Web Service Adaptation: State of the art and Research Challenges

Chrysostomos Zeginis\textsuperscript{1}, Dimitris Plexousakis\textsuperscript{1}

\textsuperscript{1}Institute of Computer Science, FORTH-ICS
P.O. Box 1385, GR 71110, Heraklion, Crete, Greece
\{zegchris, dp\}@ics.forth.gr

Technical Report 410, ICS-FORTH, October 2010

Abstract

Service-based applications have been attracting a lot of attention from both academia and industry in recent years. Therefore, as these applications need to operate in a highly dynamic and distributed world, they should adapt their behavior to ensure the expected functionality and quality. Many adaptation techniques have been proposed so far focusing on a single layer of the SBA life-cycle.

In this technical report we make a review of the current fragmented state-of-the-art approaches, uncovering the basic research challenges on web service adaptation. Furthermore, we classify these approaches according to adaptation specification and pinpoint the problems stemming from the lack of cross-layer monitoring and adaptation techniques across Service-Based Applications. Finally, we analyze the basic self-* properties of SBAs and propose a tentative research plan for our future work.
# Contents

List of Tables 4
List of Figures 5

1 Introduction 6
  1.1 Web services 6
  1.2 The need for monitoring 7
  1.3 The need for adaptation 8
  1.4 Adaptation Life-cycle 9
  1.5 Key concepts for monitoring and adaptation 11
  1.6 Current approaches and gaps 15

2 State of the art on Service-based Applications’ adaptation 17
  2.1 Adaptation taxonomy 17
    2.1.1 Taxonomy Dimension: Why? 17
    2.1.2 Taxonomy Dimension: Who? 18
    2.1.3 Taxonomy Dimension: What? 18
    2.1.4 Taxonomy Dimension: How? 19
  2.2 Adaptation at Different SBA Layers 19
  2.3 Cross-layer Adaptation 20
    2.3.1 Requirements for Cross-layer Adaptation and Monitoring 21
  2.4 Design for Adaptation 22
    2.4.1 Conceptual model 24
    2.4.2 Main ingredients 25
    2.4.3 Design Guidelines 26
  2.5 Adaptation Specification 28
    2.5.1 Implicit adaptation approaches 30
    2.5.2 Goal-based Adaptation Approaches 35
    2.5.3 Action-based Approaches 41
    2.5.4 Explicit Variability approaches 49
    2.5.5 Comparison of Adaptation approaches 52

3 Self-* properties of Web Services 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>56</td>
</tr>
<tr>
<td>3.2</td>
<td>Types of self-* behavior</td>
<td>57</td>
</tr>
<tr>
<td>3.3</td>
<td>Levels of self-* behavior</td>
<td>58</td>
</tr>
<tr>
<td>3.4</td>
<td>Lifecycle of the self-* services</td>
<td>58</td>
</tr>
<tr>
<td>4</td>
<td>Research challenges and Concluding Remarks</td>
<td>59</td>
</tr>
<tr>
<td>5</td>
<td>Bibliography</td>
<td>62</td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1: Adaptation at different functional SBA layers ........................................... 20
Table 2.2: Relationships between triggers and adaptation strategies ......................... 26
Table 2.3: Comparison of state-of-the-art approaches on Web Service Adaptation ... 53
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The adaptation diagram</td>
<td>10</td>
</tr>
<tr>
<td>1.2</td>
<td>Conceptual Adaptation and Monitoring Framework</td>
<td>12</td>
</tr>
<tr>
<td>1.3</td>
<td>High-level adaptation model</td>
<td>15</td>
</tr>
<tr>
<td>2.1</td>
<td>Adaptation and monitoring in SBAs [32]</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>Problem types and required mechanisms mechanisms within the conceptual A&amp;M framework</td>
<td>22</td>
</tr>
<tr>
<td>2.3</td>
<td>The design for adaptation diagram</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>Service replacement in case of communication failures relying on a robust peer-to-peer service discovery and selection mechanism for alternative services</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Adapters allows achieving protocol replaceability</td>
<td>32</td>
</tr>
<tr>
<td>2.6</td>
<td>MAIS architecture with context management</td>
<td>33</td>
</tr>
<tr>
<td>2.7</td>
<td>Extraction of optimum plan and WS ranking</td>
<td>34</td>
</tr>
<tr>
<td>2.8</td>
<td>A conceptual model of web services' protocol mediation</td>
<td>35</td>
</tr>
<tr>
<td>2.9</td>
<td>A–WSCE: System Architecture</td>
<td>36</td>
</tr>
<tr>
<td>2.10</td>
<td>The Processes with Adaptive Web Services architecture</td>
<td>37</td>
</tr>
<tr>
<td>2.11</td>
<td>METEOR-S architecture overview</td>
<td>38</td>
</tr>
<tr>
<td>2.12</td>
<td>The Enhanced Service-Oriented Architecture</td>
<td>40</td>
</tr>
<tr>
<td>2.13</td>
<td>The Self-healing framework</td>
<td>42</td>
</tr>
<tr>
<td>2.14</td>
<td>The architecture of SH-BPEL Engine</td>
<td>43</td>
</tr>
<tr>
<td>2.15</td>
<td>The core elements of the adaptation architecture</td>
<td>43</td>
</tr>
<tr>
<td>2.16</td>
<td>The architecture of the SCENE platform</td>
<td>44</td>
</tr>
<tr>
<td>2.17</td>
<td>WS-Diamond Overall Architecture</td>
<td>45</td>
</tr>
<tr>
<td>2.18</td>
<td>The architecture of the adapter development tool</td>
<td>48</td>
</tr>
<tr>
<td>2.19</td>
<td>Solution architecture and approach</td>
<td>49</td>
</tr>
<tr>
<td>2.20</td>
<td>Change-oriented service life cycle</td>
<td>50</td>
</tr>
<tr>
<td>3.1</td>
<td>Self-healing Composition Cycle for Web [18]</td>
<td>57</td>
</tr>
</tbody>
</table>
Chapter 1

1 Introduction

1.1 Web services

A Web Service is a self-describing, self-contained software module available via a network, such as the Internet, which completes tasks, solves problems, or conducts transactions on behalf of a user or application. Web services constitute a distributed computer infrastructure, made up of many different interacting application modules trying to communicate over private or public networks to virtually form a single logical system.

A more precise definition is published by the World Wide Web Consortium (W3C):

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

In essence, a web service provides an interface defined in terms of XML messages and that can be accessed over the Internet (or, of course, an Intranet). It is an application that exposes a function which is accessible using standard Web technology and that adheres to Web services standards. This is significant because Web services are developed for and deployed onto any platform using any programming language.

A Web service can be: (i) a self-contained business task, such as funds withdrawal or funds deposit service; (ii) a full-fledged business process, such as the automated purchasing of office supplies; (iii) an application, such as a life insurance application or demand forecasts and stock replenishment; or (iv) a service-enabled resource, such as access to a particular back-end database containing patient medical records. Web services can vary in function from simple requests (e.g. credit checking and authorization, pricing enquiries, inventory status checking, or a weather report) to complete business applications that access and combine information from multiple
sources, such as an insurance brokering system, an insurance liability computation, an automated travel planner, or a package tracking system. [43]

1.2 The need for monitoring

Once services and business processes become operational, their progress needs to be managed and monitored to gain a clear view of how services perform within their operational environment, take management decisions, and perform control actions to modify and adjust the behavior of Web-services-enabled applications. Service-level monitoring is a disciplined methodology for establishing acceptable levels of service that address business objectives, processes, and costs.

The term “monitoring” has been widely used in many disciplines and in particular in service-oriented design and engineering. Monitoring has different interpretations depending on the role the monitoring process plays in the system life-cycle. In the case monitoring is used to discover problems in the application execution, it is defined as a problem of observing the behavior of a system and determining if it is consistent with a given specification [10].

A monitor is a program that observes the execution of another program and is being defined and implemented on the basis of the monitoring specification that describes the properties and events to be observed. The actual information is being perceived and delivered to the monitor by a set of sensors. This information is being processed by the monitor in order to decide whether this information corresponds to the monitored event or property (e.g. certain QoS parameter value exceeded bound).

The service monitoring phase concerns itself with service-level measurement; monitoring is the continuous and closed loop procedure of measuring, monitoring, reporting, and improving the QoS of systems and applications delivered by service-oriented solutions. Service monitoring involves several distinct activities including logging and analysis of service execution details, obtaining business metrics by analyzing service execution data, detecting business situations that require management attention, determining appropriate control actions to take in response to business situations, whether in mitigating a risk or taking an opportunity, and using historical service performance data for continuous service improvement.

The service monitoring phase targets continuous evaluation of service-level objectives, monitoring the services layer for availability and performance, managing security policies, tracking interconnectivity of loosely coupled components, analyzing the root cause and correcting problems. To achieve this objective, service monitoring requires that a set of QoS metrics is gathered on the basis of SLAs, given that an SLA
is an understanding of service expectations. In addition, workloads need to be monitored by the service provider to ensure that the promised performance level is being delivered, and to take appropriate actions to rectify non-compliance with an SLA, such as reprioritizing and reallocating resources.

To determine whether an objective has been met [43], SLA-available QoS metrics are evaluated based on measurable data about a service (e.g. response time, throughput, availability, and so on), performance during specified times, and periodic evaluations. SLAs include other observable objectives, which are useful for service monitoring. These include compliance with differentiated service-level offerings, i.e. providing differentiated QoS for various types of customers, individualized service-level offerings and requests policing which ensures that the number requests per customer stays within a predefined limit. All these also need to be monitored and assessed. A key aspect of defining measurable objectives is to set warning thresholds and alerts for compliance failures. For instance, if the response time of a particular service is degrading then the client could be automatically routed to a back up service.

1.3 The need for adaptation

The dynamic nature of the business environment requires Web services to be highly reactive and adaptive. They should be equipped with mechanisms to ensure that they are able to adapt to meet changing requirements. In fact, services are subject to constant changes and variations. Services can evolve due to changes in structures, in behavior and policies. Such changes can be identified, detected, and foreseen in the SBA during the monitoring of the application execution and its environment.

Adaptation can be defined as a process of modifying SBAs so as to satisfy new requirements dictated by changes of the environment on the basis of Adaptation Strategies designed by the system integrator. An adaptable SBA is an application that continuously monitors and modifies itself on the basis of these strategies. [28]

In [10] the authors describe the main purposes of service adaptation:

**Ensuring the interoperability:** While standardization in Web services makes interoperability easier, adaptation still remains necessary. Adaptation is an important functionality that should be offered to enable integration inside and across enterprise boundaries. We need to generate a service that mediates the interactions among two services with different interfaces and protocols so that interoperability can be made effective. The need for adapters in Web services comes from two sources: (1) the heterogeneity at the higher levels of the interoperability stack (e.g., at business-level interfaces and protocols), and (2) the high number and diversity of clients, each of which can support different interfaces and protocols.
**Optimization:** The demand for quickly delivering new applications and adaptively managing them is a business imperative today. The QoS offerings in an application may change, new service providers and business relationships may emerge and existing ones may be modified or terminated. Furthermore, the general performance of the service is related to service velocity the time related aspect of business operations, such as service cycle time, round trip delays, wait time between events. The challenge, therefore, is to design robust and responsive systems that address these changes effectively while continually trying to optimize the operations of a service provider.

