrator), who observes how the application is executed and evolves in order to make critical decisions (e.g., triggering requests for SBA re-design/re-engineering), and *End Users*. The latter may be involved into the A&M process as follows: in case of user-centric SBAs, the adaptation aims to address the needs, preferences, and expectations of a particular user; the system adapts to the context of the user and to the way the user interacts with the application. Therefore, end-users directly or indirectly influence the way the adaptation and monitoring is performed, i.e., affect adaptation and monitoring mechanisms.

Chapter 3

Taxonomy of Monitoring Principles and Mechanisms

The problem of monitoring has been widely studied and exploited in different research areas and application domains, ranging from classical software engineering to service-oriented architectures and grid computing. The term “monitoring”, as well as the definition and conceptualization of the monitoring approach, is strongly related with the particular application area and the kind of problems envisioned in that area and domain.

Having in mind the goal of providing a holistic, comprehensive, and integrated vision on the monitoring and adaptation across various research disciplines, in the following we will try to present a generalized and universal yet practical definition of the monitoring problem. We will present a generic conceptual model for the monitoring, which refines an overall adaptation and monitoring vision adopted within S-Cube research project. Based on the conceptual model, this chapter will provide a classification of the monitoring concepts, and instantiate this classification across relevant research disciplines.

3.1 Conceptual Model

In a broad sense monitoring may be defined *as a process of collecting relevant information in order to evaluate properties of interest over analyzed system and report corresponding events*. That is, monitoring may not only to detect certain facts, but also aggregate, analyze, and reaason over those facts, parameters, and values. High-level conceptual model of the monitoring concepts is represented in Fig. 3.1. We remark that this model refines the generic adaptation and monitoring conceptual model represented in the Vision section of this document with the purpose of providing the relations among the key concepts of the monitoring framework.

As it follows from the diagram, *Monitoring* is performed with the help of *Monitoring Mechanisms*, and in particular by the *Monitors*, which are implemented by a variety of specific *Realization Mechanisms* (tools and techniques). Monitoring Mechanisms include also *Monitoring Properties*, which allow one to identify and focus only on the important events and information. In order to observe those properties, Monitors continuously collect data from various *Information Sources* and detect *Monitored Events* corresponding to these properties. Note that this model of monitoring may have recursive implementation, in a sense that one monitor may serve as a source of information for other monitors. Depending on the purpose and the problem in hand the monitors may range from rather basic components that observe very simple properties, to rather complex monitoring frameworks capable of observing very complex properties defined with high-level specification languages.

Monitoring properties are used to characterize the *Monitoring Subject* under consideration. Depending on the approach, the Monitoring Subject may refer to the SBA itself or to its environment, to its particular elements or particular aspects of the functionality, to a particular run or to the histories of

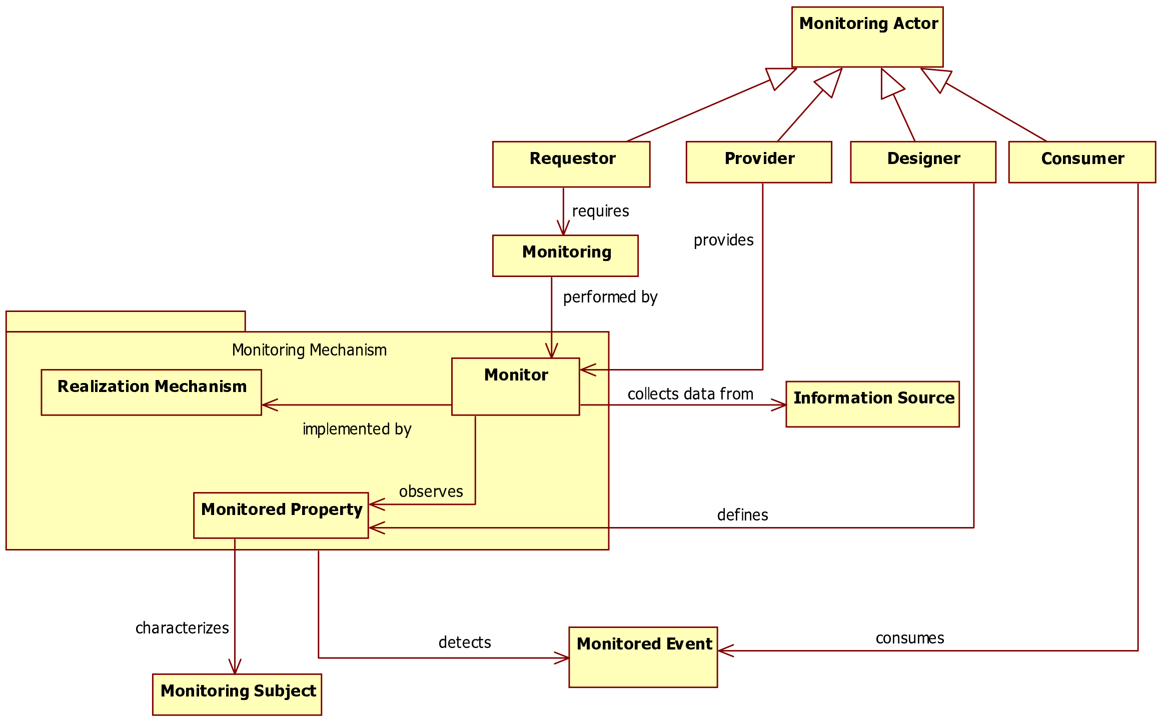


Figure 3.1: High-level monitoring model

executions.

The monitoring process involves various *Monitoring Actors* that characterize different roles, with which the users are involved in the process. One can identify the following types of actors:

- *Requestor* characterizes the stakeholders, who define the requirements to the system, or more precisely, to the monitoring subject.
- *Designer* is responsible for defining the monitoring properties corresponding to the requirements of the requesters, and, if necessary, to design the corresponding monitoring approaches.
- *Provider* represents a role in the ecosystem that owns or *provides* the monitoring functionalities.
- *Consumer* is interested in results of monitoring, i.e., aims to discover important monitoring events and react to them triggering requirements for adaptation.

Note that these roles may correspond to the software components or require human involvement. In particular, the result of the monitoring procedure (i.e., Monitored Events) may be consumed either by the SBA Manager, who will make important decisions on the necessity, e.g., to re-design the application, or by a software component that incorporates the logic to decide whether the run-time SBA adaptation is needed.

Below we will provide a classification of the monitoring problem and identified monitoring concepts.

3.2 Monitoring Taxonomy

Relevant monitoring concepts are classified accordingly. The taxonomy aims to provide a classification for and refine the key elements of the conceptual model of SBA monitoring. We will group the elements of the monitoring taxonomy in a way to answer the following four important questions: *why to monitor*?, *who monitors*?, *what to monitor*?, and *how to monitor*?. Graphical representation of the monitoring taxonomy is given in Fig. 3.2.

3.2.1 Taxonomy Dimension: Why?

The "Why?" dimension provides a description of the motivation for the monitoring. More precisely, the monitoring may be characterized by a particular **Usage** of the monitored information. In general sense, *monitoring is used to reveal critical changes in the application or its environment, which require its adaptation*. This generic purpose may have different forms depending on a particular application, domain or requirements. In particular, the following purposes of the SBA monitoring may be identified:

- *Run-time Correctness Analysis*, to check whether the execution of SBA is correct with respect certain expected specification. This may further include *Fault Monitoring*, which is used to identify different application failures, and *SLA Compliance* necessary to check whether the parameters of run-time execution correspond to the service-level agreement.
- *Diagnosis*, where monitoring is used to reveal and even predict various faults in the application behavior.
- *Optimization* problem, where monitoring is used to identify a possibility for a system to work more efficiently. For this purpose, different characteristics of the SBA performance are continuously monitored.
- *Context-awareness*, where the monitored information reflects the changes in the application environment and provides necessary drivers in order to accommodate to those changes.
- *Evolution*, where the monitoring aims to observe the histories of application execution and changes in order to devise better SBA model, better adaptation mechanisms and strategies.

3.2.2 Taxonomy Dimension: Who?

The "Who?" dimension characterizes the monitoring problem from the following points of view. First, we can characterize it from the point of view of the roles, or *Actors*, involved into monitoring process. We remark here also that the same physical entity may have different logical roles. Indeed, the monitoring results may be consumed by the same stakeholder who defines the monitoring requirements.

Second, the monitoring may be seen from different **Perspectives**. One can distinguish

- *client* perspective sees the system from "outside", aiming to check whether it delivers what is expected by the customers.
- *provider* perspective helps to understand whether it is appropriate "inside", i.e., satisfy the expectations of the system owner.
- *third-party* perspective takes an independent view on the subject of monitoring.

Note that these two aspects are rather orthogonal: the monitoring requirements may come from either client- or provider-site; the monitoring mechanisms may be provided together with the system (provider perspective), installed by the consumers (client perspective), or provided by third parties.

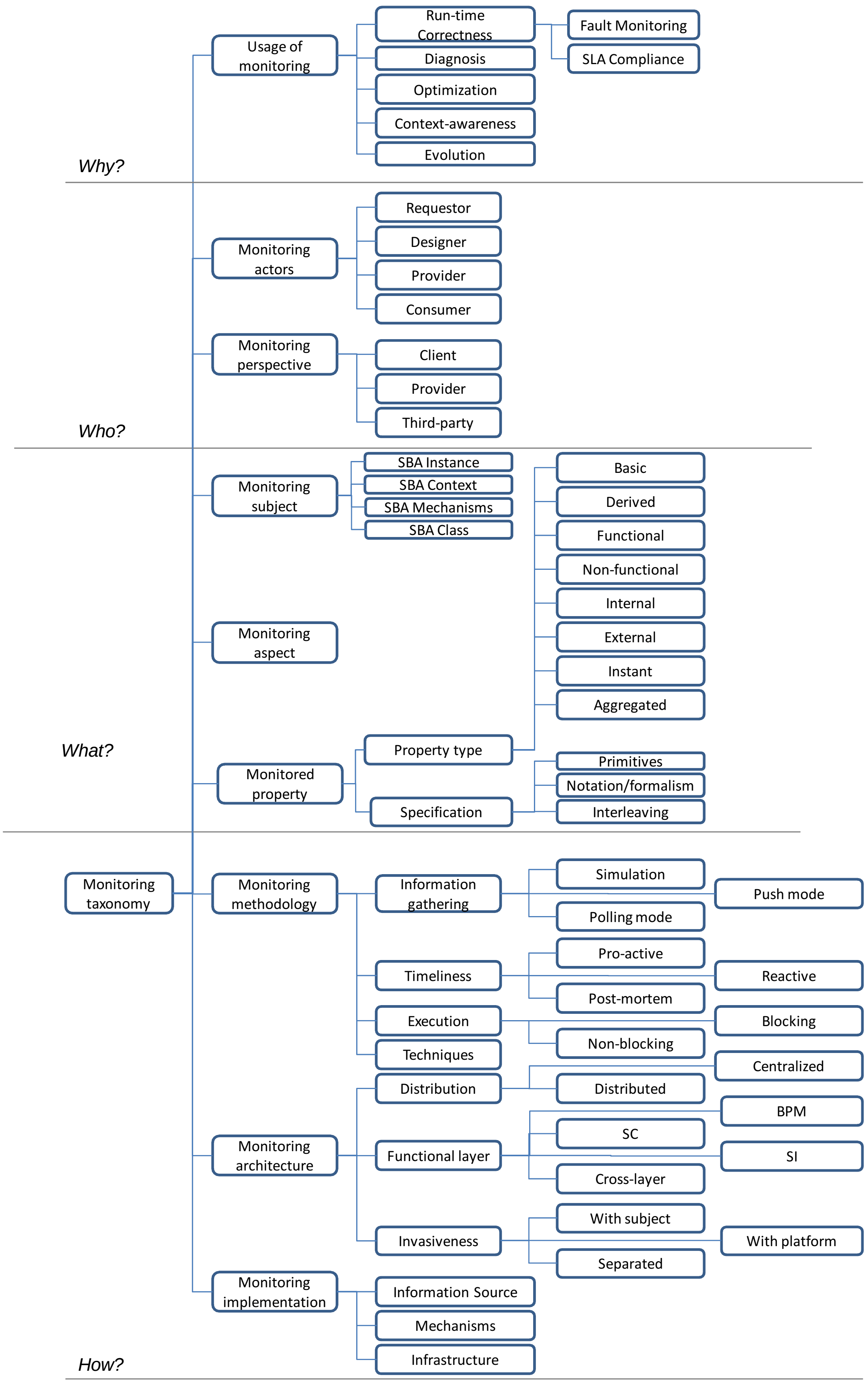


Figure 3.2: Monitoring taxonomy

3.2.3 Taxonomy Dimension: What?

The "What?" dimension is used to classify the subject of monitoring and the way it is described. In this way, we consider the following elements of the taxonomy: Monitoring Subject, Monitoring Aspect, and Monitored Property.

For the **Monitoring Subject** at the highest level we distinguish

- *SBA Instance*, corresponding for instance to a particular BPEL process run, an application customized to a particular user according to her user profile, a particular configuration of a service composition, etc;
- *SBA Class* that define the whole application model, including its business process model, business requirements and KPIs;
- *SBA Context*, or that describe the operational and information environment of the application;
- *Adaptation and Monitoring Mechanisms* themselves, providing a feedback over the way the system is observed, changed, and managed in order to improve them.

These elements may be further decomposed into the elements with finer granularity, e.g., services, compositions, infrastructural elements, traces, locations, etc.

Monitoring Aspect refers to a particular concern of the monitored system relevant for the monitoring requester. Such aspects may refer, to different dimensions of the SBA quality model (e.g., security, dependability, usability), to the functional correctness of the system, to Service-Level Agreements, user-related information and HCI aspects, business-level metrics, KPIs, and requirements.

Monitored Property provides a way to represent these aspects of the monitored system. We further classify monitored properties according to the type of the properties and to their specification. **Property Types** define various characteristics of monitored properties. We distinguish:

- *basic* or *derived*;
- *functional* or *non-functional* properties;
- *internal* or *external* properties;
- *instant* or *aggregated* properties.

Basic properties refer to the elementary primitives and events, while derived properties are recursively defined on top of other properties. Functional properties characterize the function (or behavior) that a given system is expected to provide. Typical examples of the functional properties are failures, assertions or behavioral properties, invariants. Non-functional properties define quality characteristics that often can be measured in a quantitative way. Typical non-functional properties refer to availability, latency, reliability. Internal properties refer to the characteristics internal to the application. On the contrary, external properties describe the environment of the application or its context, whatever notion of the context is exploited. Instant properties refers to the observations performed in a particular moment of time, while the aggregated properties characterize the whole execution, sets of executions or event evolution of the system collecting and aggregating historical data.

Specification of Monitored Properties characterizes the languages means used to define the properties of interest. The relevant elements of such a classification are

- *monitoring primitives*, i.e., basic building blocks used to define more complex derived properties. A typical example is the event property, which refers to elementary events mentioned in the monitored specification.
- *notation* and *formalism* used to unambiguously express the required properties.

- *level of abstraction* from the implementation and domain-specific details.
- degree of *interleaving with the application specification* that characterize how tight the relation between the monitoring specification and application specification is. This may range from cases, where the monitoring specification is a part of application logic, to the cases, where it is defined and changed completely separately from the application logic.

3.2.4 Taxonomy Dimension: How?

The way the monitoring approach is delivered may be further classified according to how it is defined and is supposed to work (Monitoring Methodology), how it is structured (Monitoring Architecture), and how it is realized (Monitoring Implementation).

Monitoring Methodology defines a set of characteristics of the monitoring process itself. It describes, in particular,

- *Information gathering*, i.e., the approach used to collect and if necessary to aggregate data from various information sources. One can distinguish between polling mode of information gathering, when, e.g., the sources are periodically queried, push mode, when the information gathering is event-driven, or more sophisticated simulation mode: a certain model of the monitored property continuously evolve on the basis of the information and events collected in either of the two previous modes.
- *Timeliness*, i.e., the characteristic of the time difference between the moment, when the event actually takes place, and the moment it is reported by the monitor. In these regards, one can distinguish reactive monitoring approaches, which aim to reports events as soon as it is possible, post-mortem approaches, which report information considerably after the events (or even series of events) take place, and pro-active approaches that try to predict the occurrence of events.
- *Execution*, i.e., the characteristics of the monitoring process with respect to the system execution process. One can distinguish between blocking (or synchronous) approaches, where the execution of the monitoring subject is blocked until all the monitoring measurements are done, and non-blocking (or asynchronous) approaches, where the monitoring process is performed in parallel with the execution / evolution of the monitoring subject.
- *Monitoring Techniques*, i.e., particular solutions exploited in order to provide the above characteristics of the monitoring process. Data or process mining, database monitoring, automata-theoretic approaches to define the logics of the monitor model are the examples of such techniques.

Monitoring Architecture defines the way the monitoring framework is structured and decomposed. The relevant characteristics of this architecture are

- *Distribution*, i.e., “horizontal” structuring of the monitoring framework. It defines how the components of the framework are logically and physically located. We distinguish between centralized architectures, where the monitoring components are concentrated in a single node, and distributed architectures, where the monitoring components are distributed across the network, according, e.g., to the distribution of SBA components.
- *Functional SBA Layers* involved in the monitoring, i.e., “vertical” structuring of the monitoring framework. The monitoring framework may be built on top of the single components and elements provided in business process management layer, service composition layer, and service infrastructure layer, or may involve sets of those components across functional layers (cross-layer monitoring).

- *Invasiveness*, i.e., characteristic of the monitoring framework from the perspective of how tightly it is integrated with the monitoring subject. We distinguish between the cases, when the monitoring facilities are integrated with the subject, the cases, when the monitoring facilities are integrated with the platform, where the subject operates, and the cases, when the monitoring facilities are completely separated and independent from the subject of monitoring.

Monitoring Implementation defines the way the monitoring methodology and architecture are realized. It is characterized by the Information Sources, the Realization Mechanisms, and the Monitoring Infrastructure.

- *Information Sources* represent various components and entities that provide all the data, which is used by the monitor in order to evaluate the monitored properties. These sources may range from rather basic elements (such as messages, log files, or timers), to more complex monitors based on top of them (sensors, probes), to hierarchically complex monitoring systems, thus providing recursive and reusable monitoring solutions. In other words, one monitor may re-use another monitor as a source if information, where the information are the events reported by the latter.
- *Realization Mechanisms* define the tools and facilities, necessary to enable a given monitoring methodology, to implement the monitoring techniques, and to build the corresponding monitoring architecture. As it follows from this generic definition, realization mechanisms strongly depend on a given monitoring problem and on the approach used for that. Typical examples include, in particular, aspect-oriented programming techniques that enable "injection" of monitors into the application or to the platform code; automatic generators of monitoring programs that are used to device executable monitors from high-level monitoring specifications; dynamic monitoring solutions, which enable on-the-fly modifications of the way the monitoring of a given subject is performed, e.g., by changing the set of monitored properties or their priorities.
- *Monitoring Infrastructure* refers to the tools and facilities that provides a basis for the monitoring framework. It includes services and APIs for relating to specific information sources, for accessing and managing other monitors, containers and execution platforms to deploy and execute monitoring code, etc. As in the case of realization mechanisms, these functionalities may be very specific for various monitoring approaches.

