Grant Agreement N° 215483

Title:	*Results of the Second Validation*
Authors:	*UniDue, POLIMI, CITY, FBK*
Editors:	*Eric Schmieders (UniDue), Andreas Metzger (UniDue)*
Reviewers:	*Michael Parkin (Tilburg)*
	Ita Richardson (Lero-UL)
Identifier:	*CD-IA-3.2.4*
Type:	
Version:	*1*
Date:	*16 March 2011*
Status:	*Final*
Class:	*Internal*

Management Summary

This document is the compilation of the set of papers used to produce the 'paper-based' deliverable CD-IA-3.2.4, which reports the validation of the integration of the IRF building blocks, i.e., the results of task T-IA-3.2.1.

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

Members of the S-Cube consortium:

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

Published S-Cube documents

All public S-Cube deliverables are available from the S-Cube Web Portal at the following URL:

http://www.s-cube-network.eu/results/deliverables/

Contents

Towards Proactive Adaptation: A Journey along the S-Cube Service Life-Cycle

Andreas Metzger*, Eric Schmieders*, Cinzia Cappiello†, Elisabetta Di Nitto†,
Raman Kazhamiakin‡, Barbara Pernici†, Marco Pistore‡
*Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen, 45127 Essen, Germany
Email: {andreas.metzger, eric.schmieders}@paluno.uni-due.de
‡FBK-Irst
Via Sommarive 18, 38050, Trento, Italy
Email: {raman, pistore}@fbk.eu
†Politecnico di Milano, DEI
Piazza Leonardo da Vinci, 32, 20133 Milano, Italy
Email: {dinitto, cappiell, pernici}@elet.polimi.it

***Abstract*—Service-oriented applications are deployed in highly dynamic and distributed settings. Therefore, such applications are often equipped with adaptation capabilities to react to critical issues during their operation, such as failures or unexpected changes of third party services or to context changes. In this paper, we discuss shortcomings of current solutions for adaptive service-oriented applications. To address those shortcomings, we introduce techniques that can be utilized to build and evolve proactive applications. Those techniques have been developed in S-Cube, the European network of Excellence on Software Services and Systems. Proactive adaptation capabilities are considered particularly promising, as they can prevent costly compensation and repair activities. Using those techniques in an integrated way is described along the phases of the service life-cycle. We use a running example to illustrate the shortcomings of current solutions for self-adaptation and to demonstrate the benefits of the S-Cube techniques.**

I. Introduction

Service-orientation is increasingly adopted as a paradigm for building highly dynamic, distributed and adaptive software systems, called service-oriented (or service-based) systems. This paradigm implies a fundamental change to how software is developed, deployed, and maintained [1]: A service-based system cannot be specified and realized completely in advance (i.e, during design-time) due to the incomplete knowledge about the interacting parties (e.g., third party service providers) as well as the system's context and communication infrastructure [2]. Thus, compared to traditional software engineering, much more decisions need to be taken during the operation of the service-oriented system (i.e., after it has been deployed). For instance, those systems will need to react to failures of their constituent services (e.g., if a service provider fails to adhere to its contract) to ensure that they maintain their expected functionality and quality.

In such a dynamic setting, evolution and adaptation methods and tools become key to enable those systems to respond to changing conditions. In accordance with the terminology defined by the S-Cube Network of Excellence [3], this paper differentiates between evolution and adaptation as follows: Evolution is considered as the modification of the system's requirements, specification, models, etc. during design time (also known as maintenance). In contrast, adaptation is considered as the modification of a specific instance of a service-based system during operation. In the current paper we focus on adaptation needed due to some malfunctioning of the system. While the general adaptation due to context changes could also be supported by the proposed techniques, this is not discussed in the present paper.

A. Problem Statement and Related Work

Adaptive systems automatically and dynamically adapt to changing conditions. The aim of adaptation (aka. "self-adaptation") is to reduce the need of human intervention as far as possible. While the behavior of a non-adaptive system is only controlled by user input, adaptive systems consider additional information about the application and its context (e.g., failures of constituent services or different network connectivity). Thus, in order to realize self-adaptive behavior, methods and tools that realize control loops are established that collect details from the application and its context (e.g., by exploiting monitoring mechanisms) and decide and act accordingly [4].

So far, the major work on adaptation has been centered around reactive adaptation capabilities based on monitoring [5]. This means that adaptation is performed *after* a deviation or critical change has occurred. Such a reactive adaptation based on monitoring, however, has at least the following two important shortcomings (cf. [6], [7] and [8]).

- It can take time before problems in a service-based system lead to monitoring events that ultimately trigger the required adaptation. One key trigger for an adaptation should be the case when the service-based system deviates from its requirements (such as expected response time for example). If only those requirements are monitored (e.g., see [9]), the monitoring events might arrive so late that an adaptation of the Service Based Application

(SBA) is not possible anymore. For instance, the system could have already terminated in an inconsistent state, or the system has already taken more time than required by the expected response time.
- Reactive adaptation can become very costly, especially when compensation or rollback actions need to be performed. As an example, when using stateful (aka. conversational) services [10], the state of the failed service might need to be transferred to an alternative service.

Of course, one can monitor the individual services of an SBA and trigger an adaptation as soon as the service has failed, i.e., violated its contract [11]. However, when using those techniques it remains unclear whether the failure of this service could lead to a violation of the SBA's requirements. This means that there may be situations in which the SBA is adapted although it would not have been necessary, because the requirements might still have been met. Consider the following simple example: Although a service might have shown a slower response time as (contractually) expected, prior service invocations (along the workflow) might have been fast enough to compensate the slower response of that service.

Such unnecessary (or "false positive") adaptations have the following shortcomings [6]:

- Unnecessary adaptations can lead to additional costs and effort that could be avoided. For instance, additional activities such as Service Level Agreement (SLA) negotiation for the alternative services might have to be performed, or the adaptation can lead to a more costly operation of the SBA, e.g., if a seemingly unreliable but cheap service is replaced by a more costly one.
- Unnecessary adaptations could be faulty (e.g., if the new service has bugs), consequently leading to severe problems.

In summary, one key problem that needs to be solved to enable proactive adaptation is to determine whether the service-based application, during its future operation, might deviate from its requirements.

B. Contribution of Paper

This paper describes techniques developed in the S-Cube[1] network of excellence to determine deviations from requirements based on monitored failures. Previous publications (such as [6], [7], [8], [12], [13]) have discussed proactive adaptation techniques mainly in isolation and confined to individual phases of the service life-cycle. A first, more integrated view on adaptation has been presented in [14]. However, the focus was on reactive adaptation and on the design time activities needed to build adaptive service-systems. In contrast, in this paper, we demonstrate how the techniques for determining proactive adaptation play together across the various life-cycle phases and how they can be jointly applied in a meaningful way. As a basis for our discussions, we employ the S-Cube service life-cycle model [15], [14], [16], [17]. In contrast to more traditional life-cycle models, this model considers the specifics of service-based systems, particularly concerning evolution and adaptation.

The remainder of the paper is structured as follows: In Section II, the S-Cube service life-cycle model is introduced. In Section IV, the S-Cube techniques that jointly allow building proactive service-based systems are discussed, differentiating between activities that are done during design-time and activities that are done during the operation phase (run-time). This discussion is illustrated by an example from the eGovernment domain, which is introduced in Section III.

II. The S-Cube Service Life-Cycle Model

The life-cycle models for SBAs that have been presented in the literature (examples include SLDC, RUP for SOA, SOMA, and SOAD, cf. [14] and [18]) are mainly focused on the phases that precede the release of software and, even in the cases in which they focus on the operation phases, they usually do not consider the possibility for SBAs to adapt dynamically to new situations, contexts, requirement needs, service faults, etc.

Specifically, the following aspects have not yet been considered in those life-cycle models:

- *Requirements elicitation and design for adaptation:* The requirements engineering phase includes the elicitation and documentation of the systems functional and quality requirements. In the dynamic setting of SBAs, not only the requirements towards the actual application logic need to be analyzed, designed, and developed, but also the context in which the system is executed needs to be understood [1]. Context changes can necessitate the adaptation of the SBA, for instance if the SLA of a third party service is violated. During design, the capabilities to observe, modify and change the SBA during run-time need to be devised.
- *Extended operation phase:* The operation phase is not only responsible for merely executing and monitoring the application, but it also requires identifying the need for an adaptation of the system as well as the where and how to enact such an adaptation [1].
- *Continuous quality assurance:* Quality assurance has an impact on all aspects of the life-cycle. Therefore, the quality characteristics that are to be assessed and ensured must be identified starting from the requirement analysis phases. Due to open nature, and dynamic contexts in which SBAs operate, quality properties that have a lifelong validity need to be "continuously" asserted [19]. For instance, in the case of third party services, there is no guarantee that a service implementation eventually fulfills the contract promised (e.g., stipulated by an SLA), or it is usually not possible during design-time to model and thus assess the behavior of the underlying distributed infrastructure (such as the Internet).

The service life-cycle model envisioned by the S-Cube network aims at incorporating those aspects. The S-Cube service life-cycle model [15], [14], [16], [17] relies on two development and adaptation loops, which can be executed in an incremental and iterative fashion:

[1] http://www.s-cube-network.eu/

- The *development and evolution loop* (see right hand side of Figure 1) addresses the classical development and deployment life-cycle phases, including requirements and design, construction and operations and management (see Section II-A).
- The *operation and adaptation loop* (see left hand side of Figure 1) extends the classical life-cycle by explicitly defining phases for addressing changes and adaptations during the operation of service-based applications (see Section II-B).