**Recovery:** Various faults can occur relatively often and unexpectedly in distributed systems. It is necessary to handle faults reported during execution of the composition instance when monitoring business process. In fact, Web services compositions often implement business critical processes whose correct and uninterrupted operation is paramount. Therefore, to achieve dependable business processes, Web services compositions have to be made reliable. Reliability can be defined as the continuity of correct service delivery. This implies zero or, at worst, relatively few failures and rapid recovery time.

**Context change:** Services are constituted by reusable software components. The adaptation goal is to optimize the service function of their execution context. The same adapter searches and applies the possible adaptation solutions: component customization, insertion, extraction or replacement.

### 1.4 Adaptation Life-cycle

Differently from classical SBAs, the distinguishing feature of the adaptable SBAs is the support for accommodating and managing various changes occurring in the application or in its context. This capability extends the traditional view on the Service-Based Application and requires monitoring and adaptation to be the core elements of the application life-cycle. Figure 1.1 depicts the basic steps of the adaptation life-cycle.
Adaptation is a process of modifying a Service-Based Application in order to satisfy new requirements and to fit new situations dictated by the environment. This general definition becomes more concrete when we consider different forms of adaptation (figure 1.1): Proactive (to prevent future problems proactively identifying and handling their sources), Reactive (to handle faults and recover from problems reported during execution of an SBA instance or a set of instances), and Postmortem (to modify (or evolve) the system at design time or when it is stopped). With respect to the human involvement, as highlighted in the figure, we distinguish the following two extreme types of adaptation: self-adaptation and human-in-the-loop adaptation. Self-adaptation is an adaptation process that is executed without any external human intervention. In this case all adaptation steps, decisions, and actions are performed by the SBA autonomously. This also assumes that all the necessary mechanisms to enact adaptation strategies are built into the application. When the adaptation process assumes any form of human intervention, one deals with human-in-the-loop adaptation. This intervention may have different forms and take place at the different phases of the adaptation cycle.

The adaptation cycle consists of the following principle steps [2]:

- **Identify adaptation need**
  - Define adaptation requirements: Identify the aspects of the SBA model that are subject to change, and the expected outcome of the adaptation process.
  - Define requirements to the monitoring subject: In order to satisfy the adaptation requirements, this practice focuses on specifying what artifacts are expected to be monitored.
Define monitored property: Specify which properties of the monitoring subject should be monitored.

Provide monitoring functionality: Monitoring functionalities that satisfy the monitoring requirements are provided through monitoring realization mechanism.

Collect monitoring results for adaptation: Results of monitoring are collected and analyzed.

Trigger adaptation: Evaluate the results from the monitoring analysis against adaptation requirements. If the need for adaptation is identified, send a request to trigger adaptation process.

Identify adaptation strategy

Design adaptation strategy: Design the ways through which the adaptation requirements are satisfied.

Select adaptation strategy: Decide which particular adaptation strategy to be chosen based on the specific adaptation needs.

Enact adaptation

Perform adaptation: The actual adaptation process is performed through adaptation realization mechanisms based on the selected adaptation strategy.

1.5 Key concepts for monitoring and adaptation

At the high level of abstraction, the adaptation and monitoring framework can be described by the concepts represented in figure 1.2. 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 Adaptation and Monitoring framework [10]. 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.
A generic adaptation and monitoring framework consists of the following elements [2], [10]:

**Monitoring Mechanisms** provide the actual realization of the methods for continuous observing and detecting relevant Monitoring Events; they identify the "How?" dimension of the monitoring process.

**Monitoring Events** deliver the information about the application evolution and changes in the environment. These events define the "What?" dimension of the monitoring process: they are used to indicate whether the SBA is executed and evolves in a normal mode, whether there are some deviations or even violations of the desired functionality.

The events identified during the monitoring process carry the information about potential changes that the system and / or the underlying platform should perform in order to adapt to a new situation. The relation between the monitoring events and the changes of SBA are defined by the **Adaptation Requirements and Objectives**. These requirements identify the "What?" dimension of the adaptation process as they describe the goals and needs the adapted SBS should verify. Especially they define the expected outcome of the adaptation process.

The **Adaptation Strategies** define the possible ways to achieve these requirements and needs given the adaptation mechanisms made available in different functional layers of SBA. The adaptation strategies should guarantee coherent and integrated cross-layer management of these adaptation techniques. Example of adaptation strategies are *reconfigure* (i.e. modify the current configuration parameters of the SBA), *substitute, compensate* (i.e. remove the negative effects of previous
executions), replan (i.e. modify the structure and the model of the SBA), re-compose, re-negotiate. Adaptation strategies can be further classified according to the location, the used methodology, and the way the strategy is specified.

- **Location** determines the placement of changes made by adaptation process:
  
  - *Horizontal placement* defines the scope of adaptation and says whether changes are local (restricted to a service or to specific clients) or global (large-scale transformational changes).
  
  - *Vertical placement* defines the affected functional SBA layers; Service Composition-Level changes which affect the behavioral protocols and/or operational semantics of SBA; Business Process-level changes which affect business rules and requirements, organizational models, clients, and even entire value chain; Cross-layer changes which affect different functional layers.

- **Adaptation Methodology** determines the timing, the distribution and the direction of the adaptation:
  
  - *Timing* defines the moment when adaptation takes place and we can distinguish three different types of adaptation (reactive, proactive, post-mortem) as defined in chapter 1.4.
  
  - *Direction* of the adaptation decides if we have to go forward or backward. Thus, we distinguish two types of adaptation: *forward adaptation*, where the system is directed to a new state, where adaptation requirements are met and *backward adaptation*, where the system reverts to a previous stable state.
  
  - *Distribution* of the adaptation distinguished between *centralized adaptation*, where adaptation actions are defined and executed on all the affected components in an integrated way and *distributed adaptation* performed locally and then propagated among components.

- **Adaptation specification** defines the notations needed to specify the adaptation strategies and the particular actions representing these strategies. It can be procedural (concrete actions to be executed), declarative (certain goals to be achieved) and hybrid. The notation can be either implicit or explicit. Implicit annotation means that adaptation strategies are hard-coded in the system and cannot be changed without modification of the adaptation mechanisms. Explicit adaptation specification, allows the designer to explicitly state the adaptation requirements or instructions. We can distinguish the following explicit adaptation specification forms:
- **Action-based specification** consists of situation-action rules that defined which actions will be executed in certain situations.

- **Goal-based specification** establishes performance objectives, leaving the system to determine which actions required to achieve these objectives.

- **Utility function-based** specification exploits utility functions to qualify and quantify the desirability of different adaptation alternatives and then on the fly decide for the “best” feasible state.

- **Explicit variability approach** defines the association points where the adaptation should take place, with a set of alternatives that define different possible implementations of the corresponding application part.

- **Adaptation action** is an action performed to modify the current state of the system so as to fulfill the adaptation requirements. These actions may be further classified according to the subject of the adaptation and the scope: service instance adaptation actions (retry, negotiate SLA, duplicate service, substitute service), flow instance adaptation actions (substitute flow, undo, redo, skip / skip to), service class actions (change SLA, suggestion for service re-design), flow class actions (re-design/replan, change service selection logic, change service registry, change platform).

The **Adaptation Mechanisms** provided at various functional SBA layers offer a means to actually implement the adaptation strategies in integrated and coordinated manner. These mechanisms include **Realization mechanisms** that perform actual adaptation actions and **Decision Mechanisms** that takes important decisions for the adaptation process.

During adaptation various elements can be modified. Thus, adaptation has different **Adaptation Subjects**. In addition, adaptation involves many **Adaptation Actors**, playing various roles in the procedure.

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 Figure 1.2 and coordinate the work of the framework and realize the relations among them. In particular:

**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).

Figure 1.3 depicts how the adaptation concepts described above cooperate and the roles plays each of them during the adaptation process.

Figure 1.3: High-level adaptation model

1.6 Current approaches and gaps

Monitoring and Adaptation systems have gained a lot of interest during the past few years both in research community and in industry. However, current approaches are still insufficient, because they mostly focus on the infrastructural layer or on specific aspects of the behavior of service compositions, such as binding and enactment. Comprehensive approaches covering higher-level aspects of the service composition and coordination layer and the business process management layer are still to be defined. Some other gaps of the current approaches are the following [29]:

- Most of them are fragmented, as they address only specific problems, particular application domains, and particular types of applications and systems. Also, in many of them the monitoring solutions are often isolated from the adaptation needs and approaches.
• Most of the approaches implement adaptation reactively, when the problem has already happened and they don’t prevent the problem from happening.

• Most of the state-of-the-art solutions for monitoring and adaptation often omit the role of application context. SBAs are commonly designed to be served to various types of users, and therefore they should be able to be customized and personalized according to information collected by them. Also SBAs should allow users participate to the current activities.

As we can see, there is open horizon on Web Service Adaptation. This topic offers many perspectives for the future and researchers will have the chance to propose many more approaches towards adaptive services.
Chapter 2

2 State of the art on Service-based Applications’ adaptation

2.1 Adaptation taxonomy

Adaptation approaches can be classified according to different criteria. In this chapter an adaptation taxonomy is defined that distinguishes approaches by Why, Who, What and How adaptation takes place [29].

2.1.1 Taxonomy Dimension: Why?

This taxonomy dimension reply to the question Why an adaptation process takes place?, i.e. which is the motivation behind adaptation. There are five adaptation types depending on the goal of adaptation process:

- **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 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.

### 2.1.2 Taxonomy Dimension: Who?

This taxonomy dimension reply to the question Who is behind an adaptation process?, i.e. who are the basic adaptation actors. There are four types of actors:

- **Adaptation Requestor** characterizes the stakeholders, who define the adaptation requirements for the SBA.

- **Adaptation Designer**, who defines the adaptation strategies to achieve the adaptation requirements.

- **Adaptation Initiator**, who initiate the modification of the application in reaction to the identified changes.

- **Adaptation Executor** is responsible for executing adaptation actions defined by the chosen adaptation strategy.

### 2.1.3 Taxonomy Dimension: What?

This taxonomy dimension replies to the question What is the adaptation target? There are three elements of this taxonomy: Subject of adaptation, i.e. the entity that would be modified by the adaptation process, Adaptation Aspect, i.e. particular concerns of the adaptation process, Adaptation Scope, i.e. temporary or permanent adaptation. For the subject of adaptation we can distinguish four types at the higher level of abstraction:

- **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.

### 2.1.4 Taxonomy Dimension: How?

This taxonomy dimension reply to the question How adaptation can be achieved and implemented?, i.e. which are the specific strategies and mechanisms used to implement the adaptation, analyzed in chapter 1.5.

### 2.2 Adaptation at Different SBA Layers

An SBA [31] can be presented by three functional layers, namely the *Business Process Management layer* (BPM), the *Service Composition and Coordination layer* (SCC) and the *Service Infrastructure layer* (SI). BPM is the highest level layer where the entire business process is considered as workflow and the business activities as the constituent services. In this layer application activities, constraints and requirements are described without going into details. SCC is the intermediate layer between BPM and SI. It refines the basic workflow designed in BPM by composing suitable services accomplishing the construction of the whole SBA. At the lowest level we have SI layer, which provides the underlying run-time environment for the composed SBA. It also stores all the available services in a service registry, from where the SCC layer discovers and gets the constituent services. Figure 2.1 shows how monitoring and adaptation processes operate in these three layers.