3.3 Monitoring in Relevant Areas and Domains

The presented conceptual model and taxonomy of SBA monitoring is rather general and abstract; it also covers different possible aspects of the monitoring problem and their arbitrary combinations. When, however, seen from the perspective of a particular functional SBA layer (BPM, SC, SI), particular problem domain (SBA evolution, correctness, optimization), or even application area, only certain elements of the taxonomy are covered with particular interpretation and usage of those elements. Below we will show how the generic monitoring taxonomy is instantiated in different function SBA layer, and even with respect to different monitoring problems. For this purpose, we present the projection of the monitoring taxonomy on a particular domain in a tabular form. The table that captures the key elements of the taxonomy, while the following text aims to clarify and explain that domain and the projection.

3.3.1 Monitoring in Business Process Management

Monitoring in Business Process Management refers to the process of collecting, aggregating, analyzing, and presenting the information regarding the execution and evolution of business process instances. The observed information may refer both to the functional aspects of the process execution and to its non-functional characteristics. Depending on timing of the evaluating this information one can distinguish *Business Activity Monitoring*, where near real time information is presented, and *post-mortem process* *analysis* (including also Business Intelligence solutions), where the collected and provided information refers to the sets of the business process executions.

Business Activity Monitoring

Monitoring in Business Process Management (BPM) is called Business Activity Monitoring (BAM) [51]. The activities monitored by BAM are instances of business processes, realized using BPM technologies such as BPEL, or by integrating heterogeneous information systems. The goal of BAM is to find trends of execution (e.g., bottlenecks, recurring conditions that lead to exceptions and failures), calculate Key Performance Indicators (KPIs), provide an overview of the overall state of the running business processes using dashboards that (visually) aggregate the available information to support, for instance, decision making (e.g. allocate more resources, change QoS parameters, etc), and proactively alert the business managers for corrective actions if KPIs are not met, business rules are violated, or exceptions occur. The output of BAM may result in the adaptation of business process instances (e.g., changes applied to the structure of the workflow run by a particular instance) as corrective actions when the monitored activities do not perform as expected. Unlike post-mortem process analysis (i.e. analysis of the logs of processes that are terminated possibly with a failure), BAM technologies support reacting to changes and violations in the business environment more promptly, and facilitate the applying of corrective actions to the business process instances while they are still alive.

BAM is as a run-time, event-driven extension of Business Intelligence[11, 93]. The term Business Intelligence denotes a broad category of applications and technologies for gathering, storing, analyzing, and providing access to data to help enterprise users make better business decisions. The emphasis of Business Intelligence is on predicting future behavior such as forecasting, scenario planning, optimization, which are derived on the basis of analytical processing and inference. At the operational level, BAM provides (near-) real-time monitoring by collecting and processing low-level events (e.g. system events such as internal errors, completion of tasks, etc) generated during the execution of the instances are collected from a diversity of sources such as (but not limited to) process engines, ERP systems, databases, legacy applications. Once collected, the events are composed and correlated using Complex Event Processing (CEP) [115] technologies. Low-level events are analyzed and aggregating into high-level events (also known as business events such as the fulfillment and violation of KPIs, failure of sub-systems, etc). Low- and high-level events can also be correlated with historical data and trends resulting from Business Intelligence to achieve a holistic view of business activities [109], which supports decision-making at operational and strategic level [116].

BAM solutions generally rely on Enterprise Service Bus facilities for the collection of low-level events, and employ dashboards to graphically visualize high-level in (near) real-time as gauges and graphs (pie-charts, histograms, etc) to provide an overlook of the current state of the execution of the various activities and the measurement of the KPIs, and to allow for prompt response to undesirable situations (e.g. processes under-performing, KPI not met).

Post-mortem process analysis

Post-mortem process analysis, also called *off-line process monitoring*, is a complementary way to perform business process monitoring. Performed after execution, the objective is not to be reactive when exceptions occurs, but to analyse in a more deep and global way the behaviour of past instances. The result of analysis is an implicit knowledge about the business process, useful for making diagnosis about process implementation, behaviour or usage. In particular, the following usages are usually identified:

- to reconstruct the current model of the underlying business process as it is used in production environment (discovery). This information may be used for evolution purposes, that is, to see how the current business process model evolves, to obtain additional information about the process (e.g., performance, decisions, etc), or to identify certain trends.

Table 3.1: Business Activity Monitoring

Dimension	**Concepts**	**Description**
Why	Usage	Business Activity Monitoring (BAM) collects, aggregates, analyses, and presents near real time information about running activities inter- and intra-organizations and involving partners and customers. BAM may trigger the adaptation of business process instances as corrective action if the functional and non-functional requirements for the business processes change, and those changes must also be applied to the business processes currently running.
Who	Actors	BAM collects *low-level events* from the human- and system- actors involved in the execution of the monitored business process instances. The information, aggregated and refined in the shape of *high-level events*, is provided to the organization's management responsible with the supervision of the business process, and responsible for taking corrective actions.
What	Subject	BAM refers to the business processes (i.e., BPEL, workflows) realized using Business Process Management (BPM) software systems, such as BPEL engines, or obtained via point-to-point integration of heterogeneous systems in the organizations. The subject of monitoring is SBA instances (running instances of business processes) or series of activities and tasks spanning multiple systems and applications, possibly requiring human involvement to be carried out.
	Aspect	BAM focus on functional and non-functional aspects of the execution of business process instances, namely the performance (in terms of fulfillment of pre-defined KPIs) and the detection of exceptions and faults.
	Property	The monitoring properties that express the correct execution of business process instances are expressed in terms of KPIs. Run-time exceptions and faults of business process instances (e.g. a BPEL process terminates for an internal error) are also monitored.
How	Methodology	Low-level events are collected in both push- and pull- manner. The human- and system- actors responsible to carry out parts of the processes autonomously *push* (generate and deliver) low-level events such as reporting the completion of a task and reaching a particular state in the business process; however, the BAM infrastructure can also *pull* (poll) low-level events from the actors (e.g. inquire about the progress achieved in executing a certain task). The aggregation and analysis of low-level events and the presentation of high-level events is pro-active. High-level events are displayed in (near) real-time on dashboards comprising gauges and graphs (pie-charts, histograms, etc) that use color conventions to simplify the detection of problems (e.g. green light if the monitored instances meet the KPIs, yellow light if the performances border the tolerance thresholds, red light if the KPIs are violated).
	Architecture	BAM architectures usually rely on the facilities provided by an Enterprise Service Bus for the collection of low-level events. High-level events are produced by Complex Event Processing (CEP) systems through the aggregation and analysis of low-level events. The presentation of the high-level events can occur in parallel on multiple dashboards connected to the CEP system with client-server approaches. The enactment of adaptation of business process instances that need corrective actions is delegated to the BPM facilities in place.
	Implementation	The implementation of BAM facilities is usually loosely coupled with the implementation of the business processes in BPM systems thanks to the abstraction provided by the events and the Enterprise Service Bus.

- to check the correspondence between the actual model and the expected one, which may be used to trigger a need for the process model re-design.

Table 3.2: Post-mortem process analysis

Dimension	**Concepts**	**Description**
Why	Usage	It is a complementary way to perform BAM. It is used after the execution of the process to analyze in details the behavior of the past instances. The results would be useful for changing and predicting new behavior usage of the process, its design and even its implementation.
Who	Actors	The analysis tool collects data and events from the system. The collected information are refined by human and presented to an expert or responsible of the business process in order to make the decision for the proper changes to adopt.
What	Subject	It is focused on monitoring a set of SBA past business process instances. This instances are represented by a (complex) events log, considered as low-level events.
	Aspect	The data collected from logs after the execution of business process instances can dealt with functional or non functional aspects (i.e., to fulfill the security requirements, to improve the QoS).
	Property	A variant of monitoring properties are handled during the post-mortem analysis from internal to external properties of the behavior expressed for both functional and non functional characteristics, in order to predict new correct behaviors.
How	Methodology	The information is collected in both push and pull way. The human and the system provide the event log (push mode) for reporting the past instances of the process, where the event are persistent and are not the on-the-fly events. The monitoring system may also provide to the analyzer low information independently of the events by reporting the past instances referring to periods of time (pull mode). The post-mortem analysis can be pro-active to predict the future BP behaviors, and their usage using data mining techniques.
	Architecture	The architecture of the framework is completely detached from the application, it is distributed. The log information are collected from different sources, the BPM system or SOAP messages.
	Implementation	The implementation of post-mortem analysis may be integrated into the corresponding BPM system.

Also due to these goals, the analysis can be useful both from the client and the provider perspective. As an example of the former, one can try to find what is the BP external behaviour (also called business protocol), i.e. what is the set of conversations really happened with consumers. As an example of the latter, one can be used to analyse internal workflow, extract rare or frequent event sequences, abnormal terminations or mine satisfied constraints.

The monitoring subject is a set of SBA past business process instances. This instances are represented by a (complex) events log. In other words each log is a sequence of events, corresponding to a particular SBA instance and produced by BPM-system or by components themselves. As it follows from the problem, here the properties are not defined explicitly. Instead, a goal is to monitor logs in order to report the reconstructed process model. Such a model is usually represented using Petri nets formalism.

Even if response time has to be acceptable for monitoring requester, it is not as much crucial as in on-line monitoring context. Moreover, the events are persistent and are not the on-the-fly events. As a consequence, more complex tasks can be performed, and advanced *data-mining technics* can be used. Data mining (in this context, we talk about *log mining*) aims at extracting models or patterns from large and heterogeneous data repository. As sources of information different approaches use either logs produces by the BPM system, where the process is executed, and/or the logs of SOAP messages when the Web service-based business processes are dealt with. In this way, the architecture of the framework is completely detached from the application, while its implementation may be integrated into the corresponding BPM system. Concerning models, one can cite *process mining* which consist in retrieving the workflow which may produce input logs. Other model mining examples are classification and clustering:

the first computes classifiers able to predict future behaviours, while the second extracts natural groups of similar logs, e.g. to detect typical classes of BP usage. For the patterns, log mining takes benefit from sequence or graph mining techniques. Let us notice two important and hard sub-problems that must be considered in the special context of BP log mining. The first one is the (potential) difficulty of log retrieval, i.e. find correlations between events to decide which events are part of the same SBA instance. The second is the need to take into account noise and uncertainty, since some instances can correspond to failures or consumer aborts not always clearly notified in logs.

3.3.2 Monitoring in Service-Oriented Architectures

Service compositions provide a realization of the business process models defined in the BPM functional SBA layer by integrating a set of services. Similarly to the monitoring in BPM, the composition monitoring may target the analysis of the composition correctness or evaluation of its characteristics. Differently from above, however, this forms of monitoring usually target more technical aspects of the application realization or may refer to checking, whether the composition is compliant with regards to the business process model.

Monitoring the correctness of service compositions

The motivation of the run-time correctness analysis is dictated by the fact that the component services participating to the composition are not under the control of the composition designer. In these circumstances, the service compositions are being designed under certain composition constraints (or choreographic assumptions [99]), i.e., the assumptions, under which the component services participate in the composition. These assumptions instantiate the monitored properties that should be observed by the monitor. Note that in this way monitoring correctness of composition may be used for the purpose of business process monitoring: the composition execution is checked for the conformance to the business process model; the process performance metrics are checked to comply with the corresponding KPIs.

The monitoring properties may express both the functional and non-functional characteristics. In the former case, they may have the form of pre-/post-conditions over a particular activity within the composition (such as a service call), temporal properties over the execution of the composition instance or series of instances, or conformance of the composition execution to the composition specification (e.g., domain monitors, [99]). In the latter case the properties focus on the characteristics that may be measured in quantitative way, such as time or statistical information over the occurrence of events. Furthermore, the properties may express the characteristics internal to the composition (i.e., state of the instance, sending a particular message) or external (i.e., information available from auxiliary web services and data sources [19]); they may be instant (e.g., measured in a particular execution point, such as assertions) or aggregated (across the whole execution of an instance or a series of instances).

In order to express various monitored properties, specific monitoring specification languages are usually provided, such as WSCoL [19, 18], RTML [15], EC-Assertion [87]. These languages rely on a variety of formalisms including linear temporal logic, event calculus, first-order logic, algebraic specification [25]. Commonly to this kind of monitoring, the advanced monitored properties are recursively derived from primitive properties and events. Typically such primitive properties refer to composition states, failures and exceptions, messages and message events, predicates over special variables, timestamps, etc. Advanced properties are defined using language-specific predicates, operators, and functions. Various approaches express properties at different level of abstraction, from domain-level properties to implementation-level properties. Usually monitoring specification is separated from the composition code in order to enable separation of concerns.

The commonly used sources of information are the process state data predicates on process states collected through appropriate probes placed throughout the process; SOAP message data related to the events or contents of the message that the service is sending / receiving; external data that does not belong

Table 3.3: Monitoring the correctness of service compositions

Dimension	**Concepts**	**Description**
Why	Usage	Monitoring of Web service compositions is mainly used as the means for *run-time correctness analysis*: it is often refers to checking at run-time whether certain predefined properties are satisfied when the composition is executed. The events that are reported by the monitors in this case correspond to the violation of the monitored properties. Such violations may be further exploited in order to trigger the requirements for the service composition adaptation, either at the level of the composition instance or at the level of the composition structure.
Who	Actors	Run-time correctness analysis addresses the monitoring from both the composition provider perspective, when the monitoring aims to analyse the properties of the composition as a whole, and the client perspective, when the goal is to check certain properties of the constituent services. In either case, the composition provider plays the roles of designer (by specifying the choreographic assumptions), of requester (by instantiating the monitoring process), provider (by deploying and running the monitors), and consumer (making decisions about SBA adaptation either manually or automatically).
What	Subject	Such monitoring concentrates on the monitoring of the single instances of service compositions (i.e., SBA instances), as well as of the sets of instances or classes of compositions (i.e., SBA class), and on the monitoring of constituent services.
	Aspect	Normally, the monitoring approaches focus on behavioral aspects of the composition execution in general (both functional and non-functional, [99, 19, 87]), or on some particular aspect, such as security [22].
	Property	The corresponding monitoring properties that express composition correctness constraints express both the functional and non-functional characteristics. Various notations and tools allow for basic and derived, internal and external, instant and aggregated properties.
How	Methodology	The information is usually collected in push mode, when the properties are evaluated upon certain events; advanced properties are updated by re-evaluation upon receiving such events. The run-time correctness analysis is often reactive; it aims to reports the violation as soon as they occur. Few approaches use pro-active composition monitoring where the current information is used to predict the potential violations, using such monitoring techniques as online testing [72] or verification [78].
	Architecture	The monitoring architecture is usually centralized around the orchestrated service composition in order to provide a possibility to immediately react to those violations [17]. In other cases, the monitoring architecture is integrated with the execution platform [87, 15, 25].
	Implementation	The implementation of the run-time correctness monitoring approaches usually relies on the infrastructure provided by the execution platform and middleware, such as a BPEL execution engine. Typically, the monitoring approaches are realized using aspect-oriented mechanisms or application weaving in order to integrate the framework into the application or platform code, and model-driven code generation approaches that provide a way to extract monitor code from the high-level monitoring specifications.

to the monitored system and should be collected externally, e.g., by calling auxiliary services; low-level events specific to a particular implementation of the execution engine or infrastructure.

Diagnosis in service compositions

Diagnosis in service compositions aims to observe various failures and exceptions across the composed services (i.e., basic monitored properties) and to report the diagnosis hypothesis describing the cause of

Table 3.4: Diagnosis in service compositions

Dimension	Concepts	Description
Why	Usage	The goal of diagnosis is to monitor various composition failures in order to reveal the faults causing such failures [9]. In this way, effective recovery adaptation strategies may be provided and adopted.
Who	Actors	In the diagnosis, the monitoring actors and perspectives are highly interleaved: the monitoring is defined, performed, and exploited both by service providers (local diagnosers) and by the composition provider (global diagnoser).
What	Subject	The subject of diagnosis is the service composition instance. The information is, however, also gathered from the constituent services.
	Aspect	The focus of the diagnosis in service composition is on the various application faults.
	Property	The observed properties refer to the application, composition, and infrastructure exceptions, and the causing failures.
How	Methodology	The monitored information is collected in push mode; the diagnosis starts immediately when the event is detected blocking the execution of the application until the hypothesis is provided.
	Architecture	In order to perform the diagnosis across service composition, the distributed architecture is adopted, where the local diagnosers are associated with the component services and the information is passed to and from the central global diagnoser.
	Implementation	For the purpose of the diagnosis, the implementation of thew diagnosis framework is highly integrated with the code of distributed services, as well as with the execution platform that should provide the global diagnosis capabilities.

the problem and the affected services. Consequently, the monitored properties are expressed in terms of exceptions specific to the service composition implementation notation. The monitored exceptions and the reported diagnosis hypothesis remain in the boundaries of the same composition instance (monitoring subject).

Monitoring for composition optimization

Monitoring for optimization relies on the two key elements: (i) up-to-date knowledge about quality parameters of the participating services and (ii) a measure of the sub-optimality of the composition. The latter is measure with the help of so called *utility function*, which aggregates weighted measurements of single quality properties of the involved services (e.g., cost, reliability, availability, performance, reputation) to calculate the composition optimality. While the former properties usually refer to SLA specifications of the involved services, the latter is usually pre-defined; when the current value of this function is non-optimal the monitoring framework may report such a degrade as a monitoring event.

Monitoring of SLA compliance

A set of properties controlled by the monitored SLA usually refer to variety of service resource and business metrics, such as quality of service characteristics (performance, availability, reliability) cost, and so on. These requirements are negotiated and agreed resulting in electronic contract, usually referred to as Service Level Agreement (SLA). SLA monitoring is, therefore, one of the main usages of the monitoring of Web services. In case of SLA violations, various adaptation activities may be applied. These actions range from simple termination of the service use on the requester side to renegotiation of the SLA properties to complex management and reconfiguration activities on the provider side.

The languages for specifying service requirements may also include the instructions over the application of requirements, e.g., duration of a contract or conditions under which the requirement should be evaluated. There exist a wide range of standard and non-standard languages and their extensions for

Table 3.5: Monitoring for composition optimization

Dimension	Concepts	Description
Why	Usage	Monitoring for composition optimization refers to continuous monitoring of the quality parameters of the services involved in the composition in order to detect changes that are critical for the adaptation to make more optimal decision over the composition configuration.
Who	Actors	Given the usage of this form of monitoring, it is performed from the provider perspective, who is in charge of designing and realizing the monitoring framework, as well as consuming the monitoring results.
What	Subject	The monitoring process aims to observe and continuously evaluate the value of a certain utility function, which characterizes the optimality of the composition, over a series of the composition execution. In other words, the subject of monitoring is the SBA class.
	Aspect	The monitoring framework may target the measurement of the composition optimality with respect to a particular metric or with respect to a set of parameters.
	Property	Various quality characteristics of the involved services represent basic monitored properties, while an utility function used to calculate the composition optimality represents the derived monitoring property.
How	Methodology	composition optimization monitoring usually adopts the polling mode of the information gathering: the queries about the quality parameters are performed either on purpose (e.g., before making a service request) or periodically. The execution may be both non-blocking and blocking (e.g., the service request is not performed until the optimality is defined and the best suitable service is chosen).
	Architecture	For the realization of the monitoring framework the approaches usually define a centralized architecture that recursively relies on the QoS monitors for the constituent services.
	Implementation	Various techniques are used for the defining and computing the utility functions, such as Markov Decision Process [71].

specifying service contracts [5, 26, 79, 85].