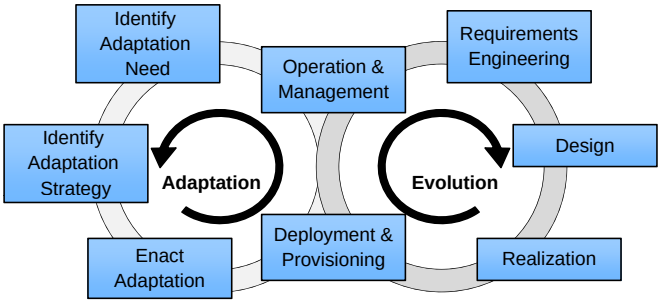

Fig. 1. The S-Cube Service Life-Cycle

A. Development and Evolution Cycle

Requirements Engineering. In the requirements engineering phase, the functional and quality requirements for the SBA are elicited and documented. The specifics of SBAs make the requirements engineering phase particularly relevant. This is related to the highly dynamic nature of SBAs and to the necessity to guarantee the continuous adaptability and the evolvability of these applications. Indeed, in a context where the application is in continuous evolution and is characterized by very blurred boundaries, the study of those requirements that exist a priori in the organizational and business setting, and that are hence largely independent from the solution, becomes very important.

Design. During the design phase, the activities and the control flow of the application are specified. In the service-oriented case, this usually means that a workflow is specified using languages such as BPEL. Together with the definition of the workflow, candidate services are identified that can provide the functionality and quality to fulfill the requirements of the SBA. This means that those services that cover, at least partially, the expected functionality and quality of service are identified. This is supported by service matchmaking techniques, such as the ones presented in [20]. A further task in this phase is to define adaptation strategies and mechanisms which enable the application to react to adaptation needs (cf. [14]).

Construction. After the design phase, the construction of the system can start. Especially, it has to be taken into account that SBAs are obtained by the integration and coordination of services from different providers. Specifically, this means that for establishing the desired end-to-end quality of those SBAs, contracts between the service providers and the service consumers on quality aspects of services have to be established. Typically, this requires some form of SLA negotiation and agreement. Following [20], this means that for each service, the best quality of service level for the available budget is negotiated with the providers of the candidate services that have been identified in the previous phase.

Deployment and Provisioning. The deployment and provisioning phase comprises all the activities needed to make the SBA available to its users. It should be noted that an SBA can itself be offered as a service.

B. Operation and Adaptation Cycle

Operation and Management. This phase specifies all the activities needed for operating and managing an SBA. The literature also uses the term governance to mean all activities that govern the correct execution of SBAs (and their constituent services) by ensuring that they provide the expected functionality and level of quality during operation. In this setting, the identification of problems in the SBA (e.g., failures of constituent services) and of changes in its context play a fundamental role. This identification is obtained by means of monitoring mechanism and, more generally, by exploiting techniques for run-time quality assurance (such as online testing or run-time verification). Together, those mechanisms and techniques are able to detect failures or critical conditions.

Identify Adaptation Need. Some failures or critical conditions become triggers for the SBA to leave “normal” operation and enter the adaptation or evolution cycle. The adaptation cycle is responsible for deciding whether the SBA needs to be adapted in order to maintain its expected functionality and quality (i.e., to meet its requirements). This is an important decision as it might well be that despite a failure of a service, the end-to-end quality of the SBA is not affected and hence there is no need to react to that situation. Such decisions may be made automatically, or it may require human intervention (end user, system integrator, application manager). Moreover, such decisions may be made in a reactive way, when the problem has already occurred, or in a proactive way, where a potential, future problem could be avoided. It should be noted that the decision could also be that there should be an evolution of the system rather than an adaptation, thereby entering the “development and evolution” cycle.

Identify Adaptation Strategy. When the adaptation needs are understood, the corresponding adaptation strategies are identified and selected. Possible types of adaptation strategies include service substitution, SLA re-negotiation, SBA re-configuration or service re-composition. It could also happen that several adaptation strategies are able to satisfy a specific adaptation need. The selection of the strategy and its instantiation (e.g., which service to use as a substitute or which re-configuration to perform) may be automatic if either the SBA or the execution platform decide the action to perform, or it can be done by (the help of) a human operator. Specifically, two questions need to be answered: ”what to adapt?” and ”how to adapt?”.

Enact Adaptation. After the choice of the adaptation strategy, the adaptation mechanisms are used to enact the adaptation. For example, service substitution, re-configuration or re-composition may be obtained using automated service discovery and dynamic binding mechanisms, while re-composition may be achieved using existing automated service composition techniques. Depending on the situation, such an adaptation can be done manually (e.g., by a human operator), semi-automatically or fully-automatically.

III. Application Scenario

In this section, an example workflow is introduced in order to illustrate the problems as well as the solution that will be presented in Section IV. The workflow specifies an eGovernment SBA that allows citizens to pay parking tickets online, thereby saving effort and costs (see [21] for a description of the eGovernment application domain as defined in S-Cube).

A. Workflow

The workflow as well as the service composition of the eGovernment application are depicted in Figure 2 as an extended activity diagram. The gray boxes denote concrete services that can be composed to an eGovernment application. In the example, each service is provided by a third party, being it an external organization or a different unit of the governmental organization. Solid connections between workflow actions and services denote the bindings established at deployment time. Dashed connections denote possible alternative services (from a different provider). In addition, the diagram is annotated with information about the negotiated response times (which could be stipulated by means of SLAs).

Let us assume that the overall workflow is expected to have a response time of at most 1250 ms. This quality requirement can be satisfied by the bound services, provided that they meet their negotiated maximum response times (as, altogether, the maximum response times along the longest path add up to 1200 ms).

In the following subsections we use this example to illustrate the shortcomings of reactive adaptation, which have been introduced in Section I-A. We assume that the *ePay* service of the example workflow fails during runtime, i.e., takes longer than the negotiated maximum response time.

B. Scenario A: Requirements Monitoring

As mentioned in Section I-A there are approaches which are restricted to monitoring of requirements. In that case monitoring events might arrive so late that an adaptation of the SBA is not possible anymore. In our example, the *ePay* service invoked by *Make Payment* might take 650 ms to respond instead of the negotiated maximum response time of 400 ms (see Scenario A in Figure 3).

Due to the fact that only the requirement (maximum response time of 1250 ms) is monitored, this failure is not registered until after *Sign* has been invoked. As a consequence the mechanism was not able to prevent the deviation from the requirements, even though the failure has occurred much earlier (see Δ in Figure 3).

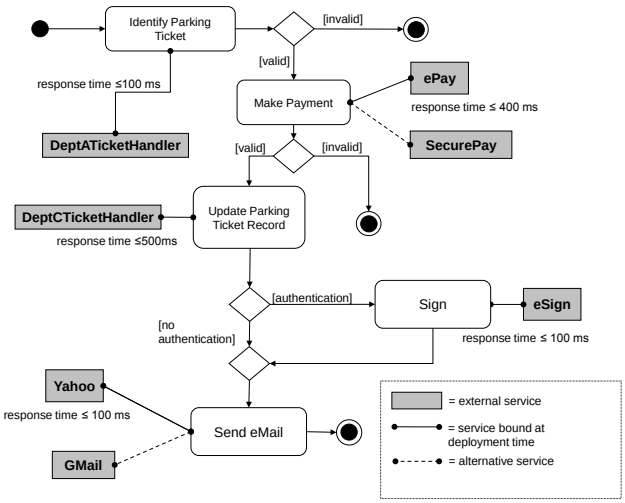

Fig. 2. Workflow of an eGovernment Application

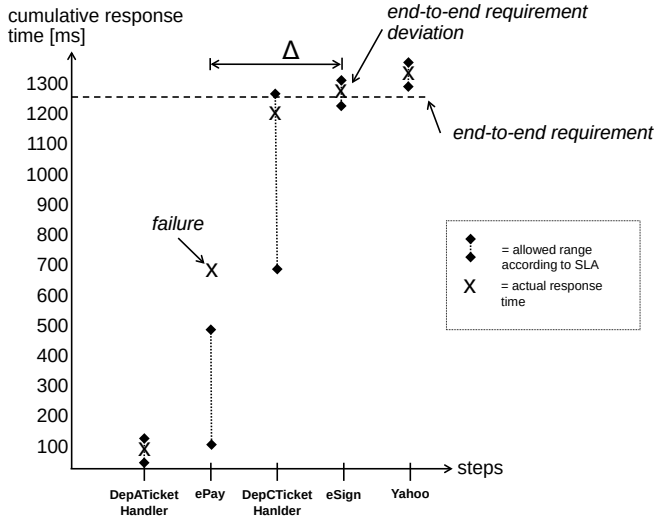

Fig. 3. Scenario A: Requirements Monitoring

C. Scenario B: Service Monitoring

Referring to Section I-A, approaches that monitor individual services exist. However, in such setting it remains unclear whether the failure of a single service leads to a violation of the SBA's requirements. In the example, let us assume that instead of 400 ms the *ePay* service invocation takes 450 ms (see Scenario B in Figure 4).

This failure is observed by means of monitoring and leads to an adaptation of the SBA. However, as obvious in the figure, the overall response time would have still matched the required response time even if no adaptation would have been performed. Thus, in this case an adaptation was triggered although it was not necessary.