![Figure 2.1: Adaptation and monitoring in SBAs [32]](image-url)
Although current approaches cover a wide spectrum of adaptation in each of the three SBA layers, none of them is complete to cover all layers. They usually considered in isolation from each other; focus on a local solution for a specific adaptation requirement without taking into account the effects to the other SBA layers. The following table 2.1 demonstrates the adaptation requirements, strategies and mechanisms in each of the three SBA layers.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Requirements</th>
<th>Strategies/actions</th>
<th>Mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPM</td>
<td>optimize process to meet violated KPI; recover from unforeseen execution; process customization</td>
<td>modify business process control flow (add, delete, replace process tasks and process fragments), modify business process data flow (change data dependencies, values); process model re-design</td>
<td>ad-hoc modifications (performed by business analysts); evolution</td>
</tr>
<tr>
<td>KPI adaptation</td>
<td>adjust to changing business goals, business context, ASN elements</td>
<td>add or remove KPI, change KPI values</td>
<td>ad-hoc modification; negotiation; evolution</td>
</tr>
<tr>
<td>ASN adaptation</td>
<td>optimize costs, transactionality; accommodate to ASN changes</td>
<td>change transaction protocol, change service; re-negotiate for an offering</td>
<td>ad-hoc modifications; negotiation mechanisms; evolution</td>
</tr>
<tr>
<td>SCC</td>
<td>Compositional-related adaptation: adjust to changed process model or KPI, process optimization, recovery</td>
<td>re-composition; control/data flow changes; FPM changes</td>
<td>dynamic discovery and binding; negotiation</td>
</tr>
<tr>
<td>Service-related adaptation</td>
<td>changes or failures of constituent services; optimization; SLA violation</td>
<td>replace a service; re-execute a service; re-negotiate QoS</td>
<td>platform-specific; reputation management</td>
</tr>
<tr>
<td>SI</td>
<td>Service discovery-related adaptation: optimization, business requirements</td>
<td>change registry; update registry (new services, new descriptions); change discovery mechanism; change selection mechanism</td>
<td>automated composition; model-driven transformations; fragmentation</td>
</tr>
<tr>
<td>Service realization-related adaptation</td>
<td>optimization, adjust to infrastructural failures</td>
<td>modify/re-configure service platform (software updates, OS changes, virtual machines, physical platforms); modify/re-configure service resources (allocation/release of resources, load balancing); adapt resource management (change resource brokering mechanisms, re-configure grid application, re-execute application)</td>
<td>ad-hoc changes; self-* techniques</td>
</tr>
</tbody>
</table>

Table 2.1: Adaptation at different functional SBA layers

2.3 Cross-layer Adaptation

The problem of monitoring and adaptation of various types of software systems 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. One of the key issues here is that the proposed approaches are very fragmented: they address only specific problems peculiar to a particular aspect of the SBA functioning and a particular functional SBA layer, i.e., business process management layer, service composition and coordination layer, and service infrastructure layer. In complex realscale applications, however, the realization of different SBA layers may be highly interleaved: different artifacts at one layer may
refer to the same artifacts in another layer, while such relations are ignored by the isolated monitoring and adaptation solutions. As a consequence, wrong problems are detected, incorrect decisions are made, and the modifications at one level may damage the functionality of another layer.

2.3.1 Requirements for Cross-layer Adaptation and Monitoring

Lack of cross-layer integrated monitoring and adaptation principles and mechanisms may lead to a variety of problems and hazards in SBAs. We distinguish the following classes of problems: lack of alignment of monitored events, lack of adaptation effectiveness, lack of compatibility, and lack of integrity. [32]

Lack of Alignment of Monitored Events
In many cases, if the monitoring is performed by specific mechanisms provided at different layer in isolation from each other, then the corresponding events are not aligned and the critical information is not adequately propagated across layers. This may lead to the situation where the source of the monitored problem is identified incorrectly; to the situation where the same problem results in separate events at different layers and then triggers different (possibly contradictory) adaptations, etc. Consider an example, where due to a bottleneck in the service middleware the performance of the involved services goes down. This may be detected independently by the infrastructure-level monitor and by the composition monitor that observes the performance of the whole composition. As a result, different adaptation actions may be triggered at different layers (e.g., composition fragmentation and parallelizing of tasks at the SCC layer and resource re-allocation at the SI layer). Furthermore, the actions may be contradictory (e.g., the new resource allocation discards the expected effects of the parallelization).

Lack of Adaptation Effectiveness
Another possible problem of monitoring and adaptation performed at different layers in isolation is that the adaptation activities may fail to achieve the expected effect, since they do not take into account the properties of other layers. For example, at the SCC layer in order to reduce execution the adaptation aims to execute independent tasks in parallel by delegating them to different services. In order to achieve this effect it is necessary that those services are independent also at the SI layer (which may not be the case in Grid).

Lack of Compatibility
The problem of cross-layer compatibility of adaptation activities refers to the situation, where the adaptation performed at one functional SBA layer is not
compatible with the requirements and constraints posed by the application design at other layers.

**Lack of Integrity**
A problem of cross-layer adaptation integrity deals with a situation, where it is not enough to perform the adaptation at a particular layer only, but several actions at different layer should be performed. For instance, when the adaptation is performed at the BPM layer (e.g., a business process is changed), it is necessary to propagate the changes also to the other layers (e.g., change the compositions that manage corresponding process instances and/or perform some compensation actions; negotiate, bind, and adjust to the new formats and protocols at the SI layer, etc.).

Figure 2.2 shows these problems in the conceptual Adaptation and Monitoring framework and the mechanisms required to solve them.

![Figure 2.2: Problem types and required mechanisms within the conceptual A&M framework](image)

**2.4 Design for Adaptation**

Modern service-based applications have to be adaptable to unforeseen changes in the functionality offered by component services, to their unavailability, to decreasing performances, to the current context of use as well as to specific requirements and needs of the specific users. Most of the existing approaches address this issue by hard coding in the infrastructure supporting the execution of service-based applications a
limited number of adaptation strategies that are triggered only when some specific and known events happen. Adaptation of service-based applications works properly only in the case the application is designed to be adaptable. Furthermore, service-based applications have to face very different adaptation needs, including customization, recovery and repair, self-optimization and context-driven adaptation.

For service-based applications, to be designed in a way to support adaptability and to support the above diversity of problems, a number of design principles and guidelines that are suitable to enable adaptation is presented. At design-time it is necessary to consider possible adaptation alternatives (different strategies and their properties) and different adaptation triggers that motivate the application adaptation, as well as to relate adaptation triggers and adaptation strategies together.

Design for Monitoring and Adaptation is a design process specifically defined to take the necessity for the SBA adaptation into account. It extends the design phases of the “classical” SBAs with all the activities that aim to incorporate into the application or into the underlying execution platform the facilities and mechanisms necessary for the adaptation and monitoring process. While concrete mechanisms and activities necessary to enable SBA adaptation vary depending on a particular form of adaptation (such as context-aware adaptation, customization, optimization, recovery) and the realization of a particular approach (e.g., autonomous vs. human-in-the-loop adaptation, run-time vs. design-time), general design steps specific to the adaptable SBA may be defined as follows:

- **Define adaptation and monitoring requirements.** Based on the application requirements and key quality properties, it is necessary to define the requirements and objectives that should be satisfied when certain discrepancy with respect to the expected SBA state, functionality or environment is detected. More precisely, the monitoring requirements specify what should be continuously observed, and when the discrepancy becomes critical for the SBA. The adaptation requirements describe the desired situation, state, or functionality, to which the SBA should be brought to. Typically, the adaptation and monitoring requirements correspond to various SBA quality characteristics that range from dependability, to functional and behavioral correctness, and to usability. In many cases monitoring requirements are derived directly from the adaptation requirements: the monitoring is often performed with the goal to identify the need for adaptation and to trigger it. Definition of adaptation and monitoring requirements is not explicitly addressed by the existing requirements engineering approaches; these requirements are implicitly identified and mapped to the corresponding capabilities in ad-hoc manner.

- **Identify appropriate adaptation and monitoring capabilities.** When the adaptation and monitoring requirements are defined, there is a need to identify
the possible candidates for their implementation. These refer to the existing adaptation and monitoring frameworks and tools provided at different functional SBA layers and to various mechanisms enabled at different layers for more general purposes, such as online testing [28], data and process mining for monitoring purposes, and service discovery, binding and automated composition [43] for adaptation purposes.

- **Define monitoring properties and adaptation strategies.** Requirements and capabilities identified in previous steps are used to provide concrete monitoring and adaptation specification for a given SBA. These specifications may be given implicitly when they are hard-coded within the given approach or explicitly. For instance, when one deals with the recovery problem, a typical implicit monitored property refers to the failures and exceptions not managed by the application code. Accordingly, the self-optimization approaches often rely on the predefined threshold for certain quality of service properties for triggering adaptation need; the corresponding adaptation strategy (e.g., re-composition) is also often predefined.

- **Incorporate adaptation and monitoring mechanisms.** Based on the above specifications, the adaptable SBA is extended with the corresponding monitors and adaptation mechanisms. Depending on the mechanisms, this extension may require integrating the monitoring and/or adaptation functionalities into the SBA code or into the underlying execution platform. A typical example of the former approach is presented in [8]: the underlying BPEL process is augmented with the calls to a special proxy that evaluates the monitored properties. On the other side, monitoring approaches presented in [43],[36] as well most of approaches to Business Activity Monitoring, rely on the mechanisms for generating monitors independent from the application and on the specific tools respectively.

### 2.4.1 Conceptual model

Figure 2.3 depicts all the concepts participate during the Design for Adaptation process [2]. The Adaptable Service-Based Application is associated with the Monitoring Requirements and Adaptation Requirements, which define when the changes in the application functionality or environment become critical and what we should achieve in that case respectively. From these requirements one should derive the Monitored Properties and Adaptation Strategies achieving them. In order to be monitored, the application is associated with the Monitors that continuously observe the monitored properties representing critical changes in the SBA functionality or environment. The Adaptation Mechanisms are identified and incorporated in order to achieve the defined strategies.
There exists a wide range of adaptation strategies to be used by different approaches. In a simple case, adaptation targets modification of the application parameters (e.g., re-configuration, re-negotiation of the SLAs, substitution of one failed or underperforming service with another one) without changing its structure. In more complex cases, the adaptation involves also modification of the application structure (e.g., re-compose the services, re-plan the underlying process, or introduce specific activities that compensate the incorrect results achieved by the faulty execution).

![Figure 2.3: The design for adaptation diagram](image)

### 2.4.2 Main ingredients

The main elements for the design for adaptation process are the adaptation strategies and the adaptation triggers [44]. The former represent the alternative ways for performing SBA adaptation in different situations, while the latter characterize those critical situations. Different alternatives may have different functionalities, characteristics, and consequences, and its suitability for dealing with a specific change can be strictly related to the context and the whole application functional and non-functional requirements. Then the identification of the most suitable adaptation strategy to activate can be a complex issue since different system characteristics have to be considered. In the next subsections, guidelines to support this selection are provided.

**Adaptation strategies**

In order to avoid the application performance degradation, it is necessary to identify the most suitable adaptation strategy that is able to maintain aligned the application behavior with the context and system requirements. Among the adaptation strategies, it
is possible to distinguish domain-independent (applicable in almost every application context) or domain-dependent strategies (limited to specific execution environments). The most common domain-independent adaptation strategies are service substitution, re-execution, (re-)negotiation of QoS properties, (re-)composition of services, compensation, trigger evolution, log/update adaptation information and fail.