3.3.3 User and HCI Aspects in Monitoring

A wide class of SBAs aims to deal with the end user, providing support for her activities, tasks, and operations. Typical examples of such systems amounts to mobile applications that are installed and operated on top of mobile device (mobile phone, PDA, etc.). For such systems the context plays an important role of their functionality and logic. Here the context encompasses various aspects such as *computing environment* (e.g. available processors, network connectivity and capacity, input/output devices), *user environment* (e.g. location, collection of nearby people) and *physical environment* (e.g. lighting, noise level) [48]. Such context is very dynamic; the occurring changes may affect the way SBA is working. In order to adjust a system with respect to the changes in the user context and in the activities the user performs, it is necessary be able to detect those changes and identify where the system needs to be adapted. This demands continuous monitoring of the system, the context, and the interaction of the users with the system. Therefore, the usage of the monitoring may amount to context monitoring in order to support *context-based system adaptation* and monitoring user-computer interactions in order to support *Human Computer Interaction (HCI) based adaptation*.

HCI monitoring

Monitoring of user's computer interaction aims to reason about the sequence of actions taken by the user in the system and helps to better understand the future action of the user [62, 57]. This process

Table 3.6: Monitoring of SLA compliance

Dimension	Concepts	Description
Why	Usage	*Monitoring of SLA compliance* for Web services aims to check whether the service functionality matches the expectations of the requester. The expectations are expressed in a form of a formal contract (SLA), which regulates the expected quality of provided service. When the agreement violation is detected, the interested parties are reported and possible adaptation activities may be fired.
Who	Actors	Given the goals of the monitoring problem, monitoring of Web services may take all the perspectives: it is performed from the client perspective in order to ensure that the expectations are met; from the provider perspective in order to see whether the resources are allocated correctly and optimally with respect to the current contracts and to ensure the contracts. The monitoring requirements are defined by the service requester, while the monitoring functionalities are delivered by the service provider, by the requester and/or by a third party, who performs auditing of the contracts on behalf of all the contractual bodies.
What	Subject	Depending on the type of the contract, the monitoring subject may refer to a particular service execution or to a series of executions (e.g., within a given period).
	Aspect	Monitoring of SLA compliance may refer to a variety of the QoS metrics and properties, as described in the contract.
	Property	The monitored properties for SLA compliance monitoring refer to various service characteristics. They may represent basic properties or measured items, as well as complex properties derived from measured items using specific functions that aggregate the measurements (total amount, number of time average throughput, etc).
How	Methodology	A common approach to the Web service monitoring is to perform timely evaluation of the relevant characteristics through a series of the service invocations. The evaluation is performed on the basis of simple measurements at the level of service infrastructure; the resulting values are aggregated and stored by the monitoring framework. The advanced properties are evaluated by querying the current values of these measurements (i.e., polling mode). It is also possible that the violation events are reported to the interested parties immediately when the evaluation occurs (push mode).
	Architecture	The realization of the Web service monitoring framework relies either on a proxy architecture, which is completely external to the monitored services, or on a instrumented service middleware platform that provides the special monitoring facilities and APIs for performing various measurements also on the provider side.
	Implementation	The implementation depends on the underlying architecture. In case of the proxy architecture the measurements are done via intercepting the SOAP messages exchanged by the service. In case of instrumented middleware platform the measurements are performed via the low-level operations and probes provided by the middleware itself.

can be used to *(i)* adapt the system that may assist the user's future action by making inference of the user intentions [86] and *(ii)* adapt the monitoring process. As an example of case *(i)*, consider, user A accessing a web based application that helps to compare prices of different products. If the application monitors that A is always comparing prices of the products only from company X and Y, ignoring all other companies, then the application may infer that A is interested only in the products of X and Y, and thereby the application can show only product information from companies X and Y to A, and hide product information from other companies. In the case of *(ii)*, consider, for example, a user visiting a web site that shows stock information of different companies and a monitor using a set of rules to monitor that the user is only visiting the site (i.e. user is neither buying nor selling stocks). But if the user starts buying or selling stocks the monitor needs to use some additional rules to monitor the user's interaction with the site. *HCI* monitoring aims to observe a particular aspect of the *SBA* executions: interactions between the user and the application.

HCI monitoring has been exploited to satisfy different user goals. For example, *HCI* monitoring helps the system developers to design their system in such a way that the system may adapt itself based on the interaction of the user with the system and assist the user to accomplish his task more conveniently [57, 86, 12]. Also this technique can be applied to continuous improvement of interactive learning environment, where continuous feedback from the learner plays an important role. In such environments HCI monitoring may help to identify indicators that can be used to generate feedback and their meaning in different contexts [62].

This approach analyzes the user interactions, compares them with the expected model of the user activities and reports the corresponding feedback [12, 86, 57] (i.e., monitored events). This requires specifying hierarchical and sequential structure of tasks that should be performed to accomplish user's goal. This specification is known as task model. Task models are specified either in some description language that provides formal syntax and semantics for creating task models [12], or as a list of keywords [86, 84]. In the later approach, keywords are derived from user interactions (e.g. keyboard input, user email, web pages read) and then a list of keywords is produced by determining the relative frequency of the original keywords. As an information source, *HCI* monitoring receives and processes user event streams from the system being monitored. User event stream is produced by users interacting with the system where the system may be instrumented to generate the required events [12, 86]. These events can be low level system events (e.g. a mouse click or move) or higher level application events (e.g. selection of a menu item) [12, 73]. It is recommended to monitor event streams from more than one event source wherever possible to produce more accurate inference of user intentions [86].

Typically, either client server architecture or repository architecture model is applied to monitor user computer interaction. In client server architecture model, the client provides the front end that is accessed by the user while the client sends the events produced due to user interaction to the server. The server works as the task monitor that receives the events and compares the events with available task models [59, 12]. In repository model, gathered events are stored in a shared repository and then dispatched to interested monitoring agents based on some predefined scheme [73, 86].

Context monitoring

In context-based monitoring, a set of rules defining the properties that should be monitored to detect changes of the context are specified [111, 103, 104]. Some formal [103, 104] or semi formal [111] languages are used to specify the properties to be monitored.

The monitoring techniques in this field facilitate a wide range of stakeholders for different purposes: they are exploited to help the application developers to design the system that will adapt the user interface based on the context changes [70, 63, 33, 105, 53, 16]; they may help the service provider to better understand the user's required quality of service and improve the delivered QoS [117, 13].

Context information can be measured at different level of abstractions, for example low-level context information can be directly captured from the environment using sensors, input devices, and high level context information can be inferred from low level context information and other information sources e.g. browsing user profile [63, 110, 13, 3].

System run-time events (i.e. context information) are matched against the specified properties to detect a change in the context. Run-time events or context information are obtained either from sensors [94], by polling system parameters (e.g. battery level in mobile phone or available bandwidth) [23, 24] or user input [94]. Given the distributed nature of context-aware applications, context-based monitoring is mostly implemented as distributed architecture with middleware support [94, 23, 24, 28]. In this setting, a component in the middleware acts as a coordinator that collects context information from distributed sources and forwards the context information to the specific application/monitor that performs

Table 3.7: HCI Monitoring

Dimension	Concepts	Description
Why	Usage	HCI monitoring is applied to collect user interaction information in a system at the runtime and analyze the collected information to predict the future actions that can be taken by the user. This process enables a system to recognize users' tasks and thereby to co-operate the user by adapting the system according to the users' focus of attention and workload.
Who	Actors	HCI monitoring is mainly performed from i) the system developers perspective, who use the observations from this process to design their system in such a way that the system can adapt itself depending on the users' needs, ii) the service providers perspective, who use the observations from this process to tune the system at runtime to improve the quality of the service.
What	Subject	This process monitors the tasks (i.e. actions) taken by the use in the system. User tasks are specified in a model, known as task model, which is written in a formal or semi-formal task descriptions language.
	Aspect	Task model describes the user tasks and patterns of events at multiple levels of detail. For example tasks could be specified at much higher levels (e.g., typing a file) or very low levels (e.g. move the mouse) of detail.
	Property	Task model describes the sequence of task that the user should perform in the system through the interfaces offered by the system. For example, a task sequence to delete a word could be, (MOVE-CURSOR,CLICK-MOUSE-BUTTON, DOUBLE-CLICK-MOUSE-BUTTON,SHIFT-CLICK-MOUSE-BUTTON, and HIT-DELETE-KEY)
How	Methodology	User event streams are gathered at the runtime from different sources and analyzed to infer user intensions. Various techniques are applied to analyze user event streams, for example, i) user events stream is compared to the specified task model, ii) a probability distribution over user tasks is inferred by applying Bayesian networks.
	Architecture	HCI monitoring applies either i) client server architecture where the client provides front end for user interactions and sends the events to the back end server that analyses the task model against the events; or ii) central repository architecture where events are stored in a shared repository and an appropriate monitoring agent analyses the task model using the events.
	Implementation	Depending on the underlying architecture different implementation mechanism is applied. In case of client server architecture, the front end (client) is often implemented as plugins to the system (e.g. application software like email client, or internet browser) for which human computer interaction is monitored.

the reasoning using context information.

3.3.4 Monitoring in Grid Computing

The purpose of Grid monitoring is directed towards two distinct areas: *infrastructure level* and *application level* monitoring. Infrastructure level monitoring is the main target of Grid monitoring and focuses on observing and recording the state of shared resources. Application level monitors concentrate on tracking the state of user-level applications being executed on various geographically-distributed Grid sites. Grid users need the support of application level monitors for adapting the monitoring system to the dynamic placement of those tasks which the user needs feedback from. The characteristics of each of these levels of monitoring differs substantially and therefore are discussed separately in the following sections.

Table 3.8: Context Monitoring

Dimension	Concepts	Description
Why	Usage	Context monitoring is the process that continuously analyzes contextual information of a system, where contextual information includes any information that may affect the behaviour of the system and/or the interaction of a user with the system. In general context of a system covers a wide range of information including user location, time of the day, nearby people and devices, temperature, light or noise. The outcome of the context monitoring process is used to adapt the system behaviour at different levels, such as notifying user displays or modifying data or controlling actuators.
Who	Actors	Context monitoring is mainly performed from i) the system developers perspective, who use the observations from this process to design their system in such a way that the system can adapt itself depending on the changes in the system context, ii) the service providers perspective, who use the observations from this process to tune the system at runtime to improve the quality of the service in different context.
What	Subject	This monitoring process monitors the context of a system, where the context of a system can be categorized as i) external or physical context that can be measured by hardware sensors, i.e. location, light, sound, movement, touch, temperature, air pressure etc. ii) internal or logical context that is mostly identified by the user interactions with the system, e.g. the user's goals, tasks, work context, business processes, the user's emotional state etc.
	Aspect	Context monitoring is applied to monitor a wide range of system requirements in different context, including i) Maintainability, i.e. the system should be able to easily modify or adapt to changing environments, ii) Portability, i.e. the system should be easily run under different computing systems.
	Property	Context information is structured in a machine readable form, which is known as context model. A context model describes a context, in terms of various attributes e.g. context category like temperature, speed, context value i.e. the raw data gathered by the sensor, time stamp i.e. when the context was sensed, and context source. Context properties are logical rules, i.e. a set of conditions defined over context attributes in context models.
How	Methodology	Context information is gathered either from the sensors or from the users and this information is matched against the specified properties to detect a change in the context
	Architecture	Depending on the architecture of the context aware application, context monitoring process can assume three types of architecture, i) Direct sensor access: where monitor collects context information directly from hardware sensors. ii) Middleware based: in this approach a component acts as a middleware to collect context information from distributed sources. This is the most dominant architecture in context monitoring process. iii) Context server: this approach extends the middleware approach where context server stores the collected context information to allow concurrent multiple access to the information
	Implementation	Collection and processing of context information can be implemented in different ways. For example, physical or hardware sensors can be used to capture almost any physical data. Virtual sensors collect context data from software, e.g. location of a user can be determined by browsing the electronic calendar of the user. Logical sensors apply logical reasoning on physical sensor and virtual sensor data to infer context information.

Monitoring Grid infrastructure

Grid infrastructure monitoring from its very beginning has been focused on collecting generic attributes of its participating resources. These attributes are often captured and expressed in specific information

Table 3.9: Monitoring Grid infrastructures

Dimension	Concepts	Description
Why	Usage	Monitoring grid infrastructure offers information which helps the *optimization* of the usage grid sites and the services they offer. For example the tasks of grid workflows can be mapped to different sites based on data collected from the infrastructure monitoring. The other main use of the grid infrastructure monitoring is the *evaluation* of the service quality of the different grid sites. In this case the reliability of a grid site is calculated on historical data collected by the monitor.
Who	Actors	Infrastructure monitoring in the grid uses the *provider* perspective because the information offered is restricted by grid site owners and the maintainers of the infrastructure monitoring system. The information is offered on multiple levels. Grid sites monitor themselves, and provide the results to for higher level grid infrastructure monitors. These monitors act as *consumers* while aggregating the individual site results, but they also provide the aggregated view for even higher level services. Infrastructure monitoring is frequently done without any specific *requester* because the monitored data is necessary at least for the archiver which supports the site evaluation.
What	Subject	The grid fabric is monitored during infrastructure monitoring. In an SBA this fabric acts as the context, therefore this monitoring area mainly observes the service environment where the SBAs services can be placed or the different workflows can be mapped. However in certain Virtual Organization level monitoring systems might monitor the health of specific applications resulting the monitoring of the SBA class of that application.
	Aspect	Behavior, health, availability and reliability of the different services offered by the grid sites.
	Property	The properties of the SBA in grid are always monitored in specific time instances, therefore the monitored data is usually time-stamped. These instantaneous properties are aggregated later on or in case of reliability measures of the given sites they can also be derived.
How	Methodology	Lower level monitors collect the information by polling, meanwhile aggregating these results usually means the lower level monitor pushes the collected data to higher levels, as a result grid or VO level monitoring systems present the monitored information with latencies.
	Architecture	The monitoring system is distributed throughout all the grid sites, and aggregation is done on specialized nodes which are usually configured to host only the monitoring data. Grid monitoring is focusing on the service infrastructure level only.
	Implementation	Basic information sources are probes, however in several systems the monitoring could be offered by the monitored system itself, and therefore the monitor could receive messages on changes in the monitored properties.

model which is a formal description of the monitored entities of a Grid. Typically the information model is expressed in UML, like the GLUE Schema v1.3 [4] (the most frequently used model in the current grid deployments), the OGF reference model (currently in draft) or the Distributed Management Task Force's (DMTF's) Common Information Model (CIM) [50]. In the actual information exchanges between monitoring information producers (e.g., monitoring providers) and consumers an XML representation of the information model is often used.

The data collected by the infrastructure level monitoring can also be archived for further use. In this case the archives can be used to calculate the reliability of the Grid resources. This data is frequently used by system administrators to monitor their sites compliance with the level of service offered.

Another main use of archived data from infrastructure level monitoring is in the decision support for higher-level Grid services such as Grid brokers or meta-schedulers. In order for these higher-level services to make the best decisions the infrastructure-level monitoring system of a Grid should allow access to the necessary information required to make each decision. For example in case of schedulers, for them to make a decision about where to send a users job execution request the monitoring system should present the state of the whole Grid as accurately as possible. Monitors supporting Grid schedulers are often hierarchical with higher level monitors aggregating the information coming from lower levels. This aggregation is made in a 'pull' fashion when the lower level monitors are queried only in cases the higher level monitor has enough resources to handle the incoming data meanwhile still serving requests about the current state of the grid.

Several levels of monitors are known:

Site level monitoring is the primary information source in the grid. This kind of infrastructure monitoring system is usually centralised on the grid site, and provide the most up-to-date information in the grid. In some monitoring systems these monitors cannot be accessed directly but by a higher level monitoring component such as the Virtual Organization (VO) or Grid level monitoring systems. However in case they can be accessed, they are accessed directly by schedulers to make more accurate decisions. This way the scheduler makes the decision on two level, first it filters the vast amount of sites available in the VO or Grid level systems and secondly it queries the most promising sites for the most actual monitoring data to make an informed decision between the filtered sites.

VO level monitoring aggregates information about all resources from a given VO. In several cases VOs offer special monitors which can for example provide availability information of the supported software of a given site. This is useful for the members because they can decide what sites they should favour when they have specific needs. For example in case they need to run an MPI application they just check which sites are reported to have a working MPI installation.

Grid level monitoring is the highest, it provides the overall state of the grid, aggregates all data from the lower level monitoring systems, usually built on top of VO level monitors, however in some cases this monitor might aggregate data directly from site level monitors. The data cached in this monitor, therefore the data available should be time-stamped to present its accuracy.

Monitoring applications on the Grid

From the infrastructure point-of-view the execution of an application is built up from several distinct phases, such as transferring its input files to the target worker node or waiting in a queue of the local resource management system. However even the most sophisticated infrastructure level monitors cannot tell what is happening during the execution of the application after it has reached the worker node until the execution finishes successfully (or, perhaps, unsuccessfully).

Users frequently need more information about the execution of a long running program, however. When an application is executed *interactively* the user can observe directly progress reports of the execution and is capable of steering the execution according their needs.

A more demanding case for more detailed execution state reports is *support for migration*. If an application can resume its execution from a checkpoint, then the monitoring system can carry out regular checkpointing (the process of taking a checkpoint). These checkpoints can help, for example, Grid schedulers in load balancing as they can be used to pause and then migrate applications to less loaded resources. This class of applications can store their output on remote storage, which means even though there were some migrations on the application, its data can be downloaded from the same logical location from the user point of view.

Finally, *application instrumentation* pushes Grid monitoring systems to their limits by generating vast amount of monitored data, which has to be delivered to a specific location (usually this final destination is the user's computer). In this case the application is prepared to push its reports about its internal behaviour, when it is running in instrumented mode. As it can be expected this behavioural reporting

Table 3.10: Monitoring applications on the Grid (MAG)

Dimension	**Concepts**	**Description**
Why	Usage	When applications are monitored in grid then usually the users would like to be informed about its detailed progress. Based on this progress information they can steer the execution in order to make sure the results of the application will be usable and it is not just wasting the resources with unnecessary computation. Other main use of application monitoring is the diagnosis, debugging or correctness of the application.
Who	Actors	Application monitoring uses the client's perspective, thus it monitors only those properties the user is interested in, because it would not be feasible to provide all possible application progress information during runtime.
What	Subject	in MAG one instance of the SBA execution is targeted during its execution. It is also frequent that they use grid infrastructure monitors to have the contextual and instance information at the same time.
	Aspect	Progress, application state, behavior, adaptability of the currently executed application.
	Property	Functional properties of the application can be measured according to the definition of the user, these are usually represent internal basic properties. The properties are measured by the user's own ways and the monitoring system only offers the infrastructure to deliver the necessary information.
How	Methodology	The application pushes its internal behavior information to the monitoring infrastructure, which has the task to deliver it as soon as possible to those peers who have expressed their interest in the applications execution.
	Architecture	The MAG is distributed on all sites of the grid and they can receive the monitoring events fired by the application which needs to be instrumented on the necessary level. The level of instrumentation defines the granularity of the information which is sent to the listeners of the application's execution events.
	Implementation	MAG usually has three distinct components. First it is composed of the eventing libraries which can send new data to the listeners, they are the primary data sources. Secondly they MAG systems usually accommodate a cacheing architecture close to the sources in order to avoid sending through the network small packets of monitored data with high latencies. Finally the listener is a component which is connected to all used caches of the SBA and periodically retrieves their content to present it for the users.

needs a new monitoring infrastructure which make sure the monitored data is not sent across the grid in small packages when they occur. Instead they are usually cached on the worker node which executes the application, and sent regularly off site to the place where the information is collected. The amount of information available makes it possible to use the monitoring data in distributed debugging, profiling or even post-mortem analysis, thus this kind of monitoring is used only by the developers and the shipped application usually uses one of the previously mentioned monitoring solutions only.