In the next section we will present techniques that enable a more proactive approach to addresses the above shortcomings.

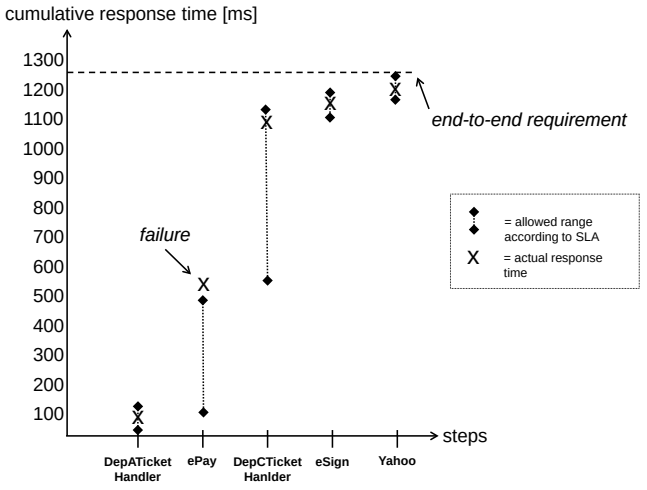

Fig. 4. Scenario B: Service Monitoring

IV. Towards Proactive Adaptation along the Life-Cycle

This section describes techniques developed in S-Cube for enabling proactive adaptation. The description is organized along the phases of the life-cycle model from Section II. In order to illustrate the techniques, we refer to the example SBA and scenarios presented in Section III.

As explained in Section I-A, adaptive SBAs automatically and dynamically adapt to changing conditions and changes of service functionality and quality. To enable such an automatic adaptation, the relevant artifacts, as well as the properties of the SBAs and their context need to be formalized to make them amenable to automated checks and decisions. In the remainder of this section, we thus introduce concrete formalization approaches, as well as techniques that build on this formalization.

A. Requirements Engineering

To automatically assess whether the application deviates from its requirements during operation and thus trigger an adaptation, functional and non-functional requirements need to be collected and formally expressed. We propose to formalize the requirements already in the requirements engineering phase, as this also facilitates an early validation of the requirements, e.g., by means of formal consistency checks (cf. [22]), and hence reduces the risk of expensive corrections in later phases.

S-Cube has developed various approaches to formalize requirements (depending on the actual SBA type). For instance ALBERT is a specification language based on temporal logics (presented in [12]). ALBERT is used to encode functional and quality attributes. In addition, the S-Cube Quality Meta Model (QMM) has been defined, which provides a set of key concepts for expressing quality requirements and constraints (see [23]).

To express the requirements for monitoring, an integrated monitoring framework and the corresponding specification language has been provided (see [24] and [25]) in the scope of S-Cube. The framework integrates the capabilities of two monitoring platforms: Dynamo [11] and ASTRO [9]. On the one hand, the language enables the specification of complex point-wise properties over service composition execution (e.g., pre- and post-conditions on service calls), taking into account current and historical values of the process variables, complex constraints, and event external properties. On the other hand, simple events and point-wise properties may be aggregated into complex behavioral expressions, also taking into account temporal and statistical information necessary for capturing non-functional requirements. While the latter capability is very close to the approach used by ALBERT, the notation allows for expressing properties over classes of processes rather than over single instances. This capability may be very important in order to trigger "evolution" of the workflow, when the problem applies to the whole SBA model rather than to a single SBA instance.

Example: In our example from Section III, we need to formalize the required response time r_{perf} of the eGovernment application. r_{perf} is an element of the given set of requirements R_{eGov} against the eGovernment application:

$$r_{perf} \in R_{eGov}$$

r_{perf} demands the response time of the eGovernment application to be at most 1250 milliseconds. Due to the capability of ALBERT to express the dependencies of monitoring data along an executed path, we choose this language to specify the requirement (r_{perf}) as follows:

$$\begin{aligned} r_{perf} \quad := \quad & onEvent(start, "Identify\ Parking\ Ticket") \\ & \rightarrow Within(onEvent(end, "Send\ eMail"), \\ & 1250) \end{aligned}$$

The $onEvent$ operator evaluates to true if the activity specified in the second argument performs the state change denoted in the first argument. The $Within$ operator evaluates to true if its first argument evaluates to true within the amount of milliseconds specified in its second argument.

B. Design

Similarly to the requirements, the workflow of the SBA needs to be formalized to support automated checks. Following the same reasoning as in the requirements engineering phase (see above), we suggest to formalize the workflow during the design phase already in order to reduce the risk of later corrections. As presented in [8] the checks can be performed by using Model Checking techniques. In S-Cube the use of BOGOR has been proposed [12] to assess whether the specified SBA satisfies the requirements. We thus formalize the workflow using the input language of the model checker, in this case BIR (Bogor Input Representation).

Example: In order to use the BOGOR Model Checker (as proposed in [12]) we specify the eGovnerment Workflow by

using BIR. The resulting specification S_{eGov} (see Listing 1 in the appendix) can be directly executed and analyzed by BOGOR.

C. Realization

During the realization phase, the quality levels (aka. service level objectives) that have been negotiated and agreed upon with the service providers (see Section II), are formalized.

Following the proposal in [8], we treat those quality levels as assumptions (A) about the SBA's context. Due to the lack of control of third-party services, those quality levels could be violated during the operation of the SBA (see Section I). To formalize A, we can use one of the quality formalization approaches as used during the requirements engineering phase.

For checking the violation of the assumptions during the operation of the SBA, monitoring mechanisms are implemented that collect the relevant data (cf. [24], [25] and [12]). This is equivalent to collecting the monitoring data in the reactive case of adaptation (cf. Section III).

Example: According to their SLAs (see Figure 2) ALBERT is used to formalize the five assumed response times. The set of assumptions A_{eGov} for the parking ticket SBA is defined as

$$\begin{aligned} A_{eGov} \quad := \quad & \{a_{DeptATicketHandler}, \\ & a_{ePay}, a_{DeptCTicketHandler}, \\ & a_{eSign}, a_{Yahoo}\} \end{aligned}$$

The assumption a_{ePay}, related to the *ePay* service invocation, is formalized as follows:

$$\begin{aligned} a_{ePay} \quad := \quad & onEvent(start, "Make\ Payment") \\ & \rightarrow Within(onEvent(end, \\ & "Make\ Payment"), 400) \end{aligned}$$

D. Deployment

Before deploying the SBA, it is checked whether the workflow specification (S), under the given assumptions (A), satisfies the requirements (R):

$$S, A \models R$$

This check ensures that the initial composition – the workflow and the services – satisfy the requirements. If this is not the case, the phases of the evolution loop (cf. life-cycle in Section II-A) are executed again in order to redesign the application, e.g., to bind faster services. If the SBA is successfully verified against the requirements the SBA is deployed.

Example: In our example S_{eGov} and A_{eGov} satisfy R_{eGov}. In consequence the SBA is deployed.

E. Operation and Management

This phase comprises the execution and the monitoring of the individual services of the deployed SBA.

Monitoring is supported by monitoring frameworks, such as Dynamo (presented in [25]). During runtime, the monitoring framework continuously assesses whether the monitoring data M satisfies the formalized assumptions A about the services:

$$M \models A$$

If a violation occurs, the SBA enters the adaptation loop (cf. Section II-B). The relevant activities are described below (Sections IV-F, IV-G, and IV-H).

Example: After finishing the SBA deployment, the eGovernment application is executed. Let us assume that the first activity, which invokes the $DeptATicketHandler$ service, lasts 90 milliseconds. The measured response time of the $DeptATicketHandler$ call is stored as monitored data $m_{DeptATicketHandler}$. $m_{DeptATicketHandler}$ satisfies the assumption that the service responds within 100 milliseconds ($a_{DeptATicketHandler}$). In the next step $ePay$ is invoked. Let us assume, that the invocation of $ePay$ is slower than expected. This is the same situation as described in Scenarios A and B (see Section III). Instead of 400 milliseconds as expected, the ePay invocation takes 450 milliseconds (cf. Scenario B). Hence, the monitoring data of the second service invocation m_{ePay} doesn't satisfy the corresponding assumption a_{ePay}:

$$m_{ePay} \not\models a_{ePay}$$

Due to this violation the phases of the adaptation loop are entered.

F. Identify Adaptation Needs

In this phase it is checked, whether the requirements are still satisfied, although the assumptions have been violated (cf. [8]). For example it might be the case that a slower response time of one service is compensated by a faster response time of a previous service, and consequently no adaptation is required.

When the check is performed, there usually are services which have not been invoked. Only when the workflow is finished, all services have been invoked. This means, that there is no monitoring data available for the not yet invoked services. For those not yet invoked services we continue to use their assumptions in the checks, i.e. we use a subset $A' \in A$. Next, it is checked, whether the workflow specification S, the monitored data M and the assumptions in A' satisfy the given requirements R.

$$S, M, A' \models R$$

If R is satisfied, then the workflow execution is continued. If R is not satisfied, the SBA must be adapted.

Example: To illustrate that the presented S-Cube approach is adequate to address the shortcomings from III-B and III-C, we compare the S-Cube approach with the requirements monitoring approach presented in Section III-B (Scenario A) and the sequence monitoring approach presented in Section III-C (Scenario B). It is checked, whether there is a deviation from the requirements, as this could indicate that an adaptation is necessary. This check also covers cases with larger delays, e.g., 500 milliseconds in Scenario A.