**Adaptation triggers**

The adaptation in SBA may be motivated by variety of factors, or triggers. Such triggers may concern the component services or the context of SBAs. As for the former, one can identify such factors as changes in the service functionality or changes in the service quality. As for the contextual triggers, one can distinguish changes in the business context, changes in the computational context, and changes in the user context.

Each trigger can be associated with a set of adaptation strategies that are suitable to re-align the application within the system and/or context requirements. In order to select the adaptation strategy to apply, it is necessary to consider that adaptation triggers may be associated with different requirements that are important for designing and performing adaptation, such as scope of the change (whether the change affects only a single running instance of the SBA or all of them); impact of the change (the possibility of the application still to accomplish its current task). Depending on these parameters different strategies may apply (Table 2.2).

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Adaptation strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes in the service functionality</td>
<td>Service Substitution, Re-execution, Re-negotiation, Re-composition, Compensation, Fail</td>
</tr>
<tr>
<td>Changes in the service quality</td>
<td>Service Substitution, Re-Negotiation</td>
</tr>
<tr>
<td>Changes in the business context</td>
<td>Service Substitution, Re-Negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information</td>
</tr>
<tr>
<td>Changes in the computational context</td>
<td>Service Substitution, Re-negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information</td>
</tr>
<tr>
<td>Changes in the user context</td>
<td>Service Substitution, Re-negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information</td>
</tr>
</tbody>
</table>

Table 2.2: Relationships between triggers and adaptation strategies

**2.4.3 Design Guidelines**

In order to design adaptable SBAs, it is necessary to relate adaptation triggers and adaptation strategies together. This may be done in various ways as defined in [44].
First, this may be done by hard-coding the corresponding elements directly in the main logic of SBAs. On the one hand, such approach does not require any specific tool and mechanism on the side of the design and execution infrastructure. On the other hand, this overloads the logic of the application, thus making it error-prone and difficult to maintain, and requires ad-hoc and non-reusable solutions when not supported by the SBA language.

Second, the adaptation logic may be hard-coded in the SBA infrastructure. While this approach clearly follows principle of separation of concerns, it is not flexible and, therefore, is hard to change it when a specific adaptation need or application domain is dealt with.

Finally, it is possible to provide design patterns and tools that allow for flexible and transparent modeling and integrating adaptation strategies and triggers. On the one hand, this allows adaptation designers to focus solely on the adaptation aspect. On the other hand, this would allow for the flexibility and reuse of the adaptation mechanisms. In particular, it is necessary to come up with the principles and guidelines for:

- **Modeling adaptation triggers**, i.e., both the situation when the adaptation is needed (monitored property) and the specific adaptation need.
- **Realizing adaptation strategies**. This includes modeling strategies, their properties, and their aggregation, and relating them to the underlying mechanisms and run-time infrastructure.
- **Associating adaptation strategies to triggers**. We have already demonstrated how the scope and impact of change influence this relation. Other factors may include autonomy (i.e., if the adaptation should be done without human involvement) or performance (e.g., how fast an adaptation strategy is).

One of the key aspects cross-cutting these design tasks is the dynamicity of the environment with respect to the adaptation problem. This refers (i) to the diversity of specific adaptation needs and (ii) to the diversity of factors the adaptation strategies depend on. According to this distinction, the following design approaches may be defined:

- **Built-in adaptation**. If possible adaptation needs and possible adaptation configurations are fixed and known a priori, it is possible to completely specify them at design time. The focus is on specifying situations, under which adaptation is triggered, and the concrete actions to be performed. The specification may be performed by extending the standard SBA notations (e.g., BPEL) with the adaptation-specific tools using ECA-like (event-condition-action) rules, variability modeling, or aspect-oriented approaches. Typical strategies suitable for such adaptations are: service substitution (by selection
from predefined list of options), re-execution, compensation, re-composition (by using predefined variants), fail.

- **Abstraction-based adaptation.** When the adaptation needs are fixed, but the possible configurations in which adaptation is triggered, are not known a priori, the concrete adaptation actions cannot be completely defined at design time. In such a case, a typical pattern is to define a generic model of an SBA and a generic adaptation strategy, which are then made concrete at deployment/run-time. For example, abstract composition model, where concrete services are discovered and bound at run-time based on the context; defining at design time a composition goal or utility function, which is then achieved or optimized by dynamic service re-composition at runtime based on a specific environment and available services. Here the focus should be on the design of such abstract models (e.g., for specifying abstract process models or composition goals) and on the design of parameterized adaptation mechanisms. Strategies that may be used for such adaptation are service concretization, service substitution (by dynamic discovery), re-composition (based on predefined goal/utility function), re-negotiation.

- **Dynamic adaptation.** Finally, it is possible that adaptation needs that may occur at run-time are not known or cannot be enumerated at design time. In such a case, it is necessary to provide specific mechanisms that select and instantiate adaptation strategies depending on a specific trigger and situation. The examples of the scenarios, where such adaptation is needed may include modifications or corrections of business process instances via ad-hoc actions and changes performed by business analyst, changes in the user activities that entail modification of current composition and creation of new ones. At runtime, these mechanisms are exploited to (i) identify one or more suitable adaptation strategies depending on a concrete situation, (ii) define concrete actions and parameters of those strategies, and (iii) execute them using the appropriate mechanisms. This type of adaptation may be built on top of the others to realize specific adaptation needs; the focus, however, is on the mechanisms for extracting specific adaptation strategies and actions at runtime. Accordingly, different strategies may apply here: re-composition, service substitution, and compensation, re-execution, evolution, fail. The realization mechanisms, however, are different; they may require active user involvement (e.g., for making decisions, for performing ad-hoc changes, etc.).

### 2.5 Adaptation Specification
A very important aspect to build adaptable service-based systems is the way the adaptation mechanisms are specified and configured. Particularly the adaptation specification takes into account two relevant aspects: when the adaptation should take place and how it would be performed. The first aspect specifies the application variability, while the second is used to drive the adaptation management.

The moment when the adaptation takes place may be defined either implicitly or explicitly. For the implicit specification, the relevant information is "predefined" in the platform and cannot be changed, without modifying the adaptation mechanisms. At design-time only abstract operations and requirements are defined without choosing concrete services to be chosen during service composition. At deployment-time the platform selects and composes the "best" candidate services from the service registry. Thus, for the implicit specification, the point of adaptation is predefined during the deployment time.

For the explicit specification, the adaptation point is predefined by the designer of the service-based application. There are four approaches how this point is defined at design-time. First, the point may be associated to a certain control block (e.g. scope in BPEL). Second, it can be associated to a certain event (e.g. reception of a message or a timeout, exception processing). Third, the moment when the adaptation should take place can be defined by a certain state of the application and its environment. Finally, the adaptation point may be associated to more complex events, such as violation of a complex behavioral requirement.

As far as the explicit specification of adaptation approaches is concerned, we can classify them in three different types: goal-based approaches, action-based or event-based approaches and approaches based on explicit variability.

Goal-based approaches describe the adaptation activities at a higher level. They define the certain goal to be reached by the system, without going into details which actions will be performed by the system or the middleware in order to achieve this goal. However, the designer or the user has the ability to guide the adaptation process by defining a set of requirements.

The specification of action-based approaches is defined as a situation-action or event-action rule, which exactly specifies what to do being in a certain situation or upon occurrence of a certain event. These actions may include:

- Adaptation actions, e.g. re-execute service invocation, re-bind service, substitute service, re-negotiate SLA, rollback to previous stable state, re-compose set of services, halt.
- Management actions, e.g. notify, log.

The specification of the approaches based on explicit variability is defined as followed: The identified variation point is associated with a set of alternative possible
implementations of the corresponding application part. For example, in business processes this corresponds to a set of potential flows. Furthermore the specification may include selection guidelines for the possible routes, e.g. preferences or ranking rules, specification or relations and dependences between different alternatives.

Based on the above distinction, the surveyed approaches will be classified into the following groups [22],[10]:

- implicit approaches
- goal-based approaches
- action-based approaches
- approaches based on explicit variability

2.5.1 Implicit adaptation approaches

In this type of approaches the decisions of the adaptation moment and the actions to be performed are predefined by the adaptation framework. A typical example of such approaches is dynamic service composition, where the services that constitute a composite service are selected dynamically. In addition, self-healing systems, where the recovery activities are somehow hard-coded, can be classified in this group. In these approaches the role of the design activities is to provide richer and more complete descriptions of the services and compositions in order to support and simplify the decisions made at run-time automatically.

Quality-driven service composition

In [60],[13] the authors propose an implicit approach towards dynamic service composition based on multi-dimensional optimization of quality of service metrics. The goal of these approaches is the optimization of service composition. They are applied on Service Composition layer. In these approaches the composed process is designed as a workflow of composing elementary tasks. At run-time a concrete elementary service is selected for a particular task from a set of services that provide the same functionality but differ in quality characteristics. The authors identify different sets of the relevant quality properties, such as price, duration, reliability, availability and define the corresponding aggregation functions for each of them. The predefined goal of these approaches is to optimize at run-time the values of these functions. Each of these quality properties has a corresponded weight which is predefined or set by the end user.
Reputation-based dynamic maintenance of composed services
In [11] the authors target the problem of maintaining dynamic service compositions, when one of the component services fail or became defective. The goal of this approach is the correction of a service composition when a failure occurs. It is applied on Business Process Management layer. The composition is designed as a BPEL process. The criterion for selecting at run-time the best possible service is the service reputation and this decision is made for each invocation of the component services. This is a proactive adaptation approach because we provide a priori the reputation information services. The reputation of each service depends on the success of the service invocation.

P2P architecture for self-healing processes
In [24] the authors address the problem of self-healing execution of BPEL processes. The goal of this approach is to correct service compositions, by looking-up and replacing the elementary services that fail during the execution of the process. It is applied on Business Process Management layer. The authors propose to exploit a peer-to-peer Resource Management Framework in order to look-up services, instead of using a centralized repository. In addition, they introduce the idea of embedding the WDSL description into this framework to achieve more robust look-up and binding.

![Figure 2.4: Service replacement in case of communication failures relying on a robust peer-to-peer service discovery and selection mechanism for alternative services.](image)

Generation of adapters for service integration
Many works have been proposed to address the problem of mismatches between the provided and the required functionalities among services developed by different parties. The goal of [12] and [9] is the customization of Service-based Applications in order to solve these mismatches and they are applied on Business Process layer. A
common way to target the predefined problem is to automatically create mediators that are responsible for smooth service interaction. Here, the authors propose two behavior-based adaptation approaches, where the generation of adapters, requires that the participating service descriptions are equipped with the interaction protocol the service implements. [12] aims at automated generation of an adapter that guarantees non-locking interaction of the services, while [9] proposes a taxonomy of behavioral mismatches and the corresponding behavioral patterns that may resolve the mismatch. This pattern is triggered when a mismatch occurred and is proposed to the application integrator, which is responsible for the adaptation process to be applied.

Reconfiguration approach based on local knowledge

Systems that want to apply adaptable reconfiguration have to be context-ware, i.e. they should be aware of the changes in their environment, in order to choose the best reaction to meet these changes. The authors in [55] propose a way that applies on System Infrastructure layer, to describe services and their current status, based on semantic properties. These properties can be used to change the system’s configuration, according to the changes in the context. Local knowledge is defined as knowledge about the immediate structure of a complex system, the components of such a system and the dependencies among the components belonging to the same layer in the architecture. Thus, the properties of a complex system constituted of the local requirements and the properties provided by the web services are used when reconfiguration is needed.