As it can be seen the caching behaviour of the high level infrastructure monitors is all time required in case of application monitoring. However the monitoring events are now buffered more closely to the execution. Instead of the pull mode delivery used these systems use push mode delivery. The event initiates the forwarding of the current buffer is usually related to its fullness. Usually the user's machine is used as a the top level monitor in order to avoid the overfilling generic top level monitors.

3.3.5 Monitoring in Component-Based Systems

In component-based system, we can find three kinds of approach *system monitoring*, *context monitoring to raise adaptations* and *monitoring for validation* that target three main motivations. The system monitoring consists of a monitoring of components and execution platform. It is used as an internal mechanism to extend platforms capabilities, or adapt the system behavior. Context monitors are used for

Table 3.11: Using monitors to extend component-based frameworks

Dimension	**Concepts**	**Description**
Why	Usage	Component-based applications and platforms are monitored to allow their extention. Component properties are monitored to detect execution scenario or component activities that can highlight a need to weave new aspects on some components of the system to extend its behavior, or extend the capabilities of the platform.
Who	Actors	In the context of platform extension, the monitoring can be made by the platform itself, or by a specialized component on the platform, and aims to track some situations specified by the application designer and then trigger some extension mechanisms.
What	Subject	This kind of monitoring aims to intercept some execution flows. That is why the components and the execution platform are monitored.
	Aspect	In order to extend the platform or the application behavior, the monitoring have to target components internal changes and component composition changes, to detect new needs of the system.
	Property	The properties monitored on components are life-cycle state, components bindings, their contents, names and attributes. Messages exchanged by components through the platform are also monitored to manage execution flows.
How	Methodology	Monitoring is passive. It just pays attention on components intenal changes, and on binding events, and gathers information to react on some changes.
	Architecture	The monitoring is cross-layered, because changes can appear due to platform behavior modifications, components composition changes or on the composition model (in a model@run-time perspective).
	Implementation	Monitoring mechanisms are deployed inside the execution platform or are part of a specific component. We can also imagine that each components implements some monitoring tools.

the development of context-aware component-based systems. Finally, the validation monitors are used to test, diagnose and debug component based systems.

Using monitors to extend component-based frameworks

To give a more precise view of the monitoring mechanisms in a component-base system extension perspective, here is a description of the Fractal Component System [29].
A Fractal component is formed out of two parts: a controller (also called membrane), and a content. The content is composed of (a finite number of) sub-components managed by the controller of the enclosing component. The controller of a component embodies the control behavior associated with the component. In particular, a component controller can:

- Provide an explicit and causally connected representation of the component's sub components;
- Intercept incoming and outgoing operation invocations targeting or originating from the component's sub components;
- Superimpose a control behavior to the behavior of the component's sub components, such as suspending, check pointing and resuming activities of these sub-components.

The controller can be accessed through a set of so-called control interfaces which manage the non-functional properties of a component such as its life cycle, bindings, content, name and attributes. This set of control interfaces can be extended with new control interfaces that can be added to a component controller. The interception mechanism reifies messages sent by and received on component interfaces. These messages can be modified, discarded or delivered to the component.
Several works have extended the controller to monitor the component activity to be able to extend its

Table 3.12: Monitoring of Context in Component-based Systems

Dimension	Concepts	Description
Why	Usage	Context-awarness abilities are interesting to diagnose and optimize the behavior of a component-based application. It makes it possible for it, to adapt its behavior to environement or execution flow changes, in order to ensure a service level agreement.
Who	Actors	The application itself monitors its context.
What	Subject	The monitoring concerns the ressources of the platform and the running system, and also user inputs that can lead to a context change.
	Aspect	The monitoring aim to reveal time and space changes of the platform or the system, and availability of external ressources.
	Property	Available free space quantity, processor disponibility or external ressource response time are examples of properties. In fact all system and platform properties that do not diectly concerns the application behavior may be of interest for context monitoring.
How	Methodology	Data gathering is made in push mode by non-blocking mechanisms. Events from the platform, the components and the system are collected and component-based system adaptations may be triggered if the situation fits predefined conditions.
	Architecture	Monitoring may be centralized in a component, or in the platform implementation and is cross-layered to get all context changes.
	Implementation	Monitors are implanted in each components or in a specialized one.

behavior. For example, in [98, 106], Pessemier et al have shown how they can monitor the component activity to weave new aspects that modify the system behavior. In [114], Vanderperren et al presents an approach to monitor event in component-based system Jasco and weave aspects when a specific scenario occurs. In the Spring framework[1], a built-in interceptor mechanism through proxy generation is used to monitor and extend the system behavior.

Monitoring of Context in Component-based Systems

The monitoring of context in a component-based system is an essential ability which makes it possible for the system to adapt itself according to context changes. The term context-awareness was introduced by Schilit et al. in ”Context-Aware Computing Applications” (in 1994 IEEE Workshop on Mobile Computing Systems and Applications). Dey, in ”Understanding and Using Context” (in Personal Ubiquitous Computing), defined context as ”any information that can be used to characterise the situation of entities”. So, Context-Aware Applications adapt their behavior based adaptation policies, by gathering information from their environment. Such information can be collected from system properties, user inputs or different sensors, and renders the creation of large scale context-aware application difficult because of two things: first, the number of data sources makes it hard to maintain a proper organisation, second, because the gigantic number of data produced makes it hard to extract meaningful information.
A recent research prototype called WildCAT [46] eases the creation of context-aware systems. WildCAT copes with issues such as number of data sources and the gigantic amount of data monitored by providing a user with easy access to sensors through a hierarchical organization. WildCAT provides the power of an SQL-like language to express conditions upon which adaptation mechanisms can be triggered.
In [43], the authors introduce a three-step cycle to efficiently process the context information. They evaluate the *caching/off-loading* adaptation mechanism in COSMOS framework. The adaptation mechanism processes the information coming from the component framework. Context monitoring is gaining importance and will be a important subject in the next few years in proactive control and adaptation of systems as discussed and tested in [113].

[1]http://www.springframework.org/

Table 3.13: Monitoring to Validate Component-Based Systems

Dimension	Concepts	Description
Why	Usage	Run-time correctness, diagnosis and optimization are parts of component-based applications validation. Monitoring is essential to detect and/or prevent application failures, and ensure the application behavior.
Who	Actors	Designers offten simulates workflows from clients or third party to monitor the application behavior and improve it. At runtime, the provider places monitors to detect or diagnose application falures.
What	Subject	Platform reactions are monitored as components and application are.
	Aspect	The stress is placed on dependability, performances an usability of components and platform.
	Property	Interesting properties can be component response time, output correctness, cross-dependency between components or availability.
How	Methodology	The application can be monitored at design time in simulation mode. This allow a fine grained monitoring of the global behavior of the system. When deployed, the monitoring is mainly made in push modeto prevent failures.
	Architecture	In order to validate applications, the monitoring have to be made at all levels : component by component, on components composition and on the platform.
	Implementation	Each component can monitor its correctness, and a specialized one can monitor allication behavior.

Monitoring to Validate Component-Based Systems

Validation of software components is very important for the overall Quality of Service of component-based software systems. To validate a component-based system we test, diagnose, and debug individual and integrations of components in the system.
Testing attempts to check the behavioral correctness of component-based system by first independently checking the correctness of individual components, and then checking the correctness of the composed system. If testing leads to detection of faults or abnormal behavior, the next step is to isolate the fault in the entire system.
Diagnosis of a component-based system is the process by which we aim to find the root cause of a failure. Generally, a set of possible suspects is postulated, analyzed and reduced, as a result of the application of test procedures, observations or other evidence. Diagnosis also help automate the ordering of a testing strategy to address all potential test results and evidence. Diagnosis detects components that are suspect possible causes, and those components that are exonerated by the test outcome. The set of suspect components contains those which have failed, and some which may or may not have failed. A subsequent application of a different test can similarly reduce the set of suspect components until all suspects are known to have failed, and the cause of the system failure has been diagnosed.
Once, a fault has been localized in a component-based system it is essential to debug the system. The debugging includes the replacement of components by linking the main process to other functioning/available components. For instance, if a component is no longer available or operational we replace it with a component with similar behavior.
Components that perform validation include test execution units and monitors, diagnosis and debug components. We briefly describe each of these validation related components below:

- **Test Execution Units and Monitors**. A test execution unit invokes a component-based system with a sequence of test inputs such as timed test sequences [112]. A test monitor component validates the execution of a test sequence to a system. A test execution unit requires several test cases for execution. Such test cases can be automatically generated[107] or specified manually by an expert. The test monitor unit is an oracle that validates the execution of a test case. The test monitor can compare an expected output with the output of a component-based system, it can perform mutation analysis [92] to see how many bugs a test case is able to detect, or it can verify the output against a post-condition. A test monitor informs a diagnosis monitor about the correct/incorrect execution of a component.

- **Diagnosis Components**. The diagnosis component attempts to find the root cause of a problem in a component based system. The faulty execution of a test case validated by a test monitor unit is the information used for diagnosis. The faulty execution of a component may not originate in the component but due to propagation of a fault from an other component. A diagnosis component creates, for example, diagnostic decision trees [10] to summarize evidence received from test monitors. The diagnosis component also measures the reliability of a test monitor and proposes replacement of monitors.

- **Debugging Components**. The management [45] or the debugging of a component based system based on a diagnosis involves the use of optimization algorithms to reconfigure the system. The replacement of a components is guided by QoS [45]. Several machine learning based approaches are available to reconfigure component based systems. In [122] the authors employ reinforcement learning, [108] presents a self-repair algorithm for managing component-based systems. In [77] the authors represent a model-driven recovery algorithm to perform single step recoveries in a component based system. In [49] the authors present a debugging approach to detect structural inconsistencies in declarative description of a component-based system.

3.4 From Monitoring to Adaptation

According to the conceptual framework shown in Figure 2.1, S-Cube vision explicitly states that monitoring and adaptation are related according to a cause-effect relationship. As previously introduced, monitoring is defined *as a process of collecting relevant information in order to evaluate properties of interest over analyzed system and report corresponding events*. As a consequence, these *Monitored Events* (the “what” in the monitoring) are made available to SBA in order to improve the service provisioning. Usually adaptation is required when the monitored event detects anomalous situations, so these events drive the definition of the *Adaptation Requirements* that, in turn, drive the adaptation execution according to one of the mechanisms introduced in the next chapter.

A formal definition of what to be monitored and how to adapt the service execution are usually formalized in a contract. In a contract, the quality of the service provisioning is specified in terms of SLA. Here, a set of parameters are identified to describe the quality of the service along with the values for these parameters that are considered acceptable for the parties. According to the S-CUBE project deliverable CD-JRA-1.3.2 a set of quality dimensions that are considered relevant in a SBA are listed. A contract about the provisioning of a SBA should include these dimensions as a way to formalize the agreements between users and providers.

In addition, a contract includes elements related to the monitoring and adaptation. About the monitoring, the contract defines “who” is in charge of performing the monitoring actions, “what” has to be monitored, and, in some cases, also “how” to monitor. About the adaptation, a contract can also specify who is responsible for the adaptation in case the service provisioning is not compliant with the SLA. In service oriented computing, WSLA [79], and WS-Agreement [64] provide some description models to express contracts. WSLA allows providers to define quality dimensions and to describe evaluation functions. Furthermore, it provides monitoring of the parameters during operations and invocation of recovery actions when contract violations occur. Similarly, WS-Agreement provides constructs for advertising the capabilities of providers and for creating agreements based on creational offers, and for monitoring agreement compliance at runtime.

According to a contract that holds between two parties, on first approximation adaptation is required for two main reasons: (i) adaptation due to lack of conformance and (ii) adaptation due to requirements changing. In more detail, in the first class, we include the adaptation mechanisms for optimize, recovery, or prevent discrepancy between expected behaviour and the actual behaviour of the SBA. In the second

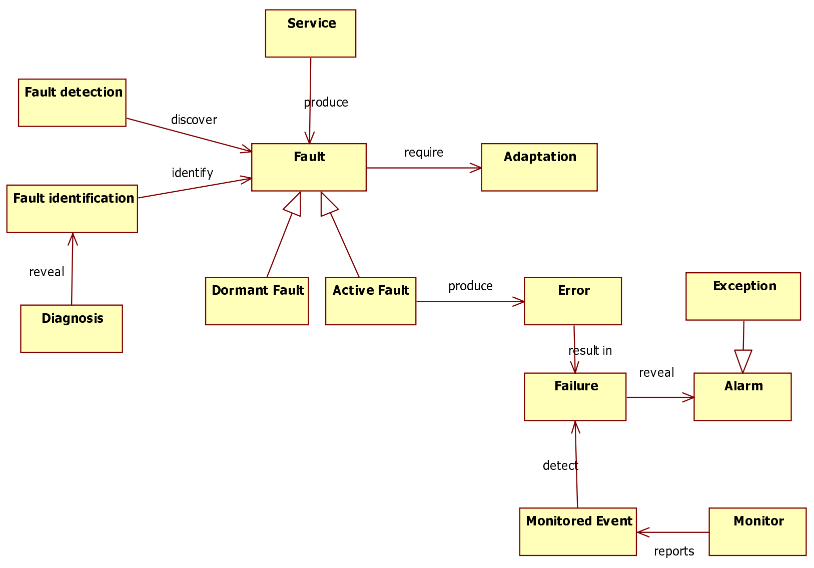


Figure 3.3: Failures and faults

class, we include the adaptation for interoperability, customization, and context matching where the SBA behaviour is fine, but the user requirements might change at run-time. As discussed in the following, the lack of conformance implies that monitors provide information about a possible misbehaviour of SBA. As a consequence, the adaptation takes place to proper repair the service execution. On the other hand, the requirements changing implies the monitoring of the user context in order to identify changes that have impact on the service execution. In this case, adaptation needs to redefine the service execution in order to meet the new requirements.

3.4.1 Lack of conformance

According to the literature in dependable systems, the lack of conformance results in system failures defined as “a discrepancy between the delivered service and the correct one” [75]. Following the terminology introduced in [75], a failure is the visible effect of an error and it is manifested to the SBA by the monitor that includes it in the list of events monitored.

As shown in Figure 3.3, in case of system malfunctioning, a fault occurs and it causes an error which, in turn, is manifested as a failure. A fault can be either active or dormant. In the former case the fault produces an error; in the latter case does not. This process is usually called the “Fault-Error-Failure Chain” and produces data that are relevant only at monitoring time. Indeed, the adaptation needs information about the fault and not only about the failure. As a consequence, a fault identification activity is required to discovery of the identity of the occurred fault and the diagnosis is the process by which this is achieved. In some sense, the diagnosis is able to pass trough the Fault-Error-Failure chain in the reverse order.

It is important to note that – for the purpose of this deliverable – we have adopted the terminology of [75] and [96]. In the software engineering discipline different definitions for the terms are found [69]. Here, an error is the cause for a fault (i.e., a defect in a software artifact). This defect can lead to a failure of the running software system.

At the end of the diagnosis, the adaptation mechanisms have the required information to enact the most suitable strategy. In some case, according to the adaptation strategy adopted, this information can also be exploited to avoid possible future critical situations. Thus, adaptation can follow a pro-active approach.

3.4.2 Requirements changing

User' requirements might change at run-time for several reasons. The user realizes that the quality of the service the user asked before is not enough to fulfill the real goal, or the environment in which the user is working changes. In the latter case, mobile devices are usually involved since they allow the user to move around while consuming the services. In case the functional, or even the non-functional, characteristics of the service depend on the environment (i.e., the context) in which the service is consumed, then the user requirements might change at run-time and the SBA needs to react properly.

Requirements changing can be also inferred starting from the actions taken by the user in the past. As discussed in the Section 3.3.3, the monitoring of HCI aims to reason about the sequence of actions taken by the user in the system and helps to better understand the future action of the user. With respect to the previous case, in which the context is analyze, and the adaptation is a reaction of a requirement changing, in this case, the adaptation is proactive: the system aims at anticipating the user intentions.

Chapter 4

Taxonomy of Adaptation Principles and Mechanisms

The dynamic nature of the business world highlights the continuous pressure to reduce expenses, to increase revenues, to generate profits, and to remain competitive. This requires Service-Based application to be highly reactive and adaptive. In fact, SBAs are subject to constant changes and variations: constituent services can evolve due to changes in structures (attributes and operations), in behavior (when services are interacting) and policies; agreements between the interacting parties may change; the requirements and the SBA context may evolve as well. Such changes can be identified, detected, and foreseen in the SBA during the monitoring of the application execution and its environment. In order to accomplish this, SBAs should be equipped with the corresponding mechanisms to meet changing requirements

As mentioned in the previous chapter concerning the monitoring part, in order to provide a holistic, comprehensive, and integrated vision on the monitoring and adaptation across various research disciplines, in the following we will try to present a generalized and universal yet practical definition of the adaptation problem. We will present a generic conceptual model for the adaptation, which refines an overall adaptation and monitoring vision adopted within S-Cube research project. Based on the conceptual model, this chapter will provide a classification of the adaptation concepts, and instantiate this classification across relevant research disciplines.

4.1 Conceptual Model

Adaptation can be defined as a process of modifying Service-Based Application in order to satisfy new requirements and to fit new situations dictated by the environment on the basis of Adaptation Strategies designed by the system integrator. An Adaptable Service-Based Application is a service-based application augmented with the corresponding control loop that monitors and modifies itself on the basis of these strategies. Notice that adaptations can be performed either because monitoring has revealed a problem or because the application identifies possible optimizations or because its execution context has changed. The context here may be defined by the set of services available to compose SBAs, the computational resources available, the parameters and protocols being in place, user preferences, environment characteristics.

High-level conceptual model of the adaptation concepts is represented in Figure 4.1.

The *Adaptation Requirements* identify the aspects of the SBA model that are subject to change, and what the expected outcome of the adaptation process is. *Adaptation Strategies* are the ways through which the adaptation requirements are satisfied. Examples of adaptation strategies are *re-configure* (i.e., modify the current configuration parameters of the SBA), *substitute* (replace one constituent service with another), *compensate* (remove the negative effect of the previously executed action by performing new

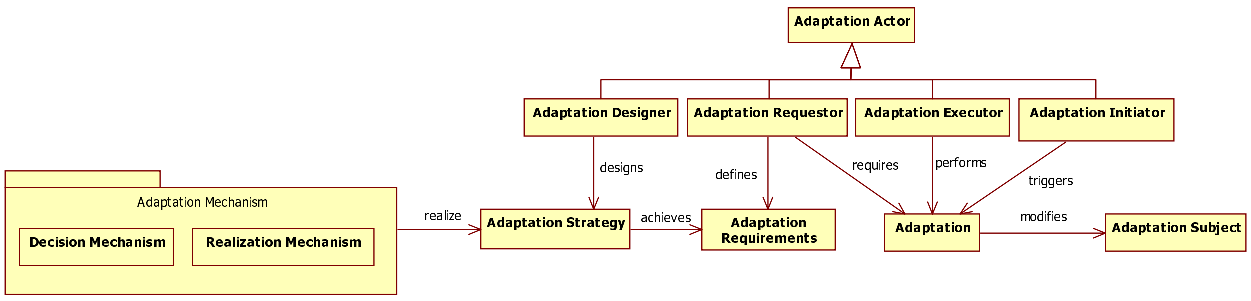


Figure 4.1: High-level adaptation model

actions), *re-plan* (modify the structure and the model of the application, which is more suitable for the current situation), *re-compose* (modify the way the services are composed), and *re-negotiate* (modify the service-level agreement with the service provider).