The approach presented in Scenario A (see Section III-B) does not observe failures at the moment when they occur –

as depicted with Δ in Figure 3. The S-Cube approach does not have this shortcoming. The continuous monitoring of the service behavior observes failures as soon as a problem occurs. Based upon such an observation, the SBA requirements are immediately checked. This provides the system with the opportunity to adapt itself to prevent the predicted requirements deviation from occurring. Of course the ability of the system to proactively adapt depends on the time available for such actions. Typically, if a failure of a service is observed more at the beginning of the workflow, more time remains to adapt the remainder of the workflow accordingly.

In order to determine such requirements violations, Model Checking techniques are used. The workflow specification (S_1), the monitoring data ($m_{DeptATicketHandler}$ and m_{ePay}) together with the assumptions of the outstanding service invocations ($a_{DeptCTicketHandler}$, a_{eSign} and a_{Yahoo}) are checked against the requirement r_{perf}. The expected overall runtime is 1450 milliseconds which exceeds the 1250 milliseconds demanded in r_{perf}. Hence, the requirement r_{perf} is not satisfied. This result is considered as an identified adaptation need. Subsequent to this check, the adaptation can be performed proactively, before the requirement is actually violated (i.e., before the system in operation deviates from its expected requirements).

The approach presented in Scenario B (see Section III-C) is not able to determine, whether a failure of a single service leads to a violation of the SBAs requirements. Each time a service fails, the SBA adapts immediately. The S-Cube approach presented in this paper allows adapting only in cases when critical failures occur, thereby avoiding unnecessary adaptations. The same check as described above assesses that the expected overall runtime does not exceed 1250 milliseconds. The requirement r_{perf} is still satisfied and thus no adaptation trigger is needed. Thereby an unnecessary adaptations is prevented, which would have been performed in Scenario B.

G. Decide on Adaptation / Identify adaptation strategy

When the need for adapting an SBA is detected, the next step is to identify and apply an appropriate adaptation strategy among the ones that are available for the considered applications. Depending on the application, the adaptation strategies may range from service re-execution, over replacement of a single service or of the process fragment, over re-negotiation of quality properties, to changes in underlying infrastructure, etc. Note that the adaptation strategies should be designed with the application since some of them require the adoption of specific infrastructure or the implementation of additional components.

Typically, the adaptation strategy is associated with a specific critical situation or a problem at design time. This association may be done either implicitly or explicitly. In the former case, the mechanisms for choosing one action or another are "hard-coded" in some decision mechanisms. A typical scenario is the replacement of a service that violates the SLA or a SBA requirement with a new one, with appropriate and most suitable characteristics. Based on the selection criteria (e.g., optimization of a quality function, adherence to application constraints), the appropriate decision mechanism may choose one service or another. In the scope of the S-Cube project, several approaches follow this vision. For example, in [26] the replacement policies realize such a decision mechanism and define the association between various types of changes (service failure, changes in service properties and models, appearance of new services, and changes in the context and requirements) and the service selection. In [27], the decision on the adaptation strategy is based on the quality factors of the SBA that should be improved. Those factors are identified through the analysis of the dependency tree that capture the relation between simple quality factors and SBA requirements. At design time, the adaptation action is assigned to the quality factors that it influences either positively or negatively. The selection of the adaptation strategy is based on the need to improve quality factors that are critical for the requirement, while trying to minimize the negative effect on the other factors. In our scenario, the requirement would need to improve the performance of the last service, and the service replacement would be proposed such that the new service has better performance, while having smaller cost with respect to alternatives.

The definition of the adaptation strategy may be also explicitly assigned to the critical situation. For example in [28] the adaptation strategy is represented in the WS-ReL, a notation for specifying and integrating recovery actions in service composition. Therefore, the adaptation is defined as a rule, where in the left hand side a critical situation is defined (as a formal requirement to be monitored) and in the right hand side a set of actions to be applied. The possible actions include re-execution of a service invocation, replacement of a service or a provider (partner link), ignoring the failure or halting the execution, executing an extra process fragment, or rolling back to a safe point. Simple actions may be joined into a complex strategy by defining a control flow over actions, like "try action A else try action B and action C". These rules are evaluated and applied by the underlying adaptation engine.

Adaptation can also be based on the causes of failures. This is particularly helpful when invoked services are stateful, and their invocation modifies the state of the service, such as in transactional services. For processes involving transactional services, if a diagnosis mechanism is available, such as in [29], the adaptation strategy can depend on the cause of the failure and its implications on the processes. This might imply an adaptation strategy involving one or more services in the process which must be dynamically generated.

H. Enact Adaptation

To enact adaptation actions, the SBA or its execution platform should be appropriately instrumented. A typical approach for realizing adaptation mechanisms for SBAs implemented as executable (BPEL) processes is to instrument the process execution engine. Such instrumentation is done via Aspect-Oriented Programming techniques, as the adaptation activities

are treated as a cross-cutting concern. Using this approach, the join points allow for injecting the adaptation logic in order to intercept and adjust process execution logic. In particular, in [28] a supervision manager component is attached to the ActiveBPEL process engine and performs the necessary supervision activities: monitoring of critical situations, evaluation of adaptation rules, and calls to the process engine infrastructure to realize the specific strategy. Similarly, in [30], where aspect-oriented techniques are adopted in order to dynamically bind services into service compositions that are realized as BPEL orchestrations.

V. Conclusion and Perspectives

This paper has introduced novel techniques developed in S-Cube (the European Network of Excellence on Software, Services and Systems) for equipping service-based applications with proactive adaptation facilities. Thereby, those techniques are able to avoid costly compensation and repair activities, as well as unnecessary adaptations, which are deemed key shortcomings of current solutions for adaptive service-oriented applications.

The techniques have been introduced along the key phases of the S-Cube service life-cycle. Thereby, this paper has demonstrated when and how those techniques should be applied when developing, evolving and adapting service-based applications.

We are confident that those techniques will become especially relevant in the setting of the "Internet of Services", where applications will increasingly be composed from third party services, which are not under the control of the service consumer. This implies that applications and their constituent services need to be continuously checked during their operation such that they can be dynamically adapted or evolved in order to respond to failures or unexpected changes of third party services.

In S-Cube, we are currently striving to push the envelope towards proactive adaptation even further. In addition to determining the need for adapting the service-based application based on actual failures of the application's constituent services, we investigate the applicability of online testing for predicting the quality of those services (e.g., see [6], [7]). Combined with the approaches introduced in this paper, this means that critical problems could be observed even earlier, thus enabling a broader range of adaptation and evolution strategies. For instance, in our running example we can only react to the violation of the response time of a constituent service by ensuring that the remainder of the workflow executes faster. However, if the quality prediction techniques forecast a violation of the expected response time of a specific service, this very service can be replaced before it is invoked in the context of the service-based application.

Another challenging problem that will be addressed is how to make the techniques robust against other kinds of "false-positives". Currently our techniques define the assumptions about a service execution to be the upper limits of the quality properties as stated in the SLAs. As a consequence, it might happen that the proactive techniques predict a performance requirements violation based on a failure of a service despite the fact that the remaining service invocations of the workflow might have executed much faster than stated in the SLAs and thus compensating for this failure. We thus will investigate in how far past monitoring data could be used to better define the assumptions that can be made about the quality properties of a service.

Acknowledgment

We thank the anonymous reviewers for their helpful and constructive comments.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). For further information please visit http://www.s-cube-network.eu/.

References

[1] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, "A journey to highly dynamic, self-adaptive service-based applications," *Automated Software Engineering*, 2008.

[2] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and opportunities," *IT professional*, vol. 8, no. 2, pp. 10–17, 2006.

[3] A. Metzger and K. Pohl, "Towards the next generation of service-based systems: The s-cube research framework," in *CAiSE 2009*, ser. LNCS, J. P. van Eck, J. Gordijin, and R. Wieringa, Eds. Berlin Heidelberg: Springer-Verlag, 2009, pp. 11–16.

[4] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, 2009.

[5] S. Benbernou, "State of the art report, gap analysis of knowledge on principles, techniques and methodologies for monitoring and adaptation of sbas," S-Cube Consortium, Deliverable PO-JRA-1.2.1, July 2008. [Online]. Available: http://www.s-cube-network.eu/results/

[6] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka, "Towards proactive adaptation with confidence augmenting service monitoring with online testing," in *Proceedings of the ICSE 2010 Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS '10)*, Cape Town, South Africa, 2-8 May 2010.

[7] J. Hielscher, R. Kazhamiakin, A. Metzger, and M. Pistore, "A framework for proactive self-adaptation of service-based applications based on online testing," in *ServiceWave 2008*, ser. LNCS, no. 5377. Springer, 10-13 December 2008.

[8] A. Gehlert, A. Bucchiarone, R. Kazhamiakin, A. Metzger, M. Pistore, and K. Pohl, "Exploiting assumption-based verification for the adaptation of service-based applications," in *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2010, pp. 2430–2437.

[9] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-Time Monitoring of Instances and Classes of Web Service Compositions," in *IEEE International Conference on Web Services (ICWS 2006)*, 2006, pp. 63–71.