Reconfiguration approach based on Super Distributed Objects (SDO)

[59] is similar to the previous work and its goal is to provide an approach for run-time reconfiguration of services. It is applied on Business Process layer. The importance of this work is that it supports the continuous provision of services by combining devices to create a flexible context-aware service system. This system integrates devices that are embedded or distributed in the user’s environment and are networked with each other. The system for providing context-aware services locates and selects devices...
appropriate to the user’s context. When the context changes while the service is being provided, devices for that service can be partially reconfigured without resetting all the parameters. Resources (devices or software components) have been modeled as objects called SDO (Super Distributed Objects).

**MAIS Project (Multichannel Adaptive Information Systems)**

[42] and [14] focus on context modeling and automatic generation of web services. Their goal is to provide a framework to define and manage the context in a general environment characterized by adaptivity and multichannel access. They are applied on Service Infrastructure layer. Modern technologies enable users accessing services using multiple channels. In the service design phase, this poses additional requirements for high software adaptivity along different technical requirements and different user expectations. During execution, services are usually dynamically selected; this service selection phase requires the identification of the most suitable service along the context that characterizes the users in the time instant in which they send the service request. The MAIS architecture aims at providing automatically and efficiently services with the appropriate features by choosing among many provider offerings. Figure 2.6 shows the MAIS architecture and its components.

Figure 2.6: MAIS architecture with context management
Adaptive Service Composition in Flexible Processes

In [4] the authors propose an implicit approach for adaptive service composition in flexible processes. This approach is applied on Business Process Management layer. The goal is to select the best set of services available at runtime, taking into consideration process constraints, but also end-user preferences and the execution context. The authors introduce a new modeling approach to the Web service selection problem that is particularly effective for large processes and when QoS constraints are severe. In the model, the Web service selection problem is formalized as a mixed integer linear programming problem, loops peeling is adopted in the optimization, and constraints posed by stateful Web services are considered. Moreover, negotiation techniques are exploited to identify a feasible solution of the problem, if one does not exist. Figure 2.7 shows the basic architecture of this approach and how an optimum plan is extracted.

![Diagram](image)

**Figure 2.7: Extraction of optimum plan and WS ranking**

Adaptation in Semantic Web Services

[57] focuses on Web service’s protocol mediation by providing a framework that allows the interaction between two services relying on different protocols. It enables the substitution of one service provider with another even though they use different interaction protocols. This protocol mediation allows for the automatic adaptation of the service requester behavior meeting the constraints of the provider’s interface. This is done by abstracting from the existing interface descriptions before services interact and instantiating the right actions when contracting a service. This approach is applied on run-time binding and especially during the Service Composition layer. Figure 2.8 shows the proposed conceptual model of web services' protocol mediation.
2.5.2 Goal-based Adaptation Approaches

Goal-based approaches can be seen as an extension of implicit approaches. Their main characteristic is that they define a goal that the application should achieve. This goal is predefined and the adaptation specification identifies only some requirements and constraints that should guide the adaptation process. For example, in dynamic composition approaches this goal can be a set of effects or post-conditions and the set of the constituent services is derived at run-time. The requirements usually have the form of SLAs, which are official contracts between the service requester and provider that define the level of functional and non-functional characteristics of the provided services.

QoS adaptation in service grids

In [1] the authors propose a method for QoS adaptation in service-oriented grids. Its main goal is to optimize at run-time the use of resources and the layer it is applied on, is the Service Infrastructure layer. The proposed adaptation scheme enables the dynamic adjustment of behavior of an application based on changes in the predefined SLA. There are proposed three levels of QoS provision: "guaranteed" for which the provider commits to provide exact values of QoS characteristics, "controlled load" for which a range of QoS values should be delivered to the consumer, and "best effort" for which the consumer does not put any constraints on the QoS properties. During re-negotiation of SLAs more resources can be given to "best effort" services, when resources are not used by other classes of services. Thus, the proposed QoS adaptation scheme is used to compensate for QoS degradation and optimize resource utilization, by increasing the number of requests managed over a particular time.
Adaptation in Web service composition and execution

In [17] the authors propose a Web Service Composition and Execution system that adapts to failure of component services or changes in their QoS offerings. Its goal is to support run-time correction through re-binding or re-composition of services in highly dynamic environments and it performs on Service Composition Layer. The main problem of adaptive service composition which is addressed states as follows: "Given the specifications of a new service, create and execute a workflow that satisfies the functional and non-functional requirements of the service, while being able to continually adapt to dynamic changes in the environment". The proposed A-WSCE system separates the functional and non-functional requirements of a new service, and enables different environmental changes to be absorbed at different stages of composition and execution. Figure 2.9 shows the A-WSCE architecture.

![Figure 2.9: A–WSCE: System Architecture](image)

PAWS: A Framework for executing Adaptive Web-Service Processes

In [3] the authors introduce PAWS (Processes with Adaptive Web Services), a framework for flexible and adaptive execution of managed service-based processes, which is applied on Business Process Management layer. This framework coherently supports both process design and execution. PAWS has two primary goals. First to be self-optimizing, i.e. PAWS should select the best available services for executing the process and define the most appropriate quality-of-service (QoS) levels for delivering them and second, PAWS should guarantee service provisioning, even in case of failures, through recovery actions and self-adaptation if the context changes. To meet these goals, PAWS provides methods and a toolset to support design-time specification of all information required for automatic runtime adaptation of user preferences and context. This work focuses on how PAWS selects and adapts candidate services for a composed process. In addition, its annotations for exploiting
runtime flexibility are described. Flexibility in PAWS is provided in terms of optimization, mediation, and self-healing functionalities. Figure 2.10 shows the main concept of PAWS framework.

![Diagram of PAWS framework](image)

**Figure 2.10: The Processes with Adaptive Web Services architecture**

The METEOR-S Approach for Configuring and Executing Dynamic Web Processes

In [56] the authors propose an approach, whose main goal is to support run-time correction through rebinding services in highly dynamic environments. It is applied on the Service Infrastructure layer. It uses Semantic Web service discovery, integer linear programming and logic based constraint satisfaction to configure the process, based on quantitative and non-quantitative process constraints. Semantic representation of Web services and process constraints are used to achieve dynamic configuration. The semantic specification of the semantic Web Services is expressed in OWL that describes the domain specific knowledge, the functional and non-functional characteristics of the relevant aspects of the process model. At deployment-time, as well as at run-time when the service failures are detected and the reconfiguration is required, the proposed platform performs service discovery based on the requested service templates and their semantic descriptions, performs quantitative analysis over non-functional properties using linear programming solver, and qualitative analysis over functional properties using a dedicated SWRL reasoner. Figure 2.11 represents an overview of METEOR-S architecture.

37
A Service discovery framework
In [51] the authors propose a framework that permits the dynamic identification and replacement of the services that failed their contracts. The goal of this approach is to correct the functional and non-functional requirements violations. The framework can be applied on Service Infrastructure and on Service Composition layer. To be more specific, this work propose a method that monitors composite services at run-time, and when violations of requirements are detected, the system automatically extracts additional constraints, that are used to dynamically perform service discovery, which will substitute the malfunctioning services. The monitor is used to detect both behavioral and QoS requirements violations.

Petri-net reconfiguration of Web Services
In [58] the authors propose a model that allows run-time reconfiguration of service-based applications that operate in highly dynamic environments. This model can be applied on Service Infrastructure layer. This approach builds a Web service configuration net based on Petri nets in order to exhibit web service configuration in a formal way. Then, an optimal algorithm is presented to help choose the best configuration with the optimal quality of services (QoS) to meet users’ non-functional requirements.

QoS-based reconfiguration
In [25] the authors discuss the QoS-based analysis techniques and tools to facilitate composition decision making. The goal of this approach is to enable run-time reconfiguration on the basis of the provided QoS and its functional layer is the Service Infrastructure layer. Particularly, it focuses on developing methods to provide the highest QoS, which is achieved by using an algorithm. The authors use an extension
of WSDL to express properties about the QoS behavior of a system. The focus is on obtaining an adaptation of the system configuration through adaptation of the observed QoS behavior. The information gathered about the QoS behavior provided is used to compare the different candidate configurations, using genetic algorithm to find the best one.

**Speeding up adaptation**

In [27] the authors propose an approach to speed up adaptation of web service compositions using expiration types. This work is applied on Business Process Management layer. Web processes must often operate in volatile environments where the quality of service parameters of the participating service providers changes during the life time of the process. In order to remain optimal, the Web process must adapt to these changes. The authors provide a mechanism called the value of changed information which measures the impact of expected changes from the predefined SLAs to reduce the computational overhead of adaptation.

**Negotiation of SLAs**

In [21] the authors propose an architecture and a search-based approach for negotiation of SLAs, which lies on Service Composition layer. Negotiation can either be performed directly by the interested stakeholders or it can be automatic. In this second case, human beings are replaced by automated negotiators that try to achieve the objective that has been suggested to them. Automated negotiation is particularly important when the consumer of a service is a software system that has to negotiate on the fly (part) of the SLA with the service. This approach presents a multi agent system-like framework that is used to perform SLA negotiation. The architecture of this framework, is composed of a Marketplace and various Negotiation Multi Agent Systems. Each negotiation participant is associated to a multi agent system and can represent a human being or an automatic system. In the second case the system encapsulates a decision model that features various search-based algorithms.

**Towards Self-Adaptive Service-Oriented Architectures**

In [20] the authors propose an approach towards self-adaptive service-oriented architectures that lies on Service Infrastructure and Service Composition layer. This approach enables clients to automatically adapt their behavior to alternative web services that provide compatible functionality through different interaction protocols. It uses an infrastructure that traces the successful interactions of the web services, automatically synthesize models that approximate the interaction protocols, and steer client-side adaptations at runtime. The main limitations of this approach are that the test suite has to be manually provided by a developer that analyzes the candidate
service and that usually many tests runs are required for a proxy to provide a suitable transformation. Figure 2.12 represents an architecture of the proposed approach.

![Image](image_url)

**Figure 2.12: The Enhanced Service-Oriented Architecture**

**Automatic Learning of Repair Strategies for Web Services**

In [41] the authors introduce a methodology and a tool for learning the repair strategies of Web Services to automatically select repair actions. This approach is applied on Service Infrastructure and Service Composition layer. This methodology is able to incrementally learn its knowledge of repairs, as faults are repaired. Thus, it is at runtime possible to achieve adaptability according to the current fault features and to the history of the previously performed repair actions. This learning technique and the strategy selection are based on a Bayesian classification of faults in permanent, intermittent and transient, followed by a comparative analysis between current fault features and previously classified faults features which suggests which repair strategy has to be applied. Therefore, this methodology includes the ability to learn autonomously both model parameters, which are useful to determine the fault type, and repair strategies which are successful and proper for a particular fault.

**Context-aware Composition of E-Services**

In [5] the authors propose an approach for context-aware composition of Web services based on an abstract description and context that is applied on Business Process Management layer. Web services are described in terms of functionality and quality of service. The context describes the channels that can be used to access Web services. This approach propose adaptation rules as the means to allow the composition and dynamically select Web service channels according to the constraints posed by available architectures and application-level requirements. Composition and adaptation rules are exemplified on a simple case for emergency management.
2.5.3 Action-based Approaches

In contrast to goal-based, action-based approaches focuses on actions to be performed upon certain events, as for example when the context changes or when a service fails. They are also more efficient, because the adaptation strategies are predefined, and as a result no addition reasoning or decision-making is required at run-time. On the other hand, action-based approaches are less flexible, since adaptation specification defines only actions to be performed only to a restricted set events and scenarios, and not to all possible events. As a sequence, such type of approaches are used for self-healing systems, where the events corresponds to a fixed set of failures occurs during the execution of the service-based application, and the adaptation actions corresponds to system recovery actions.