Adaptation Strategies are realized using the available *Adaptation Mechanisms*. These mechanisms include the tools for performing actual adaptation actions, i.e., *Realization Mechanisms*, and the tools for making important decisions about the adaptation, i.e., *Decision Mechanisms*. The latter include the mechanisms for selecting adaptation strategies among possible alternatives given the current situations, histories of previous adaptations, user decisions or preferences, etc.

The adaptation procedure may modify various elements of the SBA, i.e., may have different *Adaptation Subjects*. The adaptation process involves different kinds of *Adaptation Actors* covering various roles with which the users may be involved in the process. When these roles are performed by the corresponding software components, we speak about self-adaptation approaches.

Below we will provide a classification of the adaptation problem and identified adaptation concepts.

4.2 Adaptation Taxonomy

In this chapter, we describe a graphical representation of the adaptation taxonomy (depicted in Figure 4.2) that distinguishes approaches by **Why**, **Who**, **What**, and **How** software adaptation takes place.

4.2.1 Taxonomy Dimension: Why?

The first dimension of our taxonomy define the usage of the adaptation process, i.e., **Why** adaptation is needed. Indeed, the **Why** dimension provides a description of the motivation for the adaptation.

Depending on the goal of the adaptation process, one can distinguish between

- *Perfective Adaptation*, which aims to improve the application even it runs correctly, e.g., to optimize its quality characteristics.
- *Corrective Adaptation*, which aims to remove the faulty behavior of a SBA by replacing it by a new version that provides the same functionality. Various faults can occur relatively often and unexpectedly in distributed systems. It is therefore necessary to handle failures reported during execution of the SBA in order to recover from undesired behavior, or to change the application logic in order to remove the possible fault.
- *Adaptive Adaptation*, which modifies the application in response to changes affecting its environment. The need for this kind of adaptation in SBAs is dictated by (i) the necessity to accommodate to the changes in the SBA context (execution context, user context, or physical context); (ii) the need to ensure interoperability between interacting parties by providing appropriate adapters or

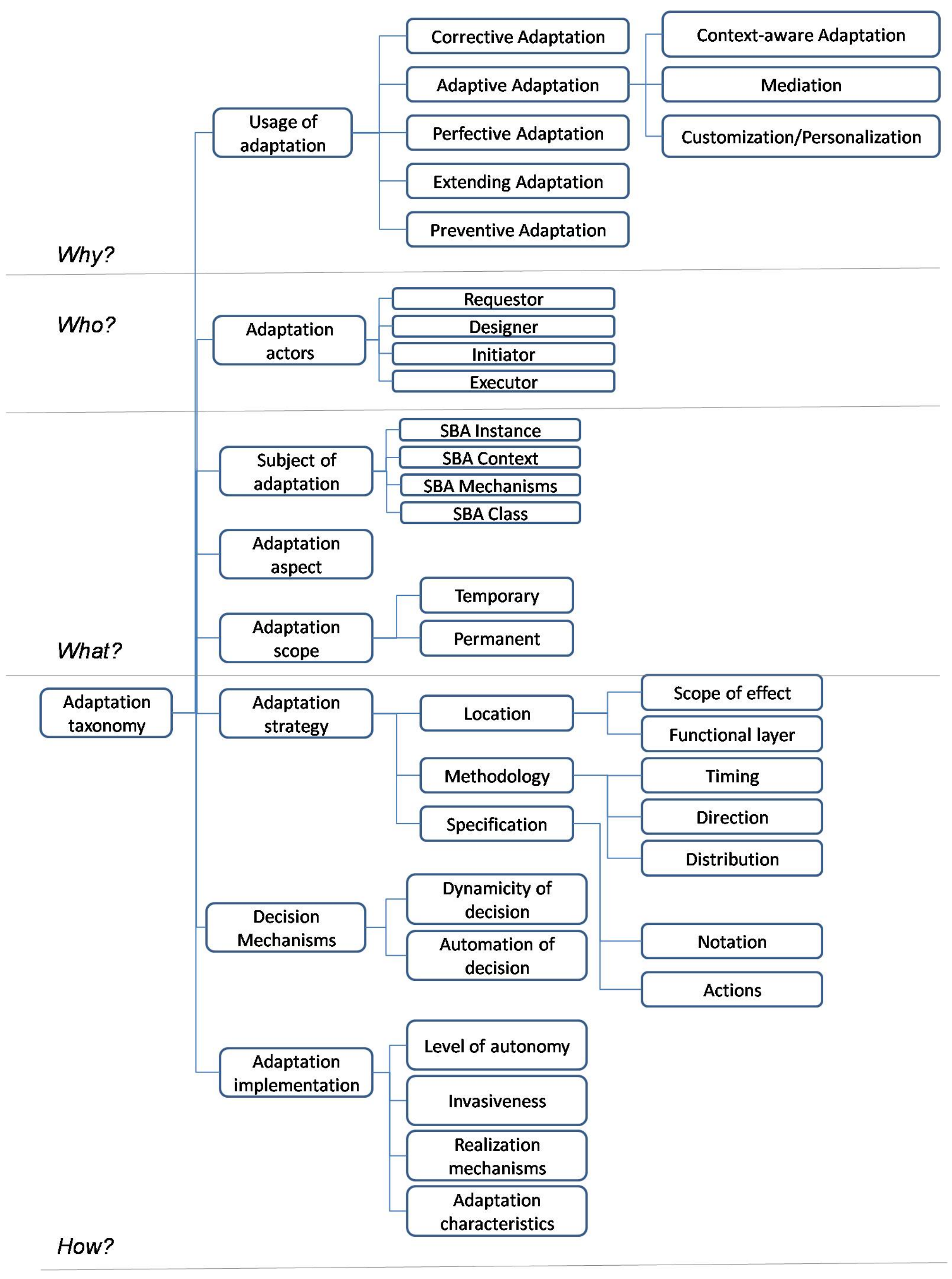


Figure 4.2: Adaptation taxonomy

mediators; (iii) the necessity to customize or personalize the application according to the needs and requirements of particular user or customers.

- *Preventive Adaptation*, which aims to prevent future faults or extra-functional issues before they occur.

- *Extending Adaptation*, which extends the application by adding new needed functionalities.

These classes may be further decomposed given particular problems in mind. For example, Adaptive adaptation may be classified into Context-aware adaptation, mediation, and customization/personalization.

4.2.2 Taxonomy Dimension: Who?

The **Who** dimension characterizes the adaptation problem from the view of the different actors (software or human) involved in the adaptation process. One can distinguish the following actors:

1. *Adaptation Requestor* characterizes the stakeholders, who defines the adapttion requirements for the SBA.
2. *Adaptation Designer*, who defines the adaptation strategies to achieve the adaptation requirements.
3. *Adaptation Initiator*, who initiate the modification of the application in reaction to the identified changes.
4. *Adaptation Executor* is responsible for executing adaptation actions defined by the chosen adaptation strategy.

When certain roles are performed by the SBA or the environment autonomously, we speak about *self-adaptation*. Otherwise, the adaptation is referred to as *human-in-the-loop adaptation*. When the human plays a role of the adaptation executor, we can speak about manual SBA adaptation, performed, e.g., at design-time.

4.2.3 Taxonomy Dimension: What?

The **What** dimension is used to classify the adaptation target and the expected result. In this way, we consider the following elements of the taxonomy: Subject of adaptation, Adaptation Aspect, and adaptation scope.

With the **Subject of Adaptation** we mean an entity that should be modified by the adaptation process. At the highest level of abstraction we distinguish

- *SBA Instance*, i.e., business process instance, an application customized to a particular user according to her user profile, a particular configuration of a service;
- *SBA Class* that define the whole application model, including its business process model, business requirements and KPIs;
- *SBA context* encompasses various aspects, i.e., user/physical/computing environment in which the application is performed;
- *Adaptation and Monitoring Mechanisms* themselves, changing the way the system is changed and managed.

Finer granularity may be thought of, such as services, compositions, rules and policies, SLAs, etc.

With the **Adaptation Aspect** we refer to a particular concern of the adaptation process: different dimensions of the SBA quality model (e.g., security, dependability, usability), functionality, HCI aspects, etc.

With the **Adaptation Scope** we refer to the effect of the adaptation process, i.e., whether it is expected to be temporary (i.e., hold only to a particular SBA instance or in a particular context) or permanent adaptation (i.e., modify the whole application model that will be applicable to other instances and situations).

4.2.4 Taxonomy Dimension: How?

The third dimension of our taxonomy is **How** adaptation can be achieved and implemented, that is, what the specific strategies are exploited and what the specific mechanisms are used to implement the,. This dimension includes the characteristics of the relations established between the monitoring artifacts and the changes of SBA addressed by the approaches; e.g., models, types, granularity,etc.

Adaptation Strategies

Adaptation strategies are the means through which adaptation is accomplished. Examples of adaptation strategies are re-configuration, re-binding, re-execution, re-planning, etc. Adaptation Strategies define the possible ways to achieve Adaptation Requirements and Objectives given the available Adaptation Mechanisms. They may be classified according to a set of characteristics, including the placement (location) of changes, the used methodology, and the way the strategy is specified.

Location determines the placement of the changes in the SBA architecture and environment:

- Scope of adaptation effect ("horizontal" placement) says whether the changes are *local* (shallow), i.e., the small-scale incremental changes localized to a service or are restricted to the clients of that service, or whether they are *global* (deep), i.e., large-scale transformational changes cascading beyond the clients of a service possibly to entire value-chain (end-to-end processes) - clients of affected services e.g., outsourcers or suppliers.

- Affected functional SBA layers ("vertical" placement), where one can distinguish between *Service Infrastructure*-level changes, when the changes affect service signatures, protocols, and the runtime environment of the service execution; *Service Composition*-level changes, when the behavioral protocols and/or operational semantics of SBA are affected; *Business Process*-level changes, when the change involve business rules and requirements, organizational models, clients, and even entire value chain; *Cross-layer* changes affect different functional layers.

Adaptation Methodology characterizes the timing, the distribution, and the direction of the adaptation.

- *Timing* defines the moment of time when the adaptation is performed. *Reactive adaptation* refers to the modification in reaction to the changes already occurred; *proactive adaptation* aims to modify SBA before a deviation will occur during the actual operation and before such a deviation can lead to problems; *post-mortem* adaptation is characterized by a significant gap between the triggering event is detected and the modification performed. Typically, the post-mortem adaptation is accomplished by re-designing/re-engineering the application.

- *Direction* of the adaptation distinguishes between *forward adaptation*, where the adaptation strategy that directs the system to a new state, where the adaptation requirements are met, and *backward adaptation*, where the adaptation strategy reverts the system to a state, previously known to meet the adaptation requirements.

- *Distribution* of the adaptation distinguishes between *centralized adaptation*, where the actions are defined and executed on all the affected components in the controlled and integrated way, and *distributed adaptation* performed locally and then propagated among components.

Adaptation Specification represents the notations needed to specify the strategies and the particular actions representing those strategies. It can range from procedural approach (concrete actions to be performed), over declarative (the description of the goals to be achieve), to hybrid. The notation may be *implicit*: in this case the adaptation strategies and actions are hard-coded within the system according to some predefined schemata and can not be changed, without modification of the adaptation mechanism. *Explicit* adaptation specification, on the other hand, allows the designer to guide or influence the adaptation process by explicitly stating the adaptation requirements or instructions. The following forms of explicit adaptation specification may be considered:

- *action-based* specification consists of situation-action rules which specify exactly what to do in certain situations;

- *goal-based* specification is a higher-level form of behavioural specification that establishes performance objectives, leaving the system or the middleware to determine the actions required to achieve those objectives;
- *utility function-based* specification exploits utility functions to qualify and quantify the desirability of different adaptation alternatives, and, therefore, permit, on the fly, determination of a "best" feasible state;
- *explicit variability* approach associate the situations, where the adaptation should take place (adaptation points), with a set of alternatives (variants) that define different possible implementations of the corresponding application part.

Adaptation action is an action performed over an adapted system with the purpose of changing it according to the adaptation requirements. Adaptation action defines an operation semantics of the adaptation strategy. Different approaches define various adaptation actions. Those actions may be further classified according to the subject of the adaptation and the scope: for example, service instance adaptation actions (*retry*, *negotiate SLA*, *duplicate service*, *substitute service*), flow instance adaptation actions (*substitute flow*, *redo*, *choose alternative behavior*, *undo*, *skip / skip to*, *compensate*), service class actions (*change SLA*, and *suggestion for service re-design*), flow class actions (*re-design/re-plan*, *change service selection logic*, *change service registry*, *change platform*).

Decision Mechanisms

Decision Mechanisms are the means through which adaptation approach may make a decision on the strategy to be performed in a given situation in order to better satisfy the adaptation requirements. The mechanisms are characterized by the dynamicity and by the automation of strategy selection.

Dynamicity of decision refers to the flexibility, with which the adaptation approach may decide on the strategy to be applied. One can distinguish: *static selection*, when the adaptation strategy is predefined and explicitly associated with the given adaptation requirement, situation or event; *dynamic selection*, when the adaptation strategy is selected at run-time based on a concrete situation, information, and context properties; and *evolution-based selection*, when the adaptation strategy is chosen taking into account not only the current situation, but also the history of previous decisions, adaptations, and their results.

Automation of decision characterizes the degree of the human involvement in the decision process. The degree can range from totally *automatic* (no user intervention is needed), to *interactive* (where the user makes the choice).

Adaptation Implementation

Adaptation implementation defines the way the adaptation methodology and architecture are realized. It is defined by the autonomy of the execution, invasiveness of the framework, realization mechanisms, and by specific characteristics of the approach that allow one to "measure" the approach.

Autonomy characterizes the involvement of the human in the adaptation execution. It can be done in a *autonomous* way (self-adapt), *manually*, or in an *interactive* form, where the execution of adaptation actions requires human involvement.

Invasiveness characterizes the adaptation framework from the perspective of how tightly it is integrated with the subject of adaptation and the execution framework. We distinguish between the cases, when the adaptation facilities are *integrated with the subject*, the cases, when the adaptation facilities are *integrated with the platform*, where the subject operates, and the cases, when the adaptation facilities are completely *separated* and independent from the subject of adaptation.

Realization mechanisms define the tools and facilities, necessary to enable a given adaptation methodology, to implement the adaptation strategies, and to build the corresponding adaptation architecture. Realization mechanisms strongly depend on a given adaptation problem and on the approach used for that. Typical examples include, in particular, *reflection* wich refers to the ability of a program to reason about, and possibly alter, its own behavior; *automated composition* that provides a support for the automated service composition in order to accomplish composition (or adaptation) goals; *Service discovery / binding* that allows to find, select, and exploit a new service as a replacement of the incorrect one; *SLA negotiation* that allows to dynamically agree on the service quality, *Aspect weaving* techniques to inject the adaptation facilities into the SBA code, *design facilities* and tools supporting manual adaptation of SBA, etc.

Adaptation characteristics address some important challenges that adaptation process should satisfy, such as safety, security, optimality, cost, performance of the adaptation process.

4.3 Adaptation in Relevant Areas and Domains

4.3.1 Adaptation at Business Process Management

Adaptation of Business Processes

Adaptation of business processes may deal with permanent modification of the whole model or only a modification of a particular instance. In the former case, one can speak of evolution, as all the new instances of the process will follow the new model. This type of adaptation is usually achieved by re-designing/re-engineering the business processes.

Consequently, the term “adaptation” in the workflow and business process management systems refers to the run-time modification and/or extension of the running process instances in order to react to various problems and to accommodate different changes in their environment. These changes may be dynamics of organizational models, upcoming of better services, and new business rules and regulations.

The goal is to change the process while it is running, without having to remodel and redeploy the process, which is in general very time-consuming. Run-time modification of the business process instances normally assumes a strategy, which is predefined at design-time and which targets the modification of the structure of the process instance control flow or data flow.

The specification of the run-time adaptation activities may be implicit or explicit. The former case corresponds to the modification of the process model, and the changes are extracted as a difference between the old model and the new one. In the latter case the adaptation actions are predefined, e.g., using explicit variability techniques (possible alternatives are enumerated and associated with the corresponding applicability conditions) or as concrete actions (using special adaptation grammar that may define tasks to insert, remove, skip, change or reset value of variables, etc).

An important mechanism for the process instance adaptation refers to ensuring correctness of the adaptation activities in order to avoid so called instance migration bug. This problem refers to the fact that the changes performed on the partially executed process instance may lead to the situation, which violates certain predefined correctness requirements. In order to accomplish this, special mechanisms are introduced to validate the applicability of the adaptation actions with respect to a current instance.

Recovery actions for workflow systems

Focusing on the corrective adaptation, recovery actions have been widely used in workflow systems. While the general concepts are shared with all workflow management systems, several differences can be found in systems developed before the definition of SOA architecture that has deeply changed the concept of workflow, moving it from a coordination of actual tasks to a coordination of requests to partners to execute specific services.

Table 4.1: Adaptation of Business Processes

Dimension	Concepts	Description
Why	Usage	The adaptation in BPM may be motivated by various factors: a process instance may be modified in order to deal with a particular customer or a particular business context (adaptive adaptation); it may be performed in order to react to a certain problem or to a business-level failure (corrective adaptation).
Who	Actors	If the adaptation targets modification of the business process model as a whole, then it is performed "manually" (i.e., by re-design), requested and initiated internally by the business analysts. The modification of the running instances, instead, is performed autonomously, or under a human supervision, on the basis of the predefined actions. The adaptation of particular process instances may be also requested by the external stakeholders, e.g., by particular customers or partners.
What	Subject	The business process modifications may be triggered by the events that happened at the model level (corresponds to the SBA class; may happen, e.g., as a result of the business process re-design) or at the instance level.
	Aspect	Usually, the adaptation in BPM focuses on the modification of the business process models and on the modification of business transactions, transaction protocols, and mechanisms.
	Scope	The scope of adaptation depends on whether the whole business (process) model is changed or only a particular instance. In the former case we refer to permanent changes, and in the latter to the temporary ones as they affect the current process instance only.
How	Strategy	The modification of the business process model are usually performed at design-time; they require significant human involvement for identification the need for adaptation, for defining adaptation activities, and for the implementing these activities. It may have an effect on the other involved parties (deep changes), and may require changes at different layers (e.g., to recompose services, to configure transactional mechanisms, etc). The strategies for the instance adaptation may be realized at operation time either reactively (modification of already running instance) or proactively (adapting a new instance); they usually affect service composition realising a given process, and do not affect other partners (shallow changes). The specification may be defined implicitly or explicitly. In the latter case it may define both actions both for forward and backward adaptation.
	Decision Mechanisms	The adaptation decisions are normally made by the responsible analysts. In case of instance adaptation the decisions may be performed by the workflow management system on the basis of the specification predefined at design time: if there is a range of possible adaptation actions available, the selection is predefined by the corresponding policies, or may require user involvement.
	Implementation	While the realization of business process model is done manually through re-design, the implementation of instance adaptation relies on the specific workflow or process management systems. The latter are extended with the corresponding adaptation functionality, and perform the changes autonomously or with a human intervention. The mechanisms to ensure the modification correctness (to avoid process migration bugs) are also considered.