[10] D. Dranidis, E. Ramollari, and D. Kourtesis, "Run-time verification of behavioural conformance for conversational web services," in *Seventh IEEE European Conference on Web Services (ECOWS 2009), 9-11 November 2009, Eindhoven, The Netherlands*, R. Eshuis, P. W. P. J. Grefen, and G. A. Papadopoulos, Eds., 2009, pp. 139–147.

[11] C. Ghezzi and S. Guinea, "Run-time monitoring in service-oriented architectures," in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds., 2007, pp. 237–264.

[12] D. Bianculli, C. Ghezzi, P. Spoletini, L. Baresi, and S. Guinea, *Advances in Software Engineering*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2008, vol. 5316, ch. A Guided Tour through SAVVY-WS: a Methodology for Specifying and Validating Web Service Compositions, pp. 131–160.

[13] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime prediction of service level agreement violations for composite services," in *3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing, co- located with ICSOC 2009*, 2009.

[14] A. Bucchiarone, C. Cappiello, E. D. Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore, "Design for adaptation of service-based applications: Main issues and requirements," in *Fifth International Workshop on Engineering Service-Oriented Applications: Supporting Software Service Development Lifecycles (WESOA)*, 2009.

[15] A. Gehlert, M. Pistore, P. Plebani, and L. Versienti, "First version of integration framework," S-Cube Consortium, Deliverable CD-IA-3.1.3, December 2009. [Online]. Available: http://www.s-cube-network.eu/results/

[16] A. Bucchiarone, R. Kazhamiakin, C. Cappiello, E. Di Nitto, and V. Mazza, "A context-driven adaptation process for service-based applications," in *PESOS 2010 - 2nd International Workshop on Principles of Engineering Service-Oriented Systems. (To Appear)*, Cape Town, South Africa, 1-2 May 2010.

[17] B. Pernici, *Methodologies for Design of Service-Based Systems*. Springer, 2010, ch. 17.

[18] E. Nitto, "State of the art report on software engineering design knowledge and survey of hci and contextual knowledge," Deliverable PO-JRA-1.1.1, 2008. [Online]. Available: http://www.s-cube-network.eu/results/

[19] D. Bianculli, C. Ghezzi, and C. Pautasso, "Embedding continuous lifelong verification in service life cycles," in *Proceedings of Principles of Engineering Service Oriented Systems (PESOS 2009), co-located with ICSE 2009, Vancouver, Canada*. IEEE Computer Society Press, May 2009.

[20] M. Comuzzi and B. Pernici, "A framework for qos-based web service contracting," *ACM Transactions on web*, vol. 3, no. 3, 2009.

[21] E. D. Nitto, V. Mazza, and A. Mocci, "Collection of industrial best practices, scenarios and business cases," S-Cube Consortium, Deliverable CD-IA-2.2.2, 2009. [Online]. Available: http://www.s-cube-network.eu/results/

[22] F. L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, and H. Wössner, *The Munich Project CIP: Volume I: the wide spectrum language CIP-L*. London, UK: Springer-Verlag, 1985.

[23] A. Gehlert and A. Metzger, "Quality reference model for SBA," Deliverable CD-JRA-1.3.2, 2008. [Online]. Available: http://www.s-cube-network.eu/results/

[24] L. Baresi, S. Guinea, R. Kazhamiakin, and M. Pistore, "An integrated approach for the run-time monitoring of bpel orchestrations," in *ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–12.

[25] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, "Dynamo + astro: An integrated approach for bpel monitoring," *Web Services, IEEE International Conference on*, vol. 0, pp. 230–237, 2009.

[26] K. Mahbub and A. Zisman, "Replacement Policies for Service-Based Systems," in *2nd Workshop on Monitoring, Adaptation and Beyond (MONA+), co- located with ICSOC 2009*, 2009.

[27] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann, "Adaptation of service-based applications based on process quality factor analysis," in *Proc. Of 2nd Intl. workshop on Monitoring, Adaptation and Beyond (MONA+)*, Collocated with ICSOC/ServiceWave'09, 2009.

[28] L. Baresi, S. Guinea, and L. Pasquale, "Integrated and composable supervision of bpel processes," in *International Conference on Service-Oriented Computing (ICSOC)*, 2008, pp. 614–619.

[29] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, "Exception handling for repair in service-based processes," *IEEE Trans. Software Eng.*, vol. 36, no. 2, pp. 198–215, 2010.

[30] D. Karastoyanova and F. Leymann, "Bpel'n'aspects: Adapting service orchestration logic," in *International Conference on Web Services (ICWS)*, 2009, pp. 222–229.

APPENDIX

Listing 1. Workflow Specification S_{eGov}

```
system ParkingTicketSpec {
 Action identifyParkingTicket;
 Action makePayment;
 Action updateParkingTicketRecord;
 Action sign;
 Action sendEMail;

 int sumMaxResponseTime := 0;

 record Action {
  string serviceName;
  int maxResponseTime;
  boolean serviceInvoked;
 }

 active thread MAIN () {
  init();
  checkWorkflow();
  checkRequirements();
 }

 function init() {
  identifyParkingTicket :=
  createAction("DeptATicketHandler", 100);
  makePayment := createAction("ePay", 400);
  updateParkingTicketRecord :=
        createAction("DeptCTicketHandler", 500);
  sign := createAction("eSign", 100);
  sendEMail := createAction("Yahoo", 100);
 }

 function checkWorkflow() {
  executeAction(identifyParkingTicket);
  choose
   do skip;
   do
    atomic
     executeAction(makePayment);
     choose
      do skip;
      do
       atomic
        executeAction(updateParkingTicketRecord);
        choose
         do executeAction(sign);
         do skip;
        end
        executeAction(sendEMail);
       end
    end
   end
 end
 }

 function checkRequirements() {
  assert sumMaxResponseTime <= 1250;
 }

 function executeAction(Action action) {
  sumMaxResponseTime := sumMaxResponseTime +
        action.maxResponseTime;
  action.serviceInvoked := true;
 }
 function createAction(string serviceName,
  int maxResponseTime) returns Action{
  Action action;
  action := new Action;
  action.serviceName := serviceName;
  action.maxResponseTime := maxResponseTime;
  action.serviceInvoked := false;
  return action;
 }
}
```

Design for Adaptation of Service-Based Applications: Main Issues and Requirements

Antonio Bucchiarone[2], Cinzia Cappiello[1], Elisabetta Di Nitto[1], Raman Kazhamiakin[2], Valentina Mazza[1], and Marco Pistore[2]

[1] Politecnico di Milano
Piazza Leonardo Da Vinci 32 20133 Milano, Italy
[2] Fondazione Bruno Kessler
Via Santa Croce 77 38100 Trento, Italy
{bucchiarone,raman,pistore}@fbk.eu,
{cappiell,dinitto,vmazza}@elet.polimi.it

Abstract. Service-based applications are considered a promising technology since they are able to offer complex and flexible functionalities in widely distributed environments by composing different types of services. These applications have to be adaptable to unforeseen changes in the functionality offered by component services and to their unavailability or decreasing performances. Furthermore, when applications are made available to a high number of potential users, they should also be able to dynamically adapt to the current context of use as well as to specific requirements and needs of the specific users.
In order to address these issues, mechanisms that enable adaptation should be introduced in the life-cycle of applications, both in the design and in the runtime phases. Unfortunately, existing design methodologies do not take into account the problem of service-based applications adaptation in a holistic way, but only in a fragmented way, proposing specific solutions for particular cases.
In this paper we propose an extension of a basic iterative service-based applications life-cycle with elements able to deal with the adaptation-specific needs. We focus, in particular, on the design phase and suggest a number of design principles and guidelines that are suitable to enable adaptation. We discuss about the effectiveness of the proposed methodology by means of real-world scenarios over various types of service-based applications.

Keywords: Life-cycle of service-based applications, Design for adaptation

1 Introduction

In the era of the Internet of Services, advanced service-based applications are considered the most promising technology since they are able to offer complex and flexible functionalities in widely distributed environments by composing different types of services. Such services are often not under the control of systems developers, but they are simply exploited to obtain a specific functionality. While this, on the one side, enables separation of concerns and highly simplifies the design effort of those in charge of

building service-based applications, on the other side, it introduces critical dependencies between service-based application themselves and the services they are exploiting. These last ones, in fact, could change without notice or be unavailable for unprecised time intervals.

Therefore, service-based applications have to be able to *adapt* to these unforeseen changes. Adaptation can be accomplished through various strategies that we will discuss in this paper. For example, the most obvious one is to replace a service that has revealed to be unsuitable with some other service that either has been identified earlier (for instance, during the design of the application) or is identified on-the-fly when needed.

Changes of component services are not the only cause that determines the need for adaptation. Another cause has to do with changes in the context in which service-based applications are executed. For instance, this context may include the information about the users of the applications. Users, in fact, may have special needs in terms of the component services to be used. For instance, who is owning a special business contract with a certain telecom provider could require that all telecom services exploited by the service-based application are those offered by that specific provider. Who owns a code for special offers on some flight reservation services may be willing to use them whenever possible. And so on.

While the literature presents a good number of approaches that deal with self-adaptation of service-based applications, most of them 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. We argue that this approach does not necessarily cover all needs that may arise. In some cases these needs are unknown and cannot be foreseen once for all. The approach that we advocate is based on the idea that adaptation strategies can be programmed at design/implementation time and be associated with triggering events whenever possible, either before the execution or during the execution itself. This approach is adopted in our earlier work [3] and in other works (e.g., [15]). However, even in these cases the emphasis is on the mechanisms offered to design strategies and to trigger them, more than on a holistic, coherent, and easy to apply *design for adaptation* approach that supports developers in the usage of the available mechanisms.