Towards Self-healing Service Compositions

In [6],[7] the authors propose approaches to recover from various faults in dynamic environments, that are applied both on Service Infrastructure layer and on Service Composition layer. First, [6] identifies and classifies the main faults of service-oriented systems and sketches some solutions to make compositions self-healing. The proposal reorganizes a service composition in such a way that it can be suitably monitored and reorganized according to changing contexts. In [7] presents an approach towards self-healing compositions called Dynamo. It is an assertion-based solution that provides special purpose languages (WSCoL and WSReL) for defining monitoring and recovery activities. The main ingredients of these languages are the atomic recovery actions and the recovery specifications that declaratively join atomic actions in two complex steps and alternatives. The set of actions include actions at service level (e.g. retry, rebind, ignore), at process level (e.g. substitute, halt, call), and management actions (e.g. notify, log). Figure 2.13 represents the proposed self-healing framework.
In [25] the authors propose a way to correct and customize the application instance without changing the application model that can be applied both on Service Infrastructure layer and on Service Composition layer. To achieve this, they introduce a policy-driven middleware, MASC, where the policies, defined in the form of ECA-rules, correspond to the customization actions that should be performed in order to react to a certain malfunction. The policies are presented in WS-Policy4MASC language which extends WS-Policy by defining new types of policy assertions. This language allows for defining not only adaptation but also monitoring rules and directives. Two types of adaptation actions are proposed: process-level actions used to alter business logic of a particular instance (e.g. add, remove, replace) and message-level actions used to deal with low-level faults, such as invocation failure, SLA violation (e.g. invocation retry, concurrent invocation of several similar services).

**SH-BPEL - A Self-Healing plug-in for Ws-BPEL engines**

In [36] the authors propose an approach that provides failure recovery for BPEL process that is applied both on Service Infrastructure layer and on Service Composition layer. The aim of this paper is to present a self-healing plug-in for a Ws-BPEL engine that enhances the ability of a standard engine to provide process-based
recovery actions. The proposed solution defines SH-BPEL, which integrates and supports the necessary recovery facilities. The original process specification is pre-processed and extended with the additional instructions and control points, which allow for performing the above actions. These additional recovery actions constitute the management API and can be invoked through a special process management interface that is made available at deployment-time. Figure 2.14 shows the architecture of SH-BPEL engine.

Figure 2.14: The architecture of SH-BPEL Engine

Policy Based Adaptive Services for Mobile Commerce

In [49] the authors address a different adaptation problem related to the changes in the context of mobile service-based applications. This approach is applied on Service Infrastructure layer. It is based on the reuse and adaptation of existing and matured standards, APIs and middleware for representing context information, usage of policies for reasoning and for the communication between the involved parties. The context (i.e. the properties of the device, possible services and their characteristics, user preferences and settings), is presented using Resource Description Framework, and then reflected in definition of policies using the concepts of template and facts. The policies define the adaptation actions that should be triggered when the system reach a certain context situation. Figure 2.15 shows the core elements of the proposed adaptation architecture.

Figure 2.15: The core elements of the adaptation architecture
Design-time modeling and model transformation

In [30] the authors propose an approach for adaptive reconfiguration of systems, that is applied on Service Composition layer. It is based on building a system model at design time, having available the initial configuration of its services. The system reconfiguration is based on rules, which are presented in left-hand-side and right-hand-side graphs. This type of rules can be easily modified by the user. When the system model meets the left-hand-side of a rule, then it is substituted by the corresponding right-hand-side. The policies described at design-time are deployed on a structure made of services which enables the run-time adaptation of the system.

SCENE: A Service Composition Execution Environment

In [19] the authors propose SCENE, a service composition language, and a runtime execution environment for service composition. SCENE offers a language that extends the standard BPEL language with rules used to guide the execution of binding and re-binding self-reconfiguration operations. The composition is represented by a runtime platform composed by a common BPEL engine executing the composition, Drools - an open source rule engine - which is responsible for running the rules associated to the composition, WS-Binder that is in charge of executing dynamic binding and re-binding, and by a Negotiation component that can be used to automatically negotiate SLAs with component services when needed. Figure 2.16 shows the architecture of the SCENE platform.
An Approach to Adapt Service Requests to Actual Service Interfaces

In [16] the authors propose an approach, applied on Service Composition layer that extends the SCENE framework to support run-time resolution of differences between interfaces and protocols of invoked services. This work defines a classification of possible mismatches, a set of predefined adaptation functions to solve each mismatch, a language to combine these functions in adaptation scripts and a component, called adapter, integrated in SCENE proxies that can execute the scripts specified by a system integrator at design time. Service selection is usually performed assuming that all the implementations of an abstract service have the same interface or protocol. This assumption is not necessarily true in an open world setting, where services built by different organizations are made available. In this paper the authors address the problem of invoking services having an interface or protocol different from those originally expected by the service requester.

WS-DIAMOND: An Approach to Web Services - Monitoring and Diagnosis

In [54] the members of the WS-Diamond EU Project propose an execution environment and design tools to design self-healing service compositions. This environment lies on Service Infrastructure and Service Composition layer. This approach consists of a set of methodologies, design-time tools, and runtime tools to design and develop a platform for observing symptoms in complex composed applications, for diagnosis of the occurred faults, and for selection and execution of repair plans. Additionally, WS-DIAMOND includes methods and tools to assess the self-healability of the processes, including repairability evaluation, diagnostability, temporal conformance checkers. Figure 2.17 shows the WS-Diamond overall architecture.

Figure 2.17: WS-Diamond Overall Architecture
On dealing with semantically conflicting business process changes (SCBP)
In [47] the authors discuss how to correctly propagate changes occurring in a business process to the process instances. This approach lies on Business Process Management layer. It identifies and classifies semantic conflicts between the process type and instance caused by the changes in the process schema. In addition, this work describes formal dynamic methods for conflicts detection arising at change propagation time. More precisely, in order to enable the process management system to semantically detect conflicting changes the authors provided two methods: the first is based on execution equivalence of the respective process schemes, and the second, is based on a direct comparison of the applied changes.

A Constraint Satisfaction Approach to Non-functional Requirements in Adaptive Web Services
In [39] the authors present a constraint satisfaction based approach to model non-functional requirements in the context of adaptive Web Services, that lies on Service Infrastructure and Service Composition layer. Policy reconciliation between service providers and service requestors in such adaptive SOA systems is often a difficult requirement to satisfy owing to the generality of the policies not only those associated with a single service provider but across multiple services. Addressing this dynamic adaptation requirement for policy based non-functional requirements, the authors present a constraint satisfaction based framework to represent, model, and work with policy based non-functional requirements in Adaptive Web services.

Evaluation of correctness criteria for dynamic workflow changes
In [46] the authors propose an approach that lies on Business Process Management layer that introduces criteria that should be considered to perform correct adaptation on running workflows. These criteria are: completeness, correctness and change realization. Rules and information needed for satisfying these criteria are also identified. Furthermore, the authors demonstrate the strengths of different approaches and provide additional solutions to overcome current limitations.

LPV - Model Identification for power management of Web Services
In [53] the authors propose an approach that lies on Service Composition layer that introduces LPV models for the performance control of Web Services. In SOA environments, service providers need to comply with the service level objectives stipulated in contracts with their customers while minimizing the operating costs of the physical infrastructure, mainly related to energy costs. Recently, this problem has begun to be formalized in terms of a constrained control problem, where the SLAs are translated into set-points for the response time of the servers, possibly different according to the customer class, and tracking performance must be traded-off with
energy saving objectives. As the behavior of the server response time is highly time varying and the workload conditions substantially change within the same business day, the LPV framework seems very promising for modeling such systems, since it better fits situations in which variability is high, with respect to the traditional queue nets model.

**Automated Generation of BPEL adapters**

In [12] the authors propose a methodology for the automated generation of adapters capable of solving behavioral mismatches between BPEL processes. This approach is applied on Business Process Management layer. The adaptation process, given two communicating BPEL processes whose interaction may lock, builds (if possible) a BPEL process that allows the two processes to successfully interoperate. A key ingredient of the adaptation methodology is the transformation of BPEL processes into YAWL workflows. This representation is used as input for a match making heuristics that tries to build an adapter for received requests. Once the adapter is generated, the interaction between the invoker and the service is analyzed, using lock analysis techniques, to discover if some interaction scenarios cannot be resolved.

**An Aspect-Oriented Framework for Service Adaptation**

In [49] the authors propose an aspect-oriented framework for service adaptation. This framework is applied on Service Infrastructure and on Business Process Management layer. This framework consists of i) a taxonomy of the different possible types of mismatch between external specification and service implementation, ii) a repository of aspect-based templates to automate the task of handling mismatches, and iii) a tool to support template instantiation and their execution together with the service implementation. Furthermore, join points have been defined for BPEL and solution patterns for some possible mismatches are defined.

**Semi-Automated Adaptation of Service Interactions**

In [38] the authors propose a tool to assist a system integrator in developing adapters for web service integration. This tool is applied on Business Process Management layer. This approach (i) identifies mismatches between service interfaces, which leads to finding mismatches of type of signature, merge/split, and extra/missing messages; (ii) identifies all ordering mismatches between service protocols and generate a tree, called mismatch tree, for mismatches that require developers' input for their resolution. In addition, semi-automated support is provided in analyzing the mismatch tree to help in resolving such mismatches. This tool works under the assumption that the requestor is also a service. The tool takes as input an interface and behavioral description of the requestor and of the service and provides hints to the human system integrator about possible mismatches it finds matching the input representations.
When a mismatch is found the developer has to specify a function to solve it. Figure 2.18 shows the architecture of the adapter development tool.

![Figure 2.18: The architecture of the adapter development tool](image)

**Run-Time Adaptation of Non-functional Properties of Composite Web Services Using Aspect-Oriented Programming**

In [37] the authors are focusing on run-time adaptation of non-functional features of a composite Web service by modifying the non-functional features of its component Web services. This approach is applied both on Service Infrastructure and on Service Composition layer. Aspect-oriented programming (AOP) technology is used for specifying and relating non-functional properties of the Web services as aspects at both levels of component and composite. This is done via a specification language for representing non-functional properties, and a formally specifiable relation function between the aspects of the component Web services and those of the composite Web service. From the end users’ viewpoint, such upfront aspect-oriented modeling of non-functional properties enables on-demand composite Web service adaptation with minimal disruption in quality of service. Figure 2.19 shows the architecture of the proposed approach.
2.5.4 Explicit Variability approaches

What makes explicit variability approaches differ from the previous three types of approaches is that they allow one to precisely define a particular time in the execution, where the adaptation changes should take place, and to represent all the relevant variants of behavior, applicable in cases of changes needed. The role of the monitoring activity in explicit variability approaches is different from the goal-based and action-based ones. Since the variation point is defined explicitly, there is no need to monitor specific conditions or assertions. Instead, the monitoring activity is used to obtain some relevant context information that may be required to choose one alternative or another. As well as in action-based approaches, the explicit variability approaches are rather efficient, since the only overhead refers to the evaluation of the selection criteria for a particular alternative, which is already defined.