The work in [35, 52, 102] presents specific workflow models that widely support recovery actions; in [65] the authors focus on the analysis, prediction, and prevention of exceptions in order to reduce their occurrences. The model presented in [67] focuses on the handling of expected exceptions and the

Table 4.2: Recovery actions for workflow systems

Dimension	Concepts	Description
Why	Usage	Recovery actions might be required when the workflow fails and it is not possible to conclude the process. Focusing on the software infrastructure that drives the workflow, failures might happen for several reasons: systems in charge of performing some activities are not reachable, are failed, (corrective adaptation) or are not able to perform the activities ensuring a given quality level (perfective adaptation).
Who	Actors	Recovery actions are usually planned at design time by the designer driven by the requirements expressed by the requester. Whenever is possible to predict possible failures, the designer is in charge of identifying, for each possible failure scenario, the candidate recovery action or a set of alternative recovery actions. When workflow failures occur, recovery actions are performed at run-time. In case the planned recovery actions fails too, the executor, usually manually, is in charge of defining new recovery actions.
What	Subject	Recovery actions can affects both the system, the service on which the workflow is based, or even the structure of the process that drives the workflow.
	Aspect	Recovery actions affect and are affected by both functional and non-functional aspects. System failures as well as low quality allows the workflow engine to realize that a recovery action is required. Available services and the related quality will drive the engine to select the best recovery action strategy.
	Scope	Recovery actions can modify the behaviour only of the failed instance of the workflow in case of transient failure. On the contrary, in case of permanent failure, the recovery action might modify the structure of the workflow for all the instances that will be executed in the future.
How	Strategy	Possible recovery actions are: re-invocation of a failed interacting system (retry) with the same data, retry with different data, substitution, redefinition of the workflow. Recovery actions can affect the different layers. The enactment of a recovery action of the BPM layer can require the enactment of recovery actions at the bottom layers.
	Decision Mechanisms	Recovery actions are usually defined manually and execute automatically.
	Implementation	For simple recovery actions (as the retry) the execution can be totally automated. About the other cases, usually a human in the loop might be considered and specific tools can help him/her during the decision phase and the implementation and execution of the proper recovery actions.

integration of exception handling in the execution environment, while in [1] the authors propose the use of "worklets", a repertoire of self-contained subprocesses and associated selection and exception handling rules to support the modelling, analysis and enactment of business processes. The work in [54] presents the requirements of a Web Service Management framework which also includes the typical functionalities addressed in self-healing systems. The authors analyze and compare multiple alternative architectures for the implementation of Web Service Management systems proposing Web service substitution and complex service re-compositions as repair actions. In addition, an extensive amount of work on flexible recovery in the context of advanced transaction models has been done, e.g., in [61, 118]. They show how some of the concepts used in transaction management can be applied to workflow environments. The approach presented in [97] is mainly related to adaptivity in Information Systems, but some solutions based on the concept of proxy, under certain conditions, enable the reaction to faults in a way totally hidden to the process engine. Similar features, this time explicitly called recovery actions, are presented in [58]. In [68] the authors consider a set of recovery policies both on tasks and regions of

Table 4.3: Recovery and repair in service compositions

Dimension	Concepts	Description
Why	Usage	The adaptation is a reaction to possible anomalies occurred while executing the composition. While the adaptation of single services might be required to tailor their interfaces to the actual needs of the context in which services are used, in the whole composition, adaptation usually happens at run-time as attempts to fix problems triggered by an executing instance of the composition. The recovery is needed when the process is not be able to perform tasks ensuring the quality of the composition (perfective adaptation), and to customize a particular instance of a composition or to mediate interactions among services (adaptive adaptation).
Who	Actors	The designers define the process' business logic and adaptation and recovery separate. The recovery capabilities depend on the services the process interacts with (the requester). Recovering actions are performed at run time by the executor.
What	Subject	The recovering actions may affect the services the process interacts with, and/or the structure of the process that drives the composition.
	Aspect	Recovery actions are dealing with the dynamic binding. They may affect and are affected by both functional and non-functional aspects including QoS. The selection of the best recovery action is driven by the QoS and the available services at run time.
	Scope	Recovery actions can modify the behaviors at the protocol level or interfaces of services interacting in the composition process for the failed instance of the composition failure. The recovery actions might also modify the whole structure of the composition process for all instances that will be executed in the future.
How	Strategy	The framework can choose among a set of recovery actions: retry the failed interacting system, substitute the faulty service involved in the interaction by another one, compensate the failed interaction by canceling its effects. The adaptation can be local and tend to be pro-active and go forward.
	Decision Mechanisms	Some of recovery actions are defined manually and the others dynamically bound, and are executed automatically to fix a given anomaly.
	Implementation	Some recovery actions can be triggered automatically such as retry. For other cases, to carrying out adequate recovery actions, a human can usually use a specific tool (i.e., planning techniques)to help for making decision, implementing and executing the actions.

a workflow. They use an extended Petri Net approach to change the normal behavior when an expected but unusual situation or failure occurs. As in our approach the recovery policies are set at design time.

4.3.2 Adaptation in Service-Oriented Architectures

Recovery and repair in service compositions

When we think of *adaptation of web service compositions*, we must think of it as reaction to possible anomalies occurred while executing the composition. While the adaptation of single services might be required to tailor their interfaces to the actual needs of the context in which services are used, when we reason on the whole composition, adaptation usually happens at run-time as attempts to fix problems triggered by an executing instance of the composition. This means that in this case adaptation must be considered in conjunction with the monitoring infrastructure, where the latter provides the hooks for adaptation to happen. Usually, probes highlight anomalies and the adaptation infrastructure tries to solve, or bypass, them.

BPEL itself would allow designers to mix *defensive programming* techniques, fault, event, and compensation handlers with the actual business logic to embed adaptation into the process, but this solution would be mixed-up, inefficient, and inflexible (any change would require modifying the BPEL process and redeploying it). In contrast, we should perceive *separation of concerns* since it maintains the actual business logic and adaptation directives separate. Designers define the process' business logic, without considering adaptation and recovery: a single comprehensive specification would result in a complex, unmanageable, and intertwined definition.

The actual recovery capabilities heavily depend on the services the process interacts with. Stateless partner services simplify the problem. Things become more complex when the process interacts with stateful or conversational services. The former are services that have persistent side-effects when called (e.g., business data are stored on a persistent database). This means that they cannot be called an indefinite number of times (even twice), and that they must provide a special operation if we want to be able to undo their effects. The latter require that a special conversation protocol be respected. In these cases, we need a way to rollback the conversation itself. This is why the use of stateful and conversational services can lead to situations in which only partial recovery is possible. We should also distinguish between those solutions that only work on the running instance, the approaches that modify the composition itself, and those that comprise both [36].

Most of the process recovery approaches, or steering solutions, present in literature limit themselves to the more simple notion of dynamic binding. These approaches try to update the set of services with which they do business, and provide optimized experiences. Some limit themselves to substituting services that offer the same interface, while others provide mediation mechanisms. For example, we may think of optimizing a BPEL process' QoS by selecting the most appropriate partner services at run time [8, 7]. Designers define global and local QoS constraints, and these data are used to retrieve candidate services from external repositories. If a QoS requirement cannot be met, the framework can choose among a set of recovery actions: retry, substitute, and compensate.

We can also embed (re)binding rules in the composition language itself. For example, SCENE [42] offer a composition language that allows designers to declare policy (re)binding rules; policies can be either global or local. The framework also provides mediation capabilities through a special-purpose mediation scripting language. Similarly to the two previous proposals, VIEDAME [91], provides a dynamic adaptation and message mediation service for partner links. Using the data collected during the monitoring step, the system chooses the most appropriate service, while XSLT or regular expressions are used to transform messages accordingly.

All these approaches concentrate on local, or forward adaptation, while there are only a few proposals that also consider backward recovery, that is, the capability of rolling the execution back to a previous activity in the process, and restart it from there. Obviously, the actual options depend on both the process' topology and its partners services: the simple re-execution of the faulty path could not solve the problem. Self-Healing plug-in for a WS-BPEL engine (a.k.a. SH-BPEL, [90]), in particular, provides process-based recovery actions of both types. Recovery actions are specified by annotating WS-BPEL processes in a way that preserves the business logic, but enables specific recovery actions: the ability of modifying the value of process variables by means of external messages; the ability of redoing a single task or an entire scope; the possibility of specifying alternative paths to be followed, in the prosecution of the execution, after the reception of an enabling message; the possibility of going back in the process to a point defined as safe for redoing the same set of tasks or for performing an alternative path; substitution of dynamically selected services in case they do not respond within a timeout threshold.

Transactional BPEL processes provide special purpose fault handlers used to rollback the process if a transaction cannot be closed. For example AO4BPEL [38] proposes an AOP-based solution. In contrast, [8] provides an ad-hoc recovery primitive to let designer move back the execution of their processes the way they want. The gain in freedom and flexibility is clearly paid in terms of guarantees of the correctness of resulting executions.

Adaptation can also be carried out by exploiting planning techniques as a means to automatically

Table 4.4: Adaptation for QoS-based optimization

Dimension	Concepts	Description
Why	Usage	Adaptation for QoS-based optimization aims at executing the a service-based process with the best quality (perfective adatation). To this aim, sometime, the composing services, as well as the process structure might change (corrective adaptation)
Who	Actors	QoS level that must be ensured at run-time is defined at design time after an agreement between the service provider and the requestor. The agreement is usually formalized in terms of SLA. The executor is in charge of monitoring and react to low-quality situation by means of adaptations.
What	Subject	In case of temporary low-quality provisioning, a specific instance is adapted. Otherwise, in case of more severe situation the SBA class could be adapted.
	Aspect	The quality aspect drives the adaptation in terms of system performance or even in terms of changed context aspects.
	Scope	QoS-based adaptation can be both temporary or permanent due to the nature and the source of the low-quality issue.
How	Strategy	Usually QoS-based adaptation is planned at design time with respect to the agreement defined in the SLA where responsibilities and penalties are also defined. The more the automation of the SLA increases, the more the adaption shifts from the design-time to the run-time. Possible recovery actions require the re-invocation of the service that make the quality lower than expected. In case of permanent problems, the substitution of such a service can be required.
	Decision Mechanisms	QoS-based adaptation is usually driven by the SLA. In this document the responsibilities about the monitoring of QoS information as well as the responsibilities about the adaptation enactment are defined.
	Implementation	Depending on the type of adaptation strategy required, its execution can be totally automated or supervised by a human.

compute the aggregation of services able to fix a given anomaly. For example, if a given service does not answer anymore, and we have no “equivalent” services readily available, we can exploit planning techniques, based on semantically-enriched services descriptions, to compute the on-the-fly composition that substitutes the faulty service. The same approach can easily be extended to cover the run-time substitution of process fragments. For example, McIlraith [88] extended and adapted Golog [82] for automatic Web service composition based on services encoded in OWL-S, while Pistore et al. [100], exploit Planning as Model Checking to automatically compose Web services.

Adaptation for QoS-based optimization

In case of Web service based processes, the quality of the overall process strictly depends on the quality provided by Web services tied to the task. Zeng et al. in [120] proposes an approach for deriving the process quality starting from the Web service properties considering some of the most relevant quality dimensions.

In this scenario, as defined in the perfective adaptation, it might happen that even if the process runs properly an adaptation is required because of insufficient quality. As a consequence, the SBA should react in order to improve the quality of the service process. The goal is to select the best set of services available at run-time, taking into consideration process constraints, but also end-user preferences and the execution context.

Web service selection results in an optimization problem that has been studied both in the research areas of service oriented computing for business processes and of grid environments. The literature has provided two generations of solutions. First generation solutions implemented local approaches [89, 120, 6], which select Web services one at the time by associating the running abstract activity to the best candidate service which supports its execution. Local approaches can guarantee only local QoS constraints, i.e., candidate Web services are selected according to a desired characteristic, e.g., the price of a single Web service invocation is lower than a given threshold.

Second generation solutions proposed global approaches [120, 32, 41, 76]. The set of services that satisfy the process constraints and user preferences for the whole application are identified before executing the process. In this way, QoS constraints can predicate at a global level, i.e., constraints posing restrictions over the whole composed service execution can be introduced. In order to guarantee the fulfillment of global QoS constraints, second generation optimization techniques consider the worst case execution scenario for the composed service. For cyclic processes, loops are unfolded, i.e., unrolled according to their maximum number of iterations [120, 32]. These approaches could be very conservative and constitutes the main limitation of second generation techniques.

Furthermore, global approaches introduce an increased complexity with respect to local solutions. The main issue for the fulfillment of global constraints is Web service performance variability. Indeed, the QoS of a Web Service may evolve relatively frequently, either because of internal changes or because of workload fluctuations [120, 37, 121]. If a business process has a long duration, the set of services identified by the optimization may change their QoS properties during the process execution or some services can become unavailable or others may emerge. In order to guarantee global constraints Web service selection and execution are interleaved: Optimization is performed when the business process is instantiated and its execution is started, and is iterated during the process execution performing re-optimization at run-time. To reduce optimization/re-optimization complexity, a number of solution have been proposed that guarantee global constraints only for the critical path [120] (i.e., the path which corresponds to the highest execution time), or reduce loops to a single task [32], satisfying global constraints only statistically, by applying the reduction formula proposed in [34]. Another drawback of second generation solutions is that, if the end-user introduces severe QoS constraints for the composed service execution, i.e., limited resources which set the problem close to infeasibility conditions (e.g., limited budget or stringent execution time limit), no solutions can be identified and the composed service execution fails [32]. While first and second generation approaches have been applied, e.g., [27, 120], the need for further research toward more advanced optimization techniques, in particular for cyclic processes [56, 6] is advocated. In addition, none of the previous approaches considers in the optimization the case of processes composed by stateful Web services, where more than one task must be performed by the same Web service.

In [7] an alternative modeling approach to the service selection problem is introduced, where: i) loops peeling is adopted in the optimization, which significantly improves the solutions based on loops unfolding, ii) negotiation is exploited if a feasible solution cannot be identified, to bargain QoS parameters with service providers offering services, reducing process invocation failures, and iii) a new class of global constraints, which allows the execution of stateful Web service components, is introduced.

Service evolution

Service evolution refers to the continuous process of development of a service through a series of consistent and unambiguous changes. Service evolution is motivated by the need to cope with multiple stakeholders, fluid requirements, and external pressures that affect entire organization. A key issue for the service evolution is the ability to support service diversification: the services should co-exist in multiple active versions.

Such changes are not performed autonomously; they require intensive human involvement to identify, design, and management of service evolution. To control service development one needs to know why a change was made, what are its implications and whether the change is complete. Such a change is requested, designed, initiated, and performed by the service provider. However, as the evolutionary changes are permanent and significant, the effect of the strategy spans the boundary of a service and

Table 4.5: Service evolution

Dimension	Concepts	Description
Why	Usage	The changes performed with service evolution originate from the need to introduce new functionality (extending adaptation), the modification of existing functionality to improve performance (perfective adaptation), the inclusion of new regulatory constraints that require that the behavior services be altered or to accommodate to the requirements of new customers (adaptive adaptation).
Who	Actors	Changes are requested, designed, initiated, and performed by the service provider (i.e., designer). In order to manage changes as a whole, the Web service consumers (i.e., requester) have to be taken into consideration as well, otherwise changes that are introduced at the service provider side can create severe disruption.
What	Subject	Service evolution deals with the permanent modification resulting in the new versions of the underlying service (i.e., SBA class adaptation).
	Aspect	Driven by a variety of requirements and factors, adaptations may affect various aspects of the service implementation at different functional layers. Moreover, evolution might be required to tailor their interfaces to the actual needs of the context in which services are used.
	Scope	As the new version of a service is created, the scope of adaptation is permanent for the new version: all future executions of this service version will follow the new model.
How	Strategy	The adaptation requirements are addressed by re-designing of a service leading to a new version. This requires intensive human involvement at deciding, designing, and implementing new service versions. The modifications are significant; they may affect different functional SBA layers; the scope of effect spans across the boundary of the service provider (deep changes).
	Decision Mechanisms	The decisions on what to change and on how the changes are realized in the new version are defined by the stakeholders and designers. Theses decisions take into account the Web service consumers as well; otherwise changes that are introduced at the service producer side can create severe disruption. Service evolution attempts to a priori validate and constrain service changes and ensuing versions so that they are consistent and well-behaved.
	Implementation	The realization of the service evolution activities is achieved by the mechanisms of Service Evolution Management. The implementation of those mechanisms is tailored to the understanding of all the points of change impact, controlling service changes, tracking and auditing all service versions, and providing status accounting.

propagates to the service customers (deep changes). In order to manage changes as a whole, the Web service consumers have to be taken into consideration as well, otherwise changes that are introduced at the service producer side can create severe disruption.

The evolution of the service is achieved through the creation and decommission of its different versions during its lifetime. These versions have to be aligned with each other in a way that would allow a service designer to track the various modifications and their effects on the service. A crucial requirement for service evolution is, therefore, a robust versioning strategy to support development of multiple versions to support their upgrades, while continuously supporting previous versions (e.g., to be able to deal with message exchanges between a provider and a client despite service changes).

In contrast to other approaches to service adaptation that aim to *a posteriori* modify itself in order to address potential functional or non-functional mismatches, service evolution attempts to *a priori* validate and constrain service changes and ensuing versions so that they are consistent and well-behaved. This is achieved by the mechanisms of *Service Evolution Management*, i.e., is the process of managing the

Table 4.6: User and HCI Aspects in Adaptation

Dimension	Concepts	Description
Why	Usage	The adaptation is required to adjust the system with respect to the context changes (adaptive adaptation) and the user interactions of the system (perfective adaptation).
Who	Actors	The adaptation is usually performed by the service provider (i.e., designer) upon the user request during the interactions or while the context changes.
What	Subject	HCI or context based adaptation can be achieved at different level of SBA from higher granularity to instance level. The context changes might also affects the monitoring rules and infrastructure.
	Aspect	The adaptation is driven by HCI and context factors and may affect various aspects of the service implementation at different functional layers including monitoring mechanisms. The non functional aspects (i.e., privacy disclosure of a user) are also taking into account.
	Scope	The adaptation is designed to be temporary performed, because the change is made with respect to a user (one application) or a particular context.
How	Strategy	In context based adaptation, the adaptation specification is actions-based, defined at design time as a set of policies and executed statically or dynamically. HCI and context based adaptation mechanisms can be implemented at different level of system architecture.
	Decision Mechanisms	Human is in the loop for the HCI adaptation. In contrary, the context-based adaptation is driven dynamically and is decided using policies defined at design time.
	Implementation	The adaptation mechanisms are implemented at different level of the system architecture from the application level to the underlying platform of the application.

effective implementation of service evolution and co-existence of multiple active service versions, in a way that ensures that permanent changes in structure, behavior, relationships, and associated service versions are achieved in a sound and consistent manner. In summary, Service Evolution Management (SEM) exhibits the following characteristics:

- mechanisms to identify all kinds of permissible changes to services and to classify them;
- propagation analysis mechanisms that record the status of services, analyze change requests and gather information about the clients of a service version to ensure compliance with respect to service updates and version contracts;
- version control mechanisms that ensure consistent behavior of services, by controlling the release of a service and changes applied to it throughout its life-cycle;
- validation and resolution mechanisms that validate the completeness of a change and maintain consistency by ensuring that a service is a well-behaved collection of service changes and versions;
- instance migration mechanisms for associating instances of running services with new versions.