The objective of this paper is to go in the direction of this design for adaptation approach. We define a life-cycle for service-based applications where adaptation is a first class concern. As we think that adaptation works properly only in the case the application is designed to be adaptable, we focus, in particular, on the identification of a number of design principles and guidelines that are suitable to enable adaptation. The effectiveness of such principles and guidelines is analyzed with reference to some real-world scenarios.

The rest of the paper is structured as follows: Section 2 discusses about the life-cycles and the adaptation platforms that are available in the literature and highlights their weaknesses with respect to our objective of providing proper support to design for adaptation. Section 3 presents three scenarios that will be used to exemplify the phases, principles, and guidelines we discuss in the paper. Section 4 discusses about the various facets of adaptation and evolution we deal with. Section 5 presents our life-cycle

while Section 6 presents the design for adaptation strategies, principles, and guidelines. Finally, Section 7 assesses them with respect to the case studies and Section 8 concludes the paper.

2 Related work

Life-cycles. Engineering service-based applications is based on the results carried out in the fields of classical software and system engineering, where the definition and analysis of suitable life-cycles has been a main objective in the past years. The current service oriented methodologies propose life-cycles built upon existing techniques often adapting them to SOA.

Almost all life-cycles in the literature do not address explicitly the possibility of a service-based application to be adaptable. With this respect, the proposals by HP [6] and Sun [12] are interesting as they aim to offer a mean to monitor and/or intervene on service-based applications to optimize them. Also the SOUP (Service Oriented Unified Process) [13] approach, based on the Rational Unified Process and on eXtreme Programming, claims to support the possibility of adaptation in order to optimize SOAs and to manage changes in the requirements and/or in the business needs. ASTRO [17] addresses the need of the automatic service composition. The methodology covers all the phases in the life-cycle, starting from requirement analysis to realization, deployment, and execution. It also supports iterations as when the monitor detects some unwanted or unexpected behaviour it is possible to loop back to the design phase. All these approaches, however, assume the intervention of humans during the adaptation and therefore do not support the autonomous, on-the-fly initiative of the application itself. On the contrary, Linner et al. [11] propose a life-cycle based on the application of evolutionary principles for adaptation. These evolutionary principles rely on genetic approaches to support self-adaptation of the service-based application. They, however, do not provide explicit guidelines on how to design adaptable service-based applications and seem to rely on a fixed adaptation strategy.

Frameworks to support adaptation. Various frameworks supporting adaptation have been defined in the literature, each of them addressing a specific issue. Some authors focus on triggering adaptation strategies as a consequence of a requirement violation [16], of the need for optimizing QoS of the service-based application [20, 2], or for satisfying some application constraints [18]. Adaptation strategies could be specified by means of policies to manage the dynamism of the execution environment [4, 3, 1] or of the context of mobile service-based applications [15]. The goal of the strategies usually proposed by the aforementioned approaches range from service selection to rebinding and application reconfiguration. In order to support this last strategy, [14] proposes the adoption of a bioinspired approach.

All aforementioned approaches show interesting features, but even those that enable the definition of various adaptation strategies lack a coherent design approach to support designers in this complex task. The methodology we propose in Section 6 can be considered as a first step in this direction.

3 Motivating scenarios

We have argued that the life-cycles and frameworks that are available in the service-based applications literature either do not explicitly support adaptation or tend to focus on specific situations and adaptation strategies. In the following, we consider three scenarios that target different domains and focus on different adaptation aspects. They clearly show that adaptation needs and strategies have different facets and, therefore, motivate our attempt to define a holistic approach where they are dealt all together.

Automotive scenario. Let us consider complex supply-chain business processes in the automobile production domain. The activities of the processes include ordering and importing automobile body parts from suppliers, manufacturing activities, customization of the specific products according to the needs of the customers, etc. The processes are usually long-running and involve a wide range of enterprise services provided by organizations such as various suppliers, logistics providers, warehouses, and regional representatives. All these participate in an Agile Service Network (ASN) where they rely one on each other services in a dynamic way. The adaptations required in such scenario could be the following:

- The partners realizing some activity can be chosen at run time from those belonging to the ASN. Such choice can be based on various business (e.g., reputation, rule compliance, price, risks, etc.) and QoS (reliability, performance, availability, etc.) factors. Moreover, it may require adjustments in the interaction protocols between the service-based application and the services, and re-negotiation of Service Level Agreements (SLAs).
- The service-based application needs to recover from some problem (e.g., delivery of some car part did not happen on time or transaction failure).
- The service-based application needs to accommodate to some change in the business context (e.g., due to new state regulations) or to address the needs of specific customers (e.g., introduce some specific personalizations in the car).

Note that such adaptations in these kinds of business scenarios are accomplished within well defined boundaries where the ASN is rather stable and the changes in the business context are quite rare. Indeed, they are not necessarily done autonomously by the service-based application as business analysts often have an important role when applying them. Finally, they may result in changes in the invoked services, in the structure of the business process, or in the data that are managed by the process.

Wine production scenario. In the wine production application domain, the activities of vineyard cultivation handling, the control of grapes maturation, their harvesting and fermentation rely on extensive use of a service-based application realized on top of a Wireless Sensor and Actuator Network (WSAN). In this context sensors and actuators are seen as service providers able, respectively, to report information regarding the state of the vineyard and to execute some specific actions.

These devices are not fully reliable. They may crash, run out of battery, or provide incorrect information. This may happen due to changes in physical context (e.g., humidity) or to the activation of new measurement activities (e.g., depending on the

season). One of the requirements for the resulting service-based application is, therefore, the ability to autonomously detect and handle these problems. In particular, the service-based application should be able to change the topology of the network to optimize the load and distribution of sensors, activate new sensors (i.e., replacing a service), signal the crashes, optimize the frequency of data transfer to keep down the power consumption, and so on. While in automotive scenario modification are rare and involve the experts, here the adaptations are highly dynamic and autonomous.

Mobile users scenario. An increasing number of modern applications aims to give end users access to various services through mobile devices. Such services include navigation and route planning services, transport ticket booking, services for accessing social networks and blogging, and a wide range of information services mashed up by those applications. For instance, a trip assistant application exploits those services for travelling to a location, finding points of interest, making reservations, and even sharing the experience with the friends. Differently to the other presented scenarios the distinguishing features characterizing such applications are:

- Continuously changing context: the system heavily depends not only on the operational context (e.g., the network throughput), but also on the user context (e.g., location and time) and even the specific situation the user is in (e.g., whether a business or personal trip is managed). Furthermore, also the used services are localized and personalized accordingly, which requires adequate service description, as well as discovery and engagement mechanisms.
- Continuously changing user goals and activities: there are no predefined business processes, but a set of various activities that may be organized according to the continuously evolving user needs. An appropriate handling of such needs is indeed necessary and may require compensation or transformation of unfinished activities.

Remarks. The above scenarios show that service-based applications have to face very different adaptation needs, including customization, recovery and repair, self-optimization and context-driven adaptation. Moreover, the actions used to realize those activities, as well as the principles underlying the way the adaptation is carried out range drastically. Indeed, in some cases simple service replacement is automatically triggered by service unavailability, while in other cases the application change is entailed by complex decision making process involving human actors. The rest of the paper is focused on presenting a holistic approach that supports the understanding and the design of these various kinds of adaptation aspects.

4 Adaptation and evolution in service-based applications

Figure 1 shows the main ingredients that are needed for building and operating *Adaptable Service-Based Applications*, which, from now on, we will call *Adaptable SBAs* or simply SBAs.

An Adaptable Service Based Application not only is usually able to satisfy some *Requirements*, but it also poses new requirements in terms of monitoring and adaptation

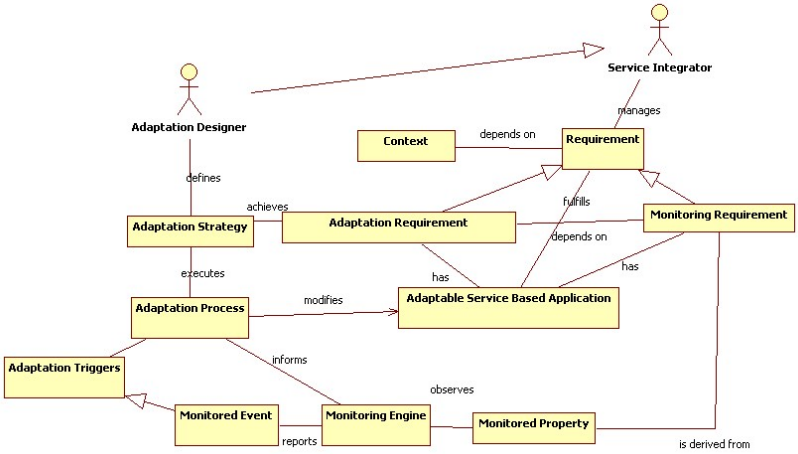

Fig. 1. Main concepts related to the definition and operation of Adaptable SBAs.

aspects. *Monitoring Requirements* concern the need for detecting (part of) those situations that may trigger the need for adapting an SBA. From these requirements, designers should derive the properties to be monitored. These are then observed at runtime by a *Monitoring Engine* that, based on their values, is able to emit some *Monitored Events*. For example, in the *Wine production scenario*, the monitoring of the environmental parameters of the vineyard (such as temperature, wind speed and so on) could lead to the detection of critical conditions to be managed. *Adaptation Requirements* are fulfilled by *Adaptation Strategies* that can be executed during the *Adaptation Process* that is triggered by *Monitored Events* or by any other external stimulus that can be acquired by the system and that leads to the modification of the Adaptable SBA. For instance, in the *Automotive Scenario*, it can happen when new state regulations (i.e., changes in business context) are defined or when the SLAs among services are violated. Note that analogously to the classification traditionally used to characterize software maintanance [7] we can identify similar types of adaptation: Perfective Adaptation, Corrective Adaptation, Adaptive Adaptation, Preventive Adaptation, and Extending Adaptation.