The challenges of service evolution

In [40] the author presents a theoretical approach for dealing with the service evolution. It classifies possible changes and proposes approaches for dealing with them. This work refers to all the functional layers of a service-based application. This paper introduces two types of service changes depending on the nature of their effects: shallow changes - where changes are confined to services or the clients - and deep changes - where cascading effects and side-effects occur. Furthermore, it categorizes the types of changes that occur in services based on functional criteria: structural, that
focus on service types, messages, interfaces and operations, *business protocol*, that affect the structure and ordering of the messages that a service and its client exchange to achieve a business goal, *policy induced*, requiring changes to policy assertions and constraints and *operational behavior*, that concentrate on the effects and side effects of changing service operations. This work introduces some key approaches for each type of changes that can be used as the basis for future research in service evolution, and sketches the characteristics of the change-oriented life cycle. Figure 2.20 shows the change-oriented life cycle.

![Change-oriented service life cycle](image)

**Figure 2.20: Change-oriented service life cycle**

**DySOA**

In [50] the authors propose an approach that allows for managing and guaranteeing the QoS parameters of the application in dynamic environments. It is applied on Service Infrastructure layer. The proposed DySOA (Dynamic Service-Oriented Architecture) is an architecture that extends service-centric applications to make them self-adaptive. DySOA allows developers to explicitly model elements that deal with QoS evaluation and variable composition configurations. Having the DySOA elements explicit enables separation of concerns, making them adaptable at runtime and reusable in next versions. This approach is based on explicit modeling of different variants corresponding to the variation points. Variation point defines a particular element of the specification where the selection may apply, whether variant defines the behavioral or functional alternative to be applied in the variation point. The QoS metrics of the application are continuously monitored and evaluated. When a violation is detected, the reconfiguration unit is responsible to select the appropriate variant at the corresponding variation point, in order to ensure QoS optimality.
Managing Process Variants in the Process Life Cycle - PROVOP

In [26] the authors propose PROVOP (PROcess Variants by OPtions) approach for managing large collections of process variants. This approach lies on Business Process Management layer. Process variants can be configured out of a basic process following an operational approach; i.e., a specific variant is derived from the basic process by applying a set of well-defined change operations to it. PROVOP provides full process life cycle support and allows for flexible process configuration resulting in a maintainable collection of process variants. The basic idea is to keep variants in the one model. For this purpose, the basic process is defined, and its variants are represented by the set of change operations that allow the migration of the basic case model into a specific variant model. The transformation operations are defined as action templates, where actions are insert, delete or move. To sum up, the functionality of PROVOP approach is the following: First, the relevant variants are selected and filtered according to contextual information. Second, the option constraints are evaluated in order to further restrict the possible options. Finally, the selected are applied and executed in process engine.

Workflow evolution

In [15] the authors propose an approach that allows for addressing process variability in dynamic process context. This approach lies on Business Process Management layer. Apart from modeling process variants, this work defines notations and formalisms for describing and implementing transformation actions. These actions define the instructions for changing both control flow and data flow model. The proposed actions are add/remove variable, insert task, add/remove successor of a task, change connection type, or change transition condition. In these regards, the proposed approach may be considered hybrid, as it includes both the explicit variability modeling and the definition of adaptation actions to be applied when the change is required.

Semantic constraints for adaptation

In [45],[34], the authors propose two approaches towards dynamic transformation of enterprise models. These approaches lie on Business Process Management layer. Both approaches propose a formal framework for representing both the underlying dynamic models and changes in them. Differently to previous approaches, these works come up with semantic constraints on the changes in these models, and with notion of correctness and consistency of the changes with respect to these constraints. In [45] the authors provide a way to perform a semi-automated adaptation of access rules in response to changes made on an organizational model, while in [34] a framework for automated verification of process changes with respect to the constraints is proposed.
2.5.5 Comparison of Adaptation approaches

The works discussed above represent different approaches towards Web service adaptation. In table 2.3 we make a comparison of all these approaches according to a set of criteria. These criteria are the following:

- **Objective** defines the adaptation goal (i.e. correction, customization, optimization etc)
- **Type of change**, which defines the type of change that triggers the adaptation process (i.e. which characteristics of the SBA change)
- **Functional layer** defines the location of the change (Service Infrastructure layer, Service Composition layer, Business Process Management layer)
- **Internality** defines whether the change is internal (e.g. faults) or external (e.g. context, requirements)
- **Change representation**, defines whether the change is represented implicitly or explicitly and
- **Adaptation representation**, which defines the way the adaptation is modeled according to the four categories already exemplified (implicit, goal-based, action-based, explicit variability)

There are many other approaches focusing on this topic, but the works presented here are a very representative sample that summarizes the state of the art on Web services’ adaptation.

As this table shows, adaptation has gained a significant attention in complementary way of the monitoring process to overcome the mismatches between services for service integration, repairing the failures, reconfiguring the systems by leveraging existing approaches in software engineering. We also notice that, the notion of adaptability in SBA till date have concentrated on capturing functional requirements, few thought and projects have been done on the non functional properties. Furthermore, there are no existing approaches addressing the adaptation cross all the functional layers of the service based systems with respect to a variety of information discussed earlier. The proposed approaches address either only particular functional layer, or a particular problem e.g. identification mismatch. The table also shows that in the adaptation research, the trend is toward all-in-one solutions by integrating a maximum of requirements in order to have a complete framework. Moreover, some other important aspects such as contextual information is thought-worthy. In fact, insignificant investigations have tackled the context when adapting the system, while its application gained more and more the service based systems.
## Table 2.3: Comparison of state-of-the-art approaches on Web Service Adaptation

<table>
<thead>
<tr>
<th>Approach</th>
<th>Objective</th>
<th>Type of Change</th>
<th>Functional layer</th>
<th>Internality</th>
<th>Change representation</th>
<th>Adaptation representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality-driven service composition</td>
<td>Composition optimization</td>
<td>QoS</td>
<td>SC</td>
<td>Internal</td>
<td>Implicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>Reputation-based dynamic maintenance of composed services</td>
<td>Correction</td>
<td>Faults</td>
<td>BPM</td>
<td>Internal</td>
<td>Implicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>P2P architecture for self-healing processes</td>
<td>Correction</td>
<td>Faults</td>
<td>BPM</td>
<td>Internal</td>
<td>Implicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>Generation of adapters for service integration</td>
<td>Customization</td>
<td>Mismatches</td>
<td>BPM</td>
<td>Internal</td>
<td>Implicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>Reconfiguration approach based on local knowledge</td>
<td>Reconfiguration</td>
<td>Context</td>
<td>SI</td>
<td>External</td>
<td>Explicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>Reconfiguration approach based on SDO</td>
<td>Reconfiguration</td>
<td>Context</td>
<td>BPM</td>
<td>External</td>
<td>Explicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>MAIS Project</td>
<td>Context modeling</td>
<td>Context</td>
<td>SI</td>
<td>External</td>
<td>Implicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>Adaptive Service Composition in Flexible Processes</td>
<td>Composition optimization</td>
<td>QoS</td>
<td>BPM</td>
<td>Internal</td>
<td>Implicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>Adaptation in Semantic Web Services</td>
<td>Customization</td>
<td>Run-time binding</td>
<td>BPM</td>
<td>Internal</td>
<td>Explicit</td>
<td>Implicit</td>
</tr>
<tr>
<td>QoS adaptation in service grids</td>
<td>Optimization</td>
<td>QoS</td>
<td>SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>Adaptation in Web service composition and execution</td>
<td>Correction, Optimization</td>
<td>QoS, service failures</td>
<td>SC</td>
<td>Internal</td>
<td>Explicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>METEOR-S</td>
<td>Correction, Optimization</td>
<td>QoS, service failures</td>
<td>SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>A Service discovery framework</td>
<td>Correction</td>
<td>Funct. and non func. requirements violations</td>
<td>SI, SC</td>
<td>External</td>
<td>Explicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>Approach</td>
<td>Objective</td>
<td>Type of Change</td>
<td>Functional layer</td>
<td>Internality</td>
<td>Change representation</td>
<td>Adaptation representation</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>----------------------------------</td>
<td>----------------</td>
<td>------------------</td>
<td>-------------</td>
<td>-----------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Petri-net reconfiguration of Web Services</td>
<td>Correction, Optimization</td>
<td>Faults, requirements</td>
<td>SI</td>
<td>Internal, External</td>
<td>Explicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>QoS-based reconfiguration</td>
<td>Reconfiguration, Optimization</td>
<td>Context</td>
<td>SI</td>
<td>External</td>
<td>Implicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>Speeding up adaptation</td>
<td>Optimization</td>
<td>QoS</td>
<td>BPM</td>
<td>Internal</td>
<td>Implicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>Negotiation of SLAs</td>
<td>Optimization</td>
<td>SLA</td>
<td>SC</td>
<td>External</td>
<td>Explicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>Towards Self-Adaptive Service-Oriented Architectures</td>
<td>Customization</td>
<td>Mismatches</td>
<td>SC,SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>Automatic Learning of Repair Strategies for Web Services</td>
<td>Learning</td>
<td>Faults</td>
<td>SC,SI</td>
<td>Internal</td>
<td>Implicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>Context-aware Composition of E-Services</td>
<td>Optimization</td>
<td>QoS, Context</td>
<td>BPM</td>
<td>Internal, External</td>
<td>Implicit</td>
<td>Goal-based</td>
</tr>
<tr>
<td>Towards Self-healing Service Compositions</td>
<td>Correction</td>
<td>Faults, Assertion violations</td>
<td>SC,SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>Policy-driven middleware for self-adaptation</td>
<td>Customization, Correction</td>
<td>Faults, specific application</td>
<td>SC,SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>SH-BPEL</td>
<td>Correction, Correction</td>
<td>Failures</td>
<td>SC,SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>Policy Based Adaptive Services for Mobile Commerce</td>
<td>Customization</td>
<td>Contextual properties</td>
<td>SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>Design-time modeling and model transformation</td>
<td>Reconfiguration, Correction</td>
<td>Behavior</td>
<td>SC</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>SCENE</td>
<td>Correction, Customization</td>
<td>Mismatches</td>
<td>SC</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>WS-DIAMOND</td>
<td>Correction</td>
<td>Faults</td>
<td>SC,SI</td>
<td>Internal</td>
<td>Implicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>On dealing with semantically conflicting business process changes (SCBP)</td>
<td>Correction</td>
<td>semantic conflicts</td>
<td>BPM</td>
<td>External</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>Approach</td>
<td>Objective</td>
<td>Type of Change</td>
<td>Functional layer</td>
<td>Internality</td>
<td>Change representation</td>
<td>Adaptation representation</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>------------------</td>
<td>-------------</td>
<td>-----------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>A Constraint Satisfaction Approach to Non-functional Requirements in Adaptive Web Services</td>
<td>Optimization</td>
<td>HCI</td>
<td>SC,SI</td>
<td>External</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>Evaluation of correctness criteria for dynamic workflow changes</td>
<td>Optimization</td>
<td>-</td>
<td>BPM</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>LPV - Model Identification for power management of Web Services</td>
<td>Customization</td>
<td>model</td>
<td>SC</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>Automated Generation of BPEL adapters</td>
<td>Correction</td>
<td>Mismatches</td>
<td>BPM</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>An Aspect-Oriented Framework for Service Adaptation</td>
<td>Correction</td>
<td>Mismatches</td>
<td>BPM./SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Action-based</td>
</tr>
<tr>
<td>Semi-Automated Adaptation of Service Interactions</td>
<td>Integration</td>
<td>Mismatches</td>
<td>BPM</td>
<td>-</td>
<td>-</td>
<td>Action-based</td>
</tr>
<tr>
<td>The challenges of service evolution</td>
<td>Evolution</td>
<td>Structural and behavioral</td>
<td>SC,SI,BPM</td>
<td>Internal, External</td>
<td>Explicit</td>
<td>Explicit Variability</td>
</tr>
<tr>
<td>DySOA</td>
<td>Customization, Correction</td>
<td>QoS</td>
<td>SI</td>
<td>Internal</td>
<td>Explicit</td>
<td>Explicit Variability</td>
</tr>
<tr>
<td>PROVOP</td>
<td>Customization</td>
<td>Process context</td>
<td>BPM</td>
<td>External</td>
<td>Explicit</td>
<td>Explicit Variability</td>
</tr>
<tr>
<td>Workflow evolution</td>
<td>Customization</td>
<td>Process context</td>
<td>BPM</td>
<td>External</td>
<td>Explicit</td>
<td>Explicit Variability</td>
</tr>
<tr>
<td>Semantic constraints for adaptation</td>
<td>Customization, Correction</td>
<td>Process / model</td>
<td>BPM</td>
<td>External</td>
<td>Explicit</td>
<td>Explicit Variability</td>
</tr>
</tbody>
</table>
Chapter 3

3 Self-* properties of Web Services

3.1 Introduction

Achieving various self-* properties has been a grand challenge of computer science and engineering since the building of the first computer. The latest reincarnation of this challenge is due to the fact that large, complex and dynamic information systems have suddenly become a key part of the infrastructure of modern societies. Accordingly, it has become very important to be able to build, manage, and exploit these systems in the most efficient way possible. In other words, these systems have to become self-*.