4.3.3 User and HCI Aspects in Adaptation

The concept of context encompasses various aspects such as computing environment (e.g. available processors, network connectivity and capacity, input output devices), user environment (e.g. location, collection of nearby people) and physical environment (e.g. lighting, noise level) [48]. Context based system adaptation is the process where a system is adjusted with respect to the change in the context and the tasks and actions of the user of the system [31, 103, 119]. The later case, where the system is adapted with respect to the user interaction, is known as Human Computer Interaction (*HCI*) based system adaptation. Clearly, these problems instantiate adaptive form of adaptation.

In the settings of context-based and in particular, *HCI*-based adaptation, it is possible that the monitoring underlying process will may have to adjust itself with respect to *(i)* context changes, which is termed as context based adaptation of the monitoring, and *(ii)* human computer interaction, which is termed as *HCI* based adaptation of the monitoring. The former may include, but is not limited to, dynamic selection of monitoring rules based on context information [47, 111] and change of monitoring infrastructure based on context information [111]. For example, consider a service that can be deployed either on desk top computer, or personal digital assistants (PDA), or mobile phone and there is a set of rules to be monitored for each of these devices. The monitor should select a set of rules to monitor based on the device the service has been deployed to. Consequently, the subject of adaptation refers not only the application instances, but also to the monitoring mechanisms.

In context-based adaptation, adaptation specification is action-based: it has a form of a set of policies that define what should be done to adapt the behaviour when a change is detected are specified at design time using formal [103, 104] or semi formal [111] languages. The adaptation strategies are applied at run-time, when context information is matched against the specified monitored properties in order to trigger specific actions prescribed in the adaptation policies. It should be noted that the actions specified in policies can be *(i)* static in nature, i.e. given a specific change observed in the context a specific type of actions should be taken to adapt the system to the context [70, 33, 57]. For example, in case of a PDA, if the battery power goes below a certain level, then switch to a monochrome display from a colour display, or *(ii)* dynamic in nature, i.e. given a change in the context run time solution should be computed that is the most appropriate for the current context [23, 110]. For example, consider a video streaming service, where the compression ratio of the video files can be determined at the runtime based on the available bandwidth.

HCI or context based adaptation can be achieved by implementing adaptation mechanism at different level of the system architecture [110, 95]. For example, adaptation mechanism can be implemented at application level, as it may be feasible for the application to decide how best to exploit the available resources [33, 110, 95]. Again adaptation mechanism can be implemented at the underlying platform of the application where entities from the platform (e.g. file access system or middleware component) are directly involved in the adaptation mechanism [110, 95]. The third approach lies in between these two approaches where the implementation of the adaptation mechanism is allocated to different levels of the system architecture [110].

4.3.4 Adaptation in Grid computing

Grid computing is defined as coordinated resource sharing. This definition already presents us the two major participants of the Grid adaptation techniques. First of all Grid mainly offers two kind of resources: storage and computing resources. The second major part is the coordination, which might adapt user requests to the currently available resources. Therefore in the current grid systems the two main entities offering adaptation is the storage broker and the (computing) resource broker. Both of these brokers are offering perfective and corrective adaptation techniques.

The example of perfective adaptation can be understood more easily with the help of storage brokers. These systems examine data requests and according to their context they translate these requests to use the data which are closer to the currently executed application which plans to process it. The primary actor who initiates the adaptation is the application itself, who requests the adaptation of the data request from the storage broker. It is important that the application usually asks for the adaptation every time a different data request is made, therefore this adaptation is temporary. Temporary adaptation can be achieved because context dependency in storage brokers, which means for example that they adapt the data requests according the requester's execution location, and they can take into account the user's

Table 4.7: Adaptation in Grid computing

Dimension	Concepts	Description
Why	Usage	Adaptation in grid computing is offered for perfective and corrective purposes. The *perfective* behavior deals with the access optimization of the various resources shared among the grid participants. The optimization can range from similar resource assignment for correlating computing tasks to choosing an execution location in the proximity of the input data storage. The *corrective adaptation* in grid has a single purpose, that the user should not notice the non guaranteed behavior of the underlying services of the grid system.
Who	Actors	The executor of the adaptation strategy is the task of the different resource brokers, which are including several adaptation strategies in their design. The adaptation is usually triggered on faults (corrective systems) or on user request (perfective systems).
What	Subject	Adaptation in grid focuses on the SBA instance. The actual execution of the application is adapted, however the erroneous situations can rise again. The infrastructure is not reorganized to better fit the SBA class, this is a manual task of adding new resources usually. However with the rise of the virtualization the infrastructure is now capable of adapting to the SBA class more closely, without affecting other SBAs offered by the same grid.
	Aspect	The usefulness is the key moving factor in grid. Thus the adaptation is applied usually when the users would gain processing time on its requests based on the adapted task descriptions they gave.
	Scope	Adaptation is temporary in grids, they don't change because of a single application, the application however is usually adapted for the currently available grid infrastructure - e.g. it is using only the available processors on a given site or it is using grid level messaging instead of plain MPI.
How	Strategy	Re-configuration and re-execution is the two usually chosen strategy in grid. Re-configuration usually occurs before execution, which means the execution request is adapted to the current grid context instead of using an improperly formulated one. Re-execution usually appears together with re-configuration, and it is applied when the resource broker realizes that there is a possible SLA violation. In this case the broker reconfigures the execution request and if it is still feasible it re-executes it accordingly.
	Decision Mechanisms	are automated with the help of the grid infrastructure, the user can influence the chosen strategies and the possible outcomes by specifying extra details on its request to the resource brokers.
	Implementation	Grid adaptation techniques are usually integrated with the grid middleware, and barely delivered with a specific application.

identity (e.g. the VO membership) who executes the application. VO membership based adaptation however leads us to preventive adaptation, since it is unlikely that a user can access data from storage resources in different VOs, therefore the storage broker makes sure that errors happen less frequently.

Corrective adaptation best represented by resource brokers. In Grid, resource brokers are the primary access points to the resources. They act as a super resource, which might tackle all computing tasks of a given virtual organization. When a user sends a task to the broker it selects a computing resource which best fits the request. However the non guaranteed services offered by the computing resources need that the broker adapts the request in case the previously selected resource rejects or fails to execute the user's task. The adaptation is usually achieved by substituting the previously selected resource with the second best fit resource. This is a self adaptation mechanism because there is no need for human interaction after the user sent the task to the resource broker. In some extreme cases no resources can complete the user's request. This case arises when the resource broker exhausts the list of the matching resources, which could fulfill the user's request, because all the resources from the list fail to process the request properly. As a result the resource broker might renegotiate the request with the user by describing the previous trials it made to finish the requested task, and suggesting adjustments to the following request.

In case QoS is the driving factor for adaptation in Grid computing, then the adaptation system might change the Grid information system's entry about a resource according to the rate of resource failures or congestion. The Grid information system (an implementation of grid level monitor) is the primary source for high level components where the Grid level decisions are made. Changes in the information system's offerings mean the whole grid infrastructure acts according to this data, therefore a simple change uniformly adapts all systems dependent on the information service. Another aspect of the QoS awareness that by using virtualized resources in Grid it is possible to change certain properties of those virtual machines which provide the computing resources. As an example adapting virtualized resources might use the ballooning capability of the virtual machine monitors (they can change the system memory in run-time).

Finally adaptation strategies can be specifically designed for parallel constructs used in Grid technologies. Like in case of process farms the number of worker processes can be adapted depending on the number of nodes present on the computing resource selected as the execution location of the process farm.

4.3.5 Adaptation in Component-Based Systems

Adaptation in software systems targets a self adaptation of the software. Most of time, adaptations are designed and tested on component-based systems, because of their high modularity.

Contexts of adaptation

Correction and prevention of system failures is probably the principal context of adaptation. A corrective adaptation aims to replace a faulting component by another one having the same behaviour. As the system can be offered different components to retrieve the same information, if an action query on a component leads to a component failure, the system will replace the component by another and try again. Sometimes, system failures are due to modifications of the runtime environment (loss of a peripheral device for example) and the system may have to realize preventive adaptations to feet the new execution conditions, avoid failures and ensure correct behavior. Some research in this domain is towards preventing component failures by launching preventive adaptations in order to prepare the system to a possible context modification.

Enhancement of the QoS makes the second context of adaptation. For example, a component-based HTTP-proxy server have to perform perfective adaptations on its behavior to find the right balance between the memory used and the latency. These adaptations can be realized by changing the request handler component implementation.

Platform extension concerns the ability of the system to take into account new components by adapting the in-place components behavior or the platform capabilities.

Adaptation strategies

The integration of real selection strategy engines in middleware software is not yet commonplace. Some works investigate concepts and techniques developed in the artificial intelligence field such as expert systems, constraints satisfaction algorithms [66], neural networks or fuzzy approaches [39].

Table 4.8: Adaptation in Component-Based Systems

Dimension	Concepts	Description
Why	Usage	Component-based system adaptation can be corrective (replacement of a non satisfying component), adaptive (to fit new needs), extending in the way you an add or remove functionalities, perfective (replacing a component or a composition by a one more performent) or preventive (replacing a component or a composition that seems to lead to a system failure).
Who	Actors	The main effort is made on self-adaptive systems. As a consequence, adaptations are (most of time) initiated and executed by the application itself, regarding monitored properties of the system or the context. The request may also come from a human or from a non-satisfied close in a contract (behavior specification), according to adaptation policies and mechanisms described by the application designer.
What	Subject	The adaptation can target a component (in a fine grain approach) or a composition in a higher granularity, and can also change adaptations policies and requesters.
	Aspect	Driven by a variety of requirements and factors, adaptations may affect various aspects of the components implementation at different functional layers.
	Scope	When adaptation concerns a component, or the composition of components, this adaptation is temporary and designed to be dynamically triggered by either adaptation policies or QoS reasons. Adaptations are permanent in the context of a version change for example. The last point is the adaptation of the adaptation policies, that tend to be dynamic by using some learning mechanisms. In general terms, the adaptation is temporary if it concerns the performances and is permanent if it targets an evolution of the system.
How	Strategy	In general case, adaptations are local on composition level; mostly reactive, they tend to be pro-active and go forward.
	Decision Mechanisms	Most adaptations are automatic (no human-in-the-loop) and dynamic, because adaptations are decided using policies.
	Implementation	Component-based systems are designed to realize self-adaptations (autonomous)

Adaptation specifications are based on precise descriptions of the software architecture. These descriptions are mainly made using architecture description languages (ADL) or model driven engineering (MDE). Some architectures description languages has been classified in Barais PhD thesis [14] according to their dynamic reconfiguration support. In [81], Lau gives an overview of components metamodels based on the same point of view. Being graphical or textual, these solutions allow the system to be described as a software component's composition. As a consequence, an adaptation of the architecture is as simple as a modification of component link's or as a component add or remove action. The adaptation specification can be explicited by getting the differences between the before-modification system description and the after one, and then passed to the adaptation implementation to be realized. The implicit way to specify the adaptation is to give the modified system description to the adaptation implementation which is in charge of expliciting the adaptation description before its realization.

Adaptation of system properties is probably the simplest adaptation mechanism in software systems. According to some environment considerations, the values of system parameters are computed to render the most efficient behavior. Most of execution platforms offer this ability and this may be considered as perfective or preventive adaptation mechanism.

Component adaptation aims to replace a component or a part of its internal code by another. This adaptation mechanism is mainly used in a corrective context. For sure, implementation of such an adaptation mechanism is based on platforms supporting dynamic modification of the application at run-time. The OSGi kernel [2] is a standard container-provider to built service-oriented software. It implements a cooperative model where applications can dynamically discover and use services provided by other applications running inside the same kernel. In this platform, applications can be installed, started, stopped, updated and uninstalled without a system restart and at any moment. According to these specifications, an application is then divided on several bundles (OSGi components). Each bundle is designed to reach the highest level of independence, giving the software enough modularity to allow partial system adaptations such as services updates, additions or removes. Fractal (cited below) can be presented as another platform allowing this kind of adaptation.

Structure adaptation of component-based application modifies the relations between components and can add, remove or replace components of the system. Used in extending and adaptive adaptations, this adaptation implementation relies on tools that support dynamic reconfiguration of deployed applications. These adaptation facilities are to be integrated within the platform (the middleware) the subject operates on. DynamicTAO [80], DART [101] and OpenCom [40, 44] offer introspection features and introactions that allow an application to change its configuration and consequently the connections between the various components. This reconfiguration has been made possible thanks to the accessible architectural entities and the numerous operations offered in order to change the topology of the application (in terms of components and connectors). Fractal [30, 29] is another example of run-time platform supporting adaptive systems. Indeed, one of the peculiarities of Fractal components is to provide control interfaces (Controller in the Fractal terminology) allowing configuration of some extra-functional aspects of the components such as the life cycle, the topology of internal composites, etc. Pukas [74] propose an implementation of the J2EE platform, introducing joint actions procedures (add, delete, connection, etc.) but is not as reflexive as Fractal.

Adaptation selection and mechanisms

The adaptation selection mechanism presented in Garlan et al. [60], is based on a principle of violation / reaction associated with architectural invariants. Each invariant is associated with an adaptation strategy described in an imperative form on the architecture model. When the system detects a breach of an invariant it triggers the adaptation implementation of the involved strategy.
QualProbes [83] and MADAM [55], in the other hand, use a dynamic selection. QualProbes incorporates an engine of fuzzy logic to dynamically set the quality parameters of the deployed applications. Each application describes the rules that control its quality parameters. The middleware is responsible for implementing these rules and for computing values of relevant parameters in accordance with available resources. The idea developed into QualProbes is quite relevant but does not take into account the dynamic architectural adaptation term (adding and deleting components or connectors). MADAM incorporates a adaptation selection strategy based on a calculation of utility function that allows him to choose the best possible configuration depending on the environment.
Batista et al. [20] provide a run-time environment called Plastik that triggers system adaptation when measured properties of the platform evolves.

Chapter 5

Conclusion

This deliverable provided a comprehensive overview of the knowledge and concepts in the field of adaptation and monitoring for service-based applications, taking into account and considering the knowledge from the S-Cube disciplines business process management, service-oriented computing, grind computing and software engineering. This knowledge has been analyzed, aligned and synthesized at various levels of detail as follows:

- Firstly, the adaptation and monitoring framework, introduced in the workpackage vision, has provided a high level view of the key logical elements needed for adaptation and monitoring of SBAs. In addition, it has depitced the basic dependencies between those elements.
- Secondly, more detailed conceptual models for monitoring and adaptation have been devised, which include a refinement of those key logical elements by introducing important (general) concepts and relationships between those concepts.
- Thirdly, for many of the concepts in the conceptual models, specialized concepts exist; e.g., different kinds of adaptation exist, including corrective, perfective or preventive adaptation. Thus, taxonomies for monitoring and adaptation have been created which classify and list these concepts.
- Finally, the concepts from above have been related to individual domains and research challenges to show their use within those domains. Based on those domain instantiations, the differing or similar usage of terms could be identified. In fact, the key definitions for concepts (both domain-specific and generic) have been provided to the S-Cube Knowledge Model.

The explicit adaptation and monitoring knowledge, as documented in this deliverable, will serve as input for the future work of this workpackage as well as for the other research workpackages. Specifically, this knowledge will be exploited in the follow-up tasks of this workpackage, which will further investigate and refine adaptation principles, techniques, and methodologies (T-JRA-1.2.2 'Integrated Adaptation Principles, Techniques and Methodologies') and which will focus, in particular, on the role of context and human involvement in the SBA monitoring and adaptation (T-JRA-1.2.3 'Comprehensive, Context-Aware Monitoring').

Bibliography

[1] M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Facilitating flexibility and dynamic exception handling in workflows through worklets. In *Short Paper Proceedings at (CAiSE)*, volume 161 of *CEUR Workshop Proc.*, Porto, Portugal, 2005.

[2] The OSGi Alliance. Osgi service platform core specification, release 4, April 2007.

[3] Christos Anagnostopoulos, Athanasios Tsounis, and Stathes Hadjiefthymiades. Context awareness in mobile computing environments: A survey. In *Mobile eConference*, 2004.

[4] Sergio Andreozzi, Stephen Burke, Flavia Donno, Laurence Field, Steve Fisher, Jens Jensen, Balazs Konya, Maarten Litmaath, Marco Mambelli, Jennifer M. Schopf, Matt Viljoen, Antony Wilson, and Riccardo Zappi. GLUE Schema Specication, version 1.3, January 2007.

[5] A. Andrieux, C. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Version 1.1, June 2004.

[6] D. Ardagna and B. Pernici. Global and Local QoS Guarantee in Web Service Selection. In *BPM 2005 Workshops Proc.*, pages 32–46, 2005. Nancy.

[7] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *Software Engineering, IEEE Transactions on*, 33(6):369–384, June 2007.

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

[9] Liliana Ardissono, Roberto Furnari, Anna Goy, Giovanna Petrone, and Marino Segnan. Fault Tolerant Web Service Orchestration by Means of Diagnosis. In *Software Architecture, Third European Workshop, EWSA 2006*, pages 2–16, 2006.

[10] T. Assaf and J.B. Dugan. Diagnosis based on reliability analysis using monitors and sensors. *Reliability Engineering & System Safety*, 93:509–521, 2008.

[11] B. Azvine, Z. Cui, and D. D. Nauck. Towards real-time business intelligence. *BT Technology Journal*, 23(3), 2005. http://dx.doi.org/10.1007/s10550-005-0043-0.

[12] B. P. Bailey, P. D. Adamczyk, T. Y. Chang, and N. A. Chilson. A framework for specifying and monitoring user tasks. In *Journal of Computers in Human Behavior, special issue on attention aware systems*, 2006.

[13] Matthias Baldauf and Schahram Dustdar anf Florian Rosenberg. A survey on context-aware systems. In *International Journal of Ad Hoc and Ubiquitous Computing*, pages 263–277, 2007.

[14] Olivier Barais. *Construire et Matriser l'evolution d'une architecture logicielle base de composants*. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Lille, France, nov 2005.

[15] Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *IEEE International Conference on Web Services (ICWS 2006)*, pages 63–71, 2006.

[16] Jakob E. Bardram. The java context awareness framework (jcaf) - a service infrastructure and programming framework for context-aware applications. In *In Hans Gellersen, Roy Want, and Albrecht Schmidt, editors, Proceedings of the 3rd International Conference on Pervasive Computing, Lecture Notes in Computer Science, Munich, Germany*. Springer Verlag, 2005.

[17] L. Baresi, S. Guinea, and L. Pasquale. Self-healing BPEL processes with Dynamo and the JBoss rule engine. In *ESSPE '07: International workshop on Engineering of software services for pervasive environments*, pages 11–20, 2007.

[18] Luciano Baresi, Domenico Bianculli, Carlo Ghezzi, Sam Guinea, and Paola Spoletini. A Timed Extension of WSCoL. In *2007 IEEE International Conference on Web Services (ICWS 2007)*, pages 663–670, 2007.