An important role in our view is played by the *Context*. It includes users and execution properties. Users' characteristics and preferences can be obtained explicitly, for instance, by filling a user profile, or derived implicitly by profiling users at run-time. Other information such us the users' geographical position, the temporal details, and the actions that characterize the interaction of the users with the surrounding space can be obtained through monitoring. Execution properties are those that concern the conditions under which the SBA and its component services execute. For instance, in the *Mobile users scenario*, the user context is the main source of adaptation since that its location or time can be source of adaptation needs (i.e., service substitution, re-negotiation, re-execution, etc.).

Generally adaptation requires some temporary modification permitting to respond to changes in the requirements and/or in the application context or to faulty situations. An example of adaptation for a service composition could be the re-execution of a unavailable service or a substitution of a unsuitable service. Other situations could require the re-design and/or the re-engineering of the application modifying it permanently, in such case adaptation is called *evolution*. Moreover evolution could be needed if a faulty situation requiring adaptation happens very often: in such case, a modification of the application logic would be preferred to the frequent enactment of the needed adaptation strategies.

5 Capturing adaptation and evolution aspects in a life-cycle

As discussed in the previous sections, there is a need for introducing a life-cycle for SBAs that takes adaptation into explicit account. The life-cycle shown in Figure 2 aims at filling this gap. Its novelty is that it highlights not only the typical design-time iteration cycle that leads to the explicit re-design of the application in order to adapt it to new needs, but it also introduces a new iteration cycle at runtime that is undertaken in all the cases in which the adaptation needs are addressed on-the-fly. The two cycles clearly are not conflicting with each other. Instead, they coexist and support each other during the lifetime of the application. We say, in particular, that design time activities allow for *evolution* of the application, that is, for the introduction of permanent and, usually, important changes, while the runtime activities allow for temporary *adaptation* of the application to the specific circumstances that are occurring at a certain time.

Figure 2 also shows the various adaptation- and monitoring-specific actions (boxes) carried out throughout the life-cycle of the SBA, the main design artifacts that are exploited to perform adaptation (hexagons), and the phases where they are used (dotted lines). At the *requirements engineering and design* phase the adaptation and monitoring requirements are used to perform the design for adaptation and monitoring. During SBA *construction*, together with the construction of the SBA, the corresponding monitors and the adaptation mechanisms are being realized. The *deployment* phase also involves the activities related to adaptation and monitoring: deployment of the adaptation and monitoring mechanisms and deployment time adaptation actions (e.g., binding). During the *operation and management* phase, the run-time monitoring is executed, using some designed properties, and help the SBA to detect relevant context and system changes. After this phase the left-side of the life-cycle is executed. Here, we can proceed in two different directions: executing evolution or adaptation of the SBA. In the first case we re-start the right-side of the cycle with the requirements engineering and design phase while in the second case we proceed identifying adaptation needs that can be triggered from monitored events, adaptation requirements or context conditions. For each adaptation need it is possible to define a set of *suitable strategies*. Each adaptation strategy can be characterized by its complexity and its functional and non functional properties.

The identification of the most suitable strategy is supported by a *reasoner* that also bases its decisions on multiple criteria extracted from the current situation and from the knowledge obtained from previous adaptations and executions. Details on these issues

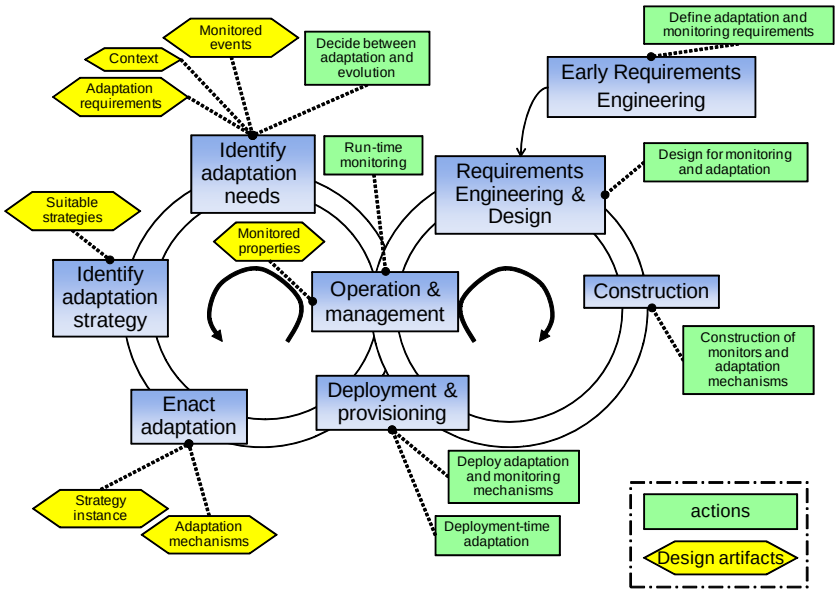

Fig. 2. The Life-Cycle of Adaptable SBAs.

are discussed in Section 6. After this selection, the *enactment of the adaptation strategy* is performed. The execution of all activities and phases in all runtime phases may be performed autonomously by SBAs or may involve active participation of the various human actors.

6 Design for adaptation: main ingredients

As discussed in the previous sections, in order to offer efficient and reliable applications, it is necessary to guarantee that the service components are always aligned with the changing world around them. At design time possible alternatives to support service adaptation should be identified. For the same SBA, several adaptation strategies can be adopted since each adaptation strategy has different functionalities, characteristics, and consequences, and its suitability for dealing with a specific change can be strictly related to the context or to the functional and non-functional application requirements. The selection of the most suitable adaptation strategy to activate can be a complex issue since different system characteristics have to be considered. In the next sections, guidelines to support this selection are provided.

6.1 Adaptation Strategies

While a SBA is executing, different changes might occur in the environment and cause inefficiencies. 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 behaviour with the context and system requirements. Among the adaptation strategies, it is possible to distinguish domain-independent or domain-dependent strategies. The former are applicable in almost every application context while the adoption of the latter is limited to specific execution environments. In the following, we aim at providing a short description of the most common domain-independent adaptation strategies:

- *Service substitution*: reconfiguration of the SBA with a dynamic substitution of the a service with another one.
- *Re-execution*: the possibility of going back in the process to a point defined as safe for redoing the same set of tasks or for performing an alternative path.
- *(Re-)negotiation*: simple termination of the service used on the requester side and re-negotiation of the SLA properties to complex management on reconfiguration activities on the provider side.
- *(Re-)composition*: reorganization and rearrangement of the control flow that links the different service components in the business application.
- *Compensation*: definition of ad-hoc activities that can undo the effects of a process that fails to complete.
- *Trigger evolution*: insertion of workflow exception able to activate the application evolution.
- *Log/update adaptation information*: storing of all the information about the adaptation activities for different goals (e.g., service reputation, QoS analysis, outcome of adaptation, ...).
- *Fail*: the system reacts to the changes by storing the system status and causing the failure of the service and re-executing it.

Considering the examples in Section 3, some domain-dependent strategies can be identified. In fact, the presence of sensors in the wine scenario could enable *Self-optimization* and *New Sensor Activation* strategies that, in case of performance degradation or sensor failures, allow sensor network to reconfigure itself and to activate a new sensor respectively.

6.2 Identification of 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 we can identify the following types:

- changes in the *service functionality*: variation of the service interface (e.g., signatures, data types, semantics), variation of service interaction protocol (e.g., ordering of messages), and failures;
- changes in the *service quality*: service availability, degrade of QoS parameters, violation of SLA, decrease of service reputation (e.g., black lists), etc.

As for the contextual triggers, one can distinguish

Table 1. Relationships between Adaptation Triggers and Adaptation Strategies

Adaptation Trigger	Adaptation Strategy
Changes in the service functionality	Service Substitution, Re-execution, Re-negotiation, Re-composition, Compensation, Fail
Changes in the service quality	Service Substitution, Re-Negotiation
Changes in the business context	Service Substitution, Re-Negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information
Changes in the computational context	Service Substitution, Re-negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information
Changes in the user context	Service Substitution, Re-negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information

– changes in the *business context*, such as changes in agile service networks, new business regulations and policies;
– changes in the *computational context*, such as different devices, protocols, networks;
– changes in the *user context*, such as different user groups and profiles, social environment or physical settings (e.g., location/time), different user activities.

Some of these aspects may be interleaved. For example, in the mobile users scenario, if the user moves to a new location (i.e., new user context), new set of services may be available to the mobile application (i.e., change in business context) with different bandwidth (i.e., change in the computational context).