However, it is a complex problem to deal with this challenge because information systems are deployed over a very wide range of environments from wireless sensor networks to powerful supercomputing clusters, from home networks to the entire Internet. It is very likely that to meet this challenge, we need to draw ideas from many disciplines that have been dealing with the design or explanation of large and/or complex systems, such as the space shuttle, an operating system, the Internet, human intelligence, an ecosystem, evolution, living organisms, etc.

Self-* does not exist on its own, it is a collection of various properties that are present in autonomic computing systems to realize self-managing or self-governing functionalities. IBM [33] proposed the notion of autonomic computing as systems that are able to manage themselves based on the high-level objectives given by human administrators. The term autonomic comes from a biology metaphor: the autonomic nerve system governs many of the low-level yet vital functions of the human body without any conscious intervention. Similarly, autonomic systems are assumed to maintain and adjust their operations in face of changing components, workloads, demands, external conditions and failures, both accidental and malicious.
3.2 Types of self-* behavior

The Vision of Autonomic Computing [33] identified four self-* properties commonly referred as self-management: self-configuration, self-optimization, self-healing, self-protection. Self-configuration is related to installation, configuration and integration of large complex systems in an autonomic way. Self-optimization is an effort made by the autonomic system to find ways of improving its operation, be more efficient in performance and cost. Self-healing is an ability to detect, diagnose and repair localized problems caused by bugs or failures in software and hardware components. Self-protection is a defense against large-scale, correlated problems arising from malicious attacks or cascading failures. It also means anticipating and avoiding problems based on early symptoms.

There many other self-* categories as self-awareness, self-monitoring, self-adjusting, self-anticipating, self-adapting, self-critical, self-defining, self-destructing, self-diagnosis, self-healing, self-governing, self-organized, self-recovery, self-reflecting, self-simulation and self-stabilizing (Sterritt, et al., 2005) and potentially others. Hence, self-* or selfware refers to some of the above mentioned functionalities that are necessary to realize (some aspects of) autonomic behavior. Definitely, a ’self-* system’ or a ’self-* service’ does not mean realizing all of these properties, as it would be close to human intelligence. Figure 3.1 shows the Self-healing Composition Cycle for Web Services.

![Figure 3.1: Self-healing Composition Cycle for Web](image-url)
3.3 Levels of self-* behavior

Autonomic properties may be present at various levels from individual computing components, e.g. a single computer or simply a disk drive to large-scale federated computing systems. These levels require different techniques to ensure self-* behavior. Higher levels involve more flexibility and dynamism. Ultimately, autonomic behavior would be expressed as declarative high-level goal-oriented statements that are decomposed into lower level detailed requirements at elements of the autonomic system on the fly. Service-oriented concepts play a fundamental role for autonomic computing nevertheless, some augmentations are necessary.

3.4 Lifecycle of the self-* services

An autonomic element (e.g. a self-* service) has its own lifecycle that goes through design, test and verification; installation and configuration; monitoring and problem determination, recovery; upgrading; uninstallation and replacement. Each such stage comprises some special features. Design, test and verification of autonomic elements require managing the relationships between elements, needs and preferences along with capabilities; specification of high-level goals and mapping them to lower-level actions. Installation and configuration of autonomic elements involve service publication and discovery in directory services. Monitoring and problem determination are essential functionalities for autonomic systems as they may be continuously performing adaptation, self-optimization, reconfiguration and self-healing. An important aspect of monitoring is ensuring the quality of required and provided services. Upgrading of the services can be installing newer versions or creating entirely new service instances in an autonomic way. Obviously, autonomic services may be engaged in many simultaneous activities representing different stages of the lifecycle. Hence, management of the lifecycle is necessary in order to prioritize and schedule the activities so that elements can maintain a consistent view.

Similarly, the relationships between autonomic elements also have a lifecycle. Specification must include output services that a certain autonomic element can produce and input services it may require. An autonomic element must be able to locate input services whereas it must be locatable by other services. Negotiation encompasses several procedures that can be involved in obtaining a service once a provider was found. After the negotiation phase, provision allows access to the elements’ resources. Operation means the utilization of the negotiated and provided services, resources and finally, termination ends the agreement between parties that can reallocate their resources for other uses.
Chapter 4

4 Research challenges and Concluding Remarks

If Web services technology shall be applied to a broad spectrum of applications, it will have to be significantly improved with respect to some capabilities. In particular, in dynamic environments it must be possible to quickly implement and deploy new services and to support evolution that might occur in the environment. To meet these expectations many approaches towards Web service adaptation. Although it is a premature topic, there has been made significant progress towards effective adaptation. Nevertheless, it is not a full covered topic and many more research challenges are still to be proposed.

These challenges [21] dictated not only by the comparative novelty of the adaptation problem in the service area, but also by the high level of fragmentation of the current research activities.

First, the proposed approaches address only particular aspects of the SBA adaptation. They target only a specific kind of application changes, or a specific functional layer of SBA. The problems of how to identify and model cross-layer dependencies as well as how to design cross-cutting adaptation strategies remain open.

Second, the set of addressed changes and the adaptation solutions is rather restricted; the proposed approaches are not flexible enough to accommodate to a wide range of scenarios or use cases. Most of the approaches deal with QoS changes and the related adaptation strategies, such as service replaceability, or with a particular set of application faults and the recovery actions. Accordingly, the kinds of goals, as well as the adaptation actions are rather simple. While on the one hand this allows for simple and efficient adaptation implementations, on the other hand this prevents creation of comprehensive and multidimensional adaptation methodologies. This requires new languages and techniques that would enable more complex and expressive adaptation capabilities, with respect to those currently presented in the literature. In particular, there is a necessity to provide notations that interleave the goal-oriented and action-oriented specification that would give more freedom to the platform in identification
and definition of the necessary adaptation strategies in the application production mode.

In these regards, an important problem is how to integrate various constraints on different aspects of the application functionality with the adaptation technique. On the one hand, there is a need to express real-world domain knowledge and properties, while on the other hand still have to be manageable by the underlying adaptation realization. An important issue is also a possibility to incorporate different types of constraints (e.g., behavioral, QoS), and hence to come up with hybrid analyzers and solvers for those constraints within the adaptation framework.

Third, there is a need to target not only the adaptation types, where the goal is to modify the system in reaction to the changes that already happened, but to adapt the system before these changes take place, i.e., preventive adaptation. This, however, requires novel techniques and methods for the application diagnosis in order to foresee the possible changes and the potential consequences of them.

Finally, in the settings, where the role of hybrid and highly dynamic open systems continuously growth, the adaptation design should target the problem of modeling various aspects of the application context and its evolution. Currently, only few adaptation proposals deal with the context-aware methods and techniques, restricting the applicability of the existing approaches to the new requirements and needs.

As far as the adaptation on Web service composition is concerned future research for the integration of online testing and Service Composition modeling as well as integration of requirements engineering with Service Composition modeling is still to be conducted. It is also important to test the whole composition (integration test). Techniques need to be developed and integrated with adaptation techniques.

For the requirements engineering field, the most predominant problem is the mapping of the requirements to service descriptions, e.g. requirements or goal descriptions using OWL-S, WSMO or others. This mapping ensures that, for example, services identified as superior in the requirements engineering activity, are actually given preference during service discovery carried out by the middleware. In addition, process instances may be tailored to the so-called context-factors, e.g. processes may be adapted to certain users or a certain device. These context-aware SC adaptations require a tight integration of SC and requirements engineering research.

There are many research issues concerning adaptation that users will further benefit from the advances:

- The integration and verification of domain knowledge will help in the development of adaptive service management technology. It is necessary to design a framework for defining semantic constraints over services in such a
way that they can express real-world domain knowledge on the one hand and
are still manageable concerning the effort of adaptation.

- Providing correctness criteria to decide how to adapt web services.
- Identification of actual differences between the (interface and protocol)
specifications of a given pair of Web services, and the composition of adapter
template solutions for resolving mismatches between protocols to generate the
adapter code.
- Another interesting issue is to investigate a framework for automatic
adaptation that accommodates hybrid constraints. In this case, a possible
solution is to study a way to combine solvers. This will provide more
flexibility and let users concentrate on requirements rather than on
implementation.

To sum up, there are still many research challenges on Web service adaptation.
Especially, proactive adaptation is a facet of this problem that is very promising and
could make Web services more effective and flexible. As we have already discussed
flexibility is a very important aspect for services in the dynamic and ever changing
environment they operate. Consequently, it would be ideal if we can apply service
adaptation before possible future changes occur.

For our research plan we intend to explore possible cross-layer monitoring and
adaptation mechanisms that can express monitored events and adaptation strategies in
a holistic and aligned manner. These mechanisms must identify exactly which is the
source of the problem and then trigger apt adaptation strategies according to the
adaptation needs of each layer.

Towards this direction we will try to identify and model the cross-layer dependencies.
Policy refinement solutions proposed in can be very helpful. As the complexity of
web services continues to increase the complexity of their management systems,
including those for QoS management, also increase. Within Policy Based
Management systems (PBMS) the desired behaviour of the managed system is
specified in a policy. Policies can be specified at many different levels of abstraction
from high-level business goals to policies for individual resources. It would be ideal
for an organization to derive their low level policies from their overall business goals
(high level policies). In the context of SBAs, policy refinement is capable of
automatically mapping high level goals of a composite service down to low level
policies.

Finally, in this research framework we intend to explore the work of Hielscher et al
[28] as a possible proactive adaptation solution in this overall cross-layer monitoring
and adaptation framework. Response time is very important in novel SBAs. So,
proactive adaptation mechanisms that would anticipate the possible changes and the
potential consequences of them would be ideal in a business environment.

61
5 Bibliography


[30]. Illner S. [et al.], *Automated runtime management of embedded service systems based on design-time modeling and model transformation*, In proceedings of Industrial Informatics, 2005.