[19] Luciano Baresi and Sam Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *Service-Oriented Computing - ICSOC 2005, Third International Conference*, pages 269–282, 2005.

[20] Thais Batista, Ackbar Joolia, and Geoff Coulson. Managing dynamic reconfiguration in component-based systems. *Software Architecture*, pages 1–17, 2005.

[21] Salima Benbernou, Luca Cavallaro, Mohand Said Hacid, Raman Kazhamiakin, Gabor Kecskemeti, Jean-Louis Pazat, Fabrizio Silvestri, Maike Uhlig, and Branimir Wetzstein. *State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs*. S-Cube project deliverable, July 2008. S-Cube project deliverable: PO-JRA-1.2.1. http://www.s-cube-network.eu/achievements-results/s-cube-deliverables.

[22] Salima Benbernou, Hassina Meziane, and Mohand-Said Hacid. Run-Time Monitoring for Privacy-Agreement Compliance. In *Service-Oriented Computing - ICSOC 2007, Fifth International Conference*, pages 353–364, 2007.

[23] Claudio Bettini, Dario Maggiorini, and Daniele Riboni. Distributed context monitoring for continuous mobile services. In *John Krogstie, Karlheinz Kautz, David Allen (Eds.): Mobile Information Systems II: IFIP Working Conference on Mobile Information Systems (MOBIS)*, pages 123–137. Springer, 2005.

[24] Claudio Bettini, Dario Maggiorini, and Daniele Riboni. Distributed context monitoring for the adaptation of continuous services. In *World Wide Web Journal (WWWJ), Special issue on Multichannel Adaptive Information Systems on the World Wide Web*. Springer, 2007.

[25] Domenico Bianculli and Carlo Ghezzi. Monitoring Conversational Web Services. In *IWSOSWE'07*, 2007.

[26] D. Box, F. Curbera, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, M. Nottingham, C. von Riegen, and J. Shewchuk. Web Services Policy Framework (WS-Policy), May 2003.

[27] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt. QoS support for time-critical gridworkflow applications. In *e-Science 2005 Proc.*, pages 108–115, 2005.

[28] P. Bratskas, N. Paspallis, and G. A. Papadopoulos. An evaluation of the state of the art in context-aware architectures. In *Sixteenth International Conference on Information Systems Development (ISD 2007)*. Springer Verlag, 2007.

[29] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean B. Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36(11-12):1257–1284, 2006.

[30] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quema, and Jean-Bernard Stefani. An open component model and its support in java. *Component-Based Software Engineering*, pages 7–22, 2004.

[31] Galle Calvary, Jolle Coutaz, and David Thevenin. Embedding plasticity in the development process of interactive systems. In *6th ERCIM Workshop: User Interface for All*, 2000.

[32] G. Canfora, M.di Penta, R. Esposito, and M. L. Villani. QoS-Aware Replanning of Composite Web Services. In *ICWS 2005 Proc.*, 2005. Orlando.

[33] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. In *In IEEE Transactions of Software Engineering Journal (TSE). November 2003*, 2003.

[34] J. Cardoso. Quality of Service and Semantic Composition of Workflows. Ph. D. Thesis, Univ. of Georgia, 2002.

[35] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data Knowl. Eng.*, 24(3):211–238, 1998.

[36] Fabio Casati, Ski Ilnicki, Li jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. eflow: A platform for developing and managing composite e-services. In *AIWoRC*, pages 341–348. IEEE Computer Society, 2000.

[37] S. Chandrasekaran, J. A. Miller, G. Silver, I. B. Arpinar, and A. P. Sheth. Performance Analysis and Simulation of Composite Web Services. *Electronic Market: The Intl. Journal of Electronic Commerce and Business Media*, 13(2):120–132, 2003.

[38] Anis Charfi and Mira Mezini. Ao4bpel: An aspect-oriented extension to bpel. In *World Wide Web*, pages 309–344, 2007.

[39] Franck Chauvel. *Méthodes et outils pour la conception de systèmes logiciels auto-adaptatifs*. PhD thesis, l'Université Européenne de Bretagne, Septembre 2008.

[40] Michael Clarke, Gordon Blair, Geoff Coulson, and Nikos Parlavantzas. An efficient component model for the construction of adaptive middleware. *Middleware 2001*, pages 160–178, 2001.

[41] D. B. Claro, P. Albers, and J. K. Hao. Selecting Web Services for Optimal Composition. In *ICWS 2005 Workshop Proc.*, 2005. Orlando.

[42] Massimiliano Colombo, Elisabetta Di Nitto, and Marco Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In Asit Dan and Winfried Lamersdorf, editors, *ICSOC*, volume 4294 of *Lecture Notes in Computer Science*, pages 191–202. Springer, 2006.

[43] Denis Conan, Romain Rouvoy, and Lionel Seinturier. Scalable processing of context information with cosmos. In Jadwiga Indulska and Kerry Raymond, editors, *DAIS*, volume 4531 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 2007.

[44] Geoff Coulson, Gordon S. Blair, Michael Clarke, and Nikos Parlavantzas. The design of a configurable and reconfigurable middleware platform. *Distributed Computing*, 15(2):109–126, 2002.

[45] Yi Cui and Klara Nahrstedt. Qos-aware dependency management for component-based systems. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, page 127, Washington, DC, USA, 2001. IEEE Computer Society.

[46] Pierre-Charles David and Thomas Ledoux. Wildcat: a generic framework for context-aware applications. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–7, New York, NY, USA, 2005. ACM.

[47] Anne-Marie Dery-Pinna, Jérémy Fierstone, and Emmanuel Picard. Component model and programming: A first step to manage human computer interaction adaptation. In *Lecture Notes in Computer Science*, pages 456–460. Springer-Verlag, 1999.

[48] Anind K Dey and Gregory D. Abowd. Towards better understanding of context and context-awareness. In *In Proceedings of the CHI 2000 Workshop on The What, Who, Where, When and How of Context-Awareness*, pages 1–6, 2000.

[49] Jian-Wan Ding, Liping Chen, and Fanli Zhou. A component-based debugging approach for detecting structural inconsistencies in declarative equation based models. *J. Comput. Sci. Technol.*, 21(3):450–458, 2006.

[50] Distributed Management Task Force, Inc. Common Information Model (CIM) Standards. `http://www.dmtf.org/standards/cim/`.

[51] H. Dresner. Business activity monitoring: Bam architecture. In *Gartner Symposium ITXPO (Cannes, France)*, 2003.

[52] J. Eder and W. Liebhart. Workflow recovery. In *Proc. of IFCIS Int. Conf. on Cooperative Information Systems (CoopIS)*, pages 124 – 134, Brussels, Belgium, 1996. IEEE.

[53] Jacob Eisenstein, Jean V, and Angel Puerta. Adapting to mobile contexts with user-interface modeling. In *Proc. of 3 rd IEEE Workshop on Mobile Computing Systems and Applications WMCSA*, 2000.

[54] E. Esfandiari and V. Tosic. Towards a web service composition management framework. In *Proc. of Int. Conf. on Web Services*, Orlando FL, USA, 2005.

[55] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven. Using architecture models for runtime adaptability. *IEEE Software*, 23(2):62–70, 2006.

[56] G. C. Fox and D. Gannon. Workflow in Grid Systems. *Concurrency and Computation: Practice and Experience*, 18(10):1009–1019, 2006.

[57] David Franklin, Jay Budzik, and Kristian Hammond. Plan-based interfaces: Keeping track of user tasks and acting to cooperate. In *7th international conference on Intelligent user interfaces*, 2002.

[58] T. Friese, J.P. Müller, and B. Freisleben. Self-healing execution of business processes based on a peer-to-peer service architecture. In *Proc. of Int. Conf. on Architecture of Computing Systems (ARCS)*, pages 108–123, Innsbruck, Austria, 2005. Springer.

[59] H. Gamboa and V. Ferreira. Widam - web interaction display and monitoring. In *5th International Conference on Enterprise Information Systems ICEIS*, 2003.

[60] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[61] D. Georgakopoulos, M.F. Hornick, and F. Manola. Customizing transaction models and mechanisms in a programmable environment supporting reliable workflow automation. *IEEE Trans. Knowl. Data Eng.*, 8(4):630–649, 1996.

[62] C. Glahn, M. Specht, and R. Koper. Towards a service-oriented architecture for giving feedback in informal learning environments. In *2nd TenCompetence Workshop*, 2007.

[63] K. Goslar, S. Buchholz, Alexander Schill, and H. Vogler. A multidimensional approach to context-awareness. In *International Institute of Informatics and Systemics; Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI2003)*, 2003.

[64] GRAAP Working Group. WS-Agreement Framework. `https://forge.gridforum.org/projects/graap-wg`.

[65] D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving business process quality through exception understanding, prediction, and prevention. In *Proc. of Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 159–168, Roma, Italy, 2001. Morgan Kaufmann.

[66] Guillaume Grondin, Noury Bouraqadi, and Laurent Vercouter. Madcar : An abstract model for dynamic and automatic (re-)assembling of component-based applications. *Component-Based Software Engineering*, pages 360–367, 2006.

[67] C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Trans. Software Eng.*, 26(10):943–958, 2000.

[68] R. Hamadi and B. Benatallah. Recovery nets: Towards self-adaptive workflow systems. In *Proc. of Int. Conf. on Web Information Systems Engineering (WISE)*, volume 3306 of *Lecture Notes in Computer Science*, pages 439–453, Brisbane, Australia, 2004. Springer.

[69] B. Hambling, P. Morgan, A. Samaroo, G. Thompson, and P. Williams. *Software Testing: An ISEB Foundation*. British Computer Society, 2007.

[70] M-A. Hariri, D. Tabary, S. Lepreux, and C. Kolski. Context aware business adaptation toward user interface adaptation. In *Communications of SIWN*, pages 46–52. Springer Verlag, 2008.

[71] John Harney and Prashant Doshi. Speeding up adaptation of web service compositions using expiration times. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1023–1032, 2007.

[72] Julia Hielscher, Raman Kazhamiakin, Andreas Metzger, and Marco Pistore. A Framework for Proactive Self-Adaptation of Service-based Applications Based on Online Testing. In *Service-Wave 2008*. to be published, 10-13 December 2008.

[73] Eric Horvitz and Johnson Apacible. Learning and reasoning about interruption. In *5th international conference on Multimodal interfaces*, 2003.

[74] Gang Huang, Hong Mei, and Fu-Qing Yang. Runtime recovery and manipulation of software architecture of component-based systems. *Automated Software Engineering*, 13(2):257–281, 2006.

[75] International Federation for Information Processing. WG 10.4 on Dependable Computing and Fault Tolerance. http://www.dependability.org/wg10.4/, 2005.

[76] M. C. Jaeger, G. Muhl, and S. Golze. QoS-aware composition of web services: An evaluation of selection algorithms. In *COOPIS 2005 Proc.*, 2005. Cyprus.

[77] Kaustubh R. Joshi, William H. Sanders, Matti A. Hiltunen, and Richard D. Schlichting. Automatic model-driven recovery in distributed systems. In *SRDS '05: Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, pages 25–38, Washington, DC, USA, 2005. IEEE Computer Society.

[78] R. Kazhamiakin, M. Pistore, and M. Roveri. A Framework for Integrating Business Processes and Business Requirements. In *Proceedings of the International Enterprise Distributed Object Computing Conference (EDOC)*, pages 9–20, 2004.

[79] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Network Syst. Manage.*, 11(1), 2003.

[80] Fabio Kon, Manuel Román, Ping Liu, Jina Mao, Tomonori Yamane, a Claudio Magalh and Roy H. Campbell. Monitoring, security, and dynamic configuration with the dynamictao reflective orb. In *Middleware '00: IFIP/ACM International Conference on Distributed systems platforms*, pages 121–143, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[81] Kung-Kiu Lau and Zheng Wang. A taxonomy of software component models. *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, pages 88–95, Aug.-3 Sept. 2005.

[82] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *J. Log. Program.*, 31(1-3):59–83, 1997.

[83] Baochun Li and Klara Nahrstedt. Qualprobes: Middleware qos profiling services for configuring adaptive applications. *Middleware 2000*, pages 256–272, 2000.

[84] Henry Lieberman. An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[85] Heiko Ludwig, Asit Dan, and Robert Kearney. Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. In *Service-Oriented Computing - ICSOC 2004, Second International Conference*, pages 65–74, 2004.

[86] Paul P. Maglio, Rob Barrett, Christopher S. Campbell, and Ted Selker. Suitor: an attentive information system. In *5th international conference on Intelligent user interfaces*, 2000.

[87] Khaled Mahbub and George Spanoudakis. Monitoring WS-Agreements: An Event Calculus-Based Approach. In Luciano Baresi and Elisabetta Di Nitto, editors, *Test and Analysis of Web Services*, pages 265–306. Springer, 2007.

[88] Sheila A. McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, pages 482–496. Morgan Kaufmann, 2002.

[89] Daniel Menascé. QoS issues in Web Services. *IEEE Internet Comp.*, 6(6):72–75, 2002.

[90] Stefano Modafferi, Enrico Mussi, and Barbara Pernici. Sh-bpel: a self-healing plug-in for ws-bpel engines. In *Proceedings of the 1st Workshop on Middleware for Service Oriented Computing, MW4SOC 2006, Melbourne, Australia, November 27 - December 01, 2006*, pages 48–53, 2006.

[91] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. Non-intrusive monitoring and service adaptation for ws-bpel. In *WWW*, pages 815–824, 2008.

[92] Jean-Marie Mottu, Benoit Baudry, and Yves Le Traon. Mutation analysis testing for model transformations. In *proceedings of the European Conference on Model Driven Architecture (ECMDA 06)*, Bilbao, Spain, July 2006.

[93] D. Nesamoney. Bam: Event-driven business intelligence for the real-time enterprise, 2004. DM REVIEW, THOMSON MEDIA.

[94] A. Newberger and A. Dey. Designer support for context monitoring and control. In *Intel Research*, 2003.

[95] B.D. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in odyssey. In *Mobile Networks and Applications 4*, pages 245–254, 1999.

[96] Barbara Pernici and Anna Maria Rosati. Automatic learning of repair strategies for web services. In *ECOWS '07: Proceedings of the Fifth European Conference on Web Services*, pages 119–128, Washington, DC, USA, 2007. IEEE Computer Society.

[97] B. Pernici (Ed). *Mobile Information Systems Infrastructure and Design for Adaptivity and Flexibility*. Springer, 2006.

[98] Nicolas Pessemier, Lionel Seinturier, Thierry Coupaye, and Laurence Duchien. A model for developing component-based and aspect-oriented systems. In Welf Löwe and Mario Südholt, editors, *Software Composition*, volume 4089 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2006.

[99] Marco Pistore and Paolo Traverso. Assumption-Based Composition and Monitoring of Web Services. In Luciano Baresi and Elisabetta Di Nitto, editors, *Test and Analysis of Web Services*, pages 307–335. Springer, 2007.

[100] Marco Pistore, Paolo Traverso, and Piergiorgio Bertoli. Automated composition of web services by planning in asynchronous domains. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, pages 2–11. AAAI, 2005.

[101] Pierre-Guillaume Raverdy, Hubert Le, Van Gong, and Rodger Lea. Dart : a reflective middleware for adaptive applications. In *in OOPSLA'98 13th Workshop : Reflective programming in C++ and Java*, 1998.

[102] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *Proc. of Int. Conf. on Data Engineering ICDE*, pages 1113–1114, Tokyo, Japan, 2005.

[103] M. Salifu, Y. Yu, and B.Nuseibeh. Analysing monitoring and switching requirements using constraint satisfiability. In *Technical Report- ISSN 1744-1986; Department of Computing; Faculty of Maths, Computing and Technology, UK*, 2008.

[104] Mohammed Salifu, Yijun Yu, and Bashar Nuseibeh. Specifying monitoring and switching problems in context. In *15th IEEE International Requirements Engineering Conference*, 2007.

[105] Ahmed Seffah, Peter Forbrig, and Homa Javahery. Multi-devices "multiple" user interfaces: development models and research opportunities. In *Journal of Systems and Software 73*, pages 287–300, 2004.

[106] Lionel Seinturier, Nicolas Pessemier, Laurence Duchien, and Thierry Coupaye. A component model engineered with components and aspects. In Ian Gorton, George T. Heineman, Ivica Crnkovic, Heinz W. Schmidt, Judith A. Stafford, Clemens A. Szyperski, and Kurt C. Wallnau, editors, *CBSE*, volume 4063 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 2006.

[107] S. Sen, B. Baudry, and J.-M. Mottu. On Combining Multi-formalism Knowledge to Select Models for Model Transformation Testing. In *ICST'08: 1st Int. Conf. on Software Testing Verification and Validation*, Lillehamer, Norway, Apr 2008.

[108] Sylvain Sicard, Fabienne Boyer, and Noel De Palma. Using components for architecture-based management: the self-repair case. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 101–110, New York, NY, USA, 2008. ACM.

[109] Savitha Srinivasan, Vikas Krishna, and Scott Holmes. Web-log-driven business activity monitoring. *IEEE Computer*, 38(3), 2005. http://dx.doi.org/10.1109/MC.2005.109.

[110] J.-Z. Sun and J. Sauvola. Towards a conceptual model for context-aware adaptive services. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 27–29, 2003.

[111] Vanish Talwar, Chetan Shankar, Sandro Rafaeli, Dejan Milojicic, Subu Iyer, Keith Farkas, and Yuan Chen. Adaptive monitoring: Automated change management for monitoring systems. In *Proceedings of the 13th Workshop of the HP OpenView University Association (HP-OVUA 2006)*, pages 21–24, 2006.

[112] Abbas Tarhini and Hacène Fouchal. Conformance testing of real-time component based systems. In *ISSADS*, pages 167–181, 2005.

[113] Antti-Matti Vainio, Miika Valtonen, and Jukka Vanhala. Proactive fuzzy control and adaptation methods for smart homes. *IEEE Intelligent Systems*, 23(2):42–49, 2008.

[114] Wim Vanderperren, Davy Suvée, María Agustina Cibrán, and Bruno De Fraine. Stateful aspects in jasco. In Thomas Gschwind, Uwe Aßmann, and Oscar Nierstrasz, editors, *Software Composition*, volume 3628 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2005.

[115] L. Verner. The power of events: An introduction to complex event processing in distributed enterprise systems, 2002. Addison-Wesley Professional.

[116] L. Verner. Bpm: The promise and the challenge, 2004. ACM Queue.

[117] K.E. Wac. Towards qos-awareness of context-aware mobile applications and services. In *In: Proceedings of the On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE (OTM2005), 31 Oct - 4 Nov 2005, Agia Napa, Cyprus. pp. 751-760. Lecture Notes in Computer Science 3760*, pages 751–760. Springer Verlag, 2005.

[118] H. Wächter and A. Reuter. The ConTract model. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 219–263. Morgan Kaufmann, 1992.

[119] Hao Yan and Ted Selker. Context-aware office assistant. In *5th international conference on Intelligent user interfaces*, 2000.

[120] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnamam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Trans. on Software Engineering*, 30(5), May 2004.

[121] L. Zhang and D. Ardagna. SLA based profit optimization in autonomic computing systems. In *ICSOC 2004 Proc.*, pages 173–182, 2004. New York.

[122] Qijun Zhu and Chun Yuan. A reinforcement learning approach to automatic error recovery. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 729–738, Washington, DC, USA, 2007. IEEE Computer Society.