As represented in Table 1 each trigger can be associated with a set of adaptation strategies that are suitable to re-align the application within the system and/or context requirements. In order to select the adaptation strategy to apply, it is necessary to consider that adaptation triggers may be associated with different requirements that are important for designing and performing adaptation, in particular:

– *Scope* of the change, i.e., whether the change affects only a single running instance of the SBA or influences the whole model. For example, a SLA violation affects a single instance, while change in the business policy may require modification of the whole application.
– *Impact* of the change, i.e., the possibility of the application to accomplish its current task. For example, one of the services may become unavailable, the corresponding task may still be completed by another service, while if the business transaction fails or if the user changes his/her mind, there is a need to accomplish a different task.

Depending on these parameters different strategies may apply. For example, when the scope of the change concerns the whole application model “trigger evolution” strategy applies. As for the impact, such strategies as “re-execution” or “substitution” may apply when the SBA state did not change and the task still can be accomplished. On the other hand, “compensation”, “fail”, or “trigger evolution” apply when there is no way to complete the current task (e.g., business transaction fails, the user changes his mind, critical failure takes place, etc).

6.3 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.

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 to 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. Examples include replacement of a service in automotive domain, where the number of possible partner services is restricted and fixed; concrete activities to be performed for compensating business transaction; change of the process variant in case of business workflow customization. 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 [8] using ECA-like (event-condition-action) rules [3, 1], variability modeling [5], or aspect-oriented approaches [10]. Typical strategies suitable for such adaptations are: service substi-

tution (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 [19]; defining at design time a composition goal or utility function, which is then achieved or optimized by dynamic service re-composition at run-time based on a specific environment and available services [20]. 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 parametrized 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 run-time, 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 run-time. 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.).

7 Discussion

In this section we illustrate how the design for adaptation activities may be performed in different scenarios represented in Section 3. In particular, given the specific characteristics of the scenario, we show the factors triggering adaptation, the types of adaptation realization suitable for the scenarios, and the appropriate adaptation strategies. Table 2 summarizes all these aspects with reference to the considered case studies.

Automotive scenario. The critical changes that require adaptation in this scenario range from instance-specific problems (e.g., failures and SLA violations, specific customers)

Table 2. Adaptation characteristics of the scenarios in Section 3.

Case Study	Properties	Adaptation Trigger	Design Approach	Adaptation Strategy
Automotive	Relatively stable context and potential partners, long-running SBA instances, diversity of adaptation needs, decisions require human involvement	functional changes, failures, SLA violations, changes in business context	Dynamic adaptation (human-driven); built-in adaptation (for compensation or process customization)	Service substitution (selecting from ASN partners); SLA re-negotiation; re-composition by ad-hoc changes of process control/data; re-composition by selecting predefined process variants; compensation; trigger evolution
Wine	Fully dynamic and unreliable services, fully autonomous SBA	degrade of service (sensor) QoS	Abstraction-based adaptation	Re-composition of services (to optimize resource utility function), domain-specific actions (e.g., data transfer frequency changes)
Mobile user	Strong dependency from context and goals of users	context changes, changes of user activities	Abstraction-based (for context changes), dynamic	service substitution (by dynamic discovery); re-composition.

to the changes that affect the whole SBA (e.g., changes in business context). In the former case, it is possible to apply built-in adaptation and define the reactions at design-time by completely describing the corresponding strategy (compensation activities, process variants for different customers) or its parameters (for SLA re-negotiation, for service substitution). In the latter case, the specific adaptation strategy is chosen at run-time as the effect of changes on the system is not known. In the business settings, such a choice can hardly be automated; the business requirements and decisions require human involvement. In particular, business analysts make decisions on triggering evolution and/or on how the running process instances should be changed (i.e., ad-hoc process modifications).

Wine production scenario. In this scenario the dynamically changing state of the WSAN network requires continuous monitoring and optimization of the resource usage. For this purpose, the adaptation should (i) re-arrange the sensor network in order to minimize the sensor energy consumption, and (ii) optimize the modes, in which the sensors operate, e.g., by optimizing the data transfer frequency. While the latter solution requires domain-specific realization mechanisms, the former may be achieved by dynamic re-composition of services to minimize of the utility function corresponding to the energy consumption (see, e.g., [20]).

Mobile user application scenario. In this scenario the SBA should adapt (i) to the changes in its context, including also the context of the user (e.g., changing location and time, different user settings), and (ii) to the changes in the user activities and plans. The former may be very dynamic: different services may apply for different locations or user settings. Abstraction-based adaptation is indeed required in this case: the abstract activities (e.g., buy a ticket for local transportation) are defined at design-time and made concrete at run-time using service concretization techniques (e.g., buy a ticket using online service of Milan public transport company). This requires appropriate description of the abstract activities as well as the services, e.g., using Semantic Web languages. Besides, it is necessary to take into account that the services may have different proto-

cols, or fulfill certain activity only partially: automated composition mechanisms may be applied in this case.

To deal with the changes in user activities and plans, it is necessary to understand the impact of those changes on the current processes and state of the SBA (i.e., perform dynamic adaptation). Depending on the outcome, different adaptations may apply (e.g., compensate or re-compose some tasks, fail). Differently from automotive scenario where the business analysts are high-level domain experts, these decisions can not be delegated to the mobile user, as they may have no expertise on the low-level technical details of the SBA. Therefore, it is necessary to design such decision mechanisms that at run-time may reason on the specific situation in order to reveal an appropriate strategy and its parameters (e.g., to decide whether re-composition may be done, to derive concrete composition goal and the corresponding composition, etc.). In [9], in particular, such decision mechanism relies on the analysis of personal information of the user (e.g., context, agenda, tickets and reservations, etc.).

8 Conclusions

This paper proposes a design method for SBAs that targets the adaptation requirements of those applications and aims at overcoming the fragmentation in current approaches for SBA adaptation. The approach is based on a novel life-cycle that considers adaptation as a first class concern and that covers the different facets of adaptation, both during the design phase and at run-time. After describing this life-cycle, the paper digs into the problem of design for adaptation, i.e., of identifying the design principles that are suitable to enable adaptation; in particular, different adaptation approaches — built-in, abstraction-based, dynamic — are identified and their links with adaptation triggers and adaptation strategies are analyzed.

Admittedly, this paper is just a first step towards our ultimate goal of defining a holistic design method for adaptable SBAs. Still, the effectiveness of such principles and guidelines is witnessed by their capability to capture the key aspects of adaptation in the different, heterogeneous real-world scenarios considered in this paper.

Our future roadmap includes a refinement of guidelines and principles presented in this paper, their formalization into patterns, and the definition of more precise criteria to decide on the patterns that are most appropriate for a given adaptation need. We also intend to work on the development of mechanisms and tools supporting the methodology, building on top of the actions and artifacts identified in Figure 2. Finally, we intend to work on a stronger empirical evaluation of the proposed methodology, by applying it to the real-world scenarios we already exploited in this paper.

Acknowledgements

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

References

1. L. Baresi, S. Guinea, and L. Pasquale. Self-healing bpel processes with dynamo and the jboss rule engine. In *ESSPE '07: International workshop on Engineering of software services for pervasive environments*, pages 11–20, New York, NY, USA, 2007. ACM.
2. G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In H.-G. Beyer and U.-M. O'Reilly, editors, *GECCO*, pages 1069–1075. ACM, 2005.
3. M. Colombo, E. D. Nitto, and M. Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In *ICSOC*, pages 191–202, 2006.
4. A. Erradi, P. Maheshwari, and V. Tosic. Policy-driven middleware for self-adaptation of web services compositions. In *Middleware '06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 62–80, New York, NY, USA, 2006. Springer-Verlag New York, Inc.
5. A. Hallerbach, T. Bauer, and M. Reichert. Managing Process Variants in the Process Lifecycle. In *10th Int'l Conf. on Enterprise Information Systems (ICEIS'08)*, 2008.
6. Hewlett-Packard Development Company, http://h71028.www7.hp.com/enterprise/cache/484275-0-0-225-12 1.html. *SOA Governance*.
7. O. Is. International Standard - ISO/IEC 14764 IEEE Std 14764-2006. pages 1–46, 2006.
8. D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. P. Buchmann. Extending bpel for run time adaptability. In *EDOC*, pages 15–26, 2005.
9. R. Kazhamiakin, P. Bertoli, M. Paolucci, M. Pistore, and M. Wagner. Having Services "YourWay!": Towards User-Centric Composition of Mobile Services. In *Future Internet Symposium*, 2008.
10. W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati. An Aspect-Oriented Framework for Service Adaptation. In *International Conference on Service Oriented Computing*. Springer-Verlag, December 2006.
11. D. Linner, H. Pfeffer, I. Radusch, and S. Steglich. Biology as inspiration towards a novel service life-cycle. In *ATC*, pages 94–102, 2007.
12. S. Microsystems. Soa rq methodology. www.sun.com/products/soa/soa_methodology.pdf.
13. K. Mittal. Service oriented unified process. http://www.kunalmittal.com/html/soup.html.
14. H. Pfeffer, D. Linner, and S. Steglich. Dynamic adaptation of workflow based service compositions. In *ICIC '08: Proceedings of the 4th international conference on Intelligent Computing*, pages 763–774, Berlin, Heidelberg, 2008. Springer-Verlag.
15. E. Rukzio, S. Siorpaes, O. Falke, and H. Hussmann. Policy based adaptive services for mobile commerce.
16. G. Spanoudakis, A. Zisman, and A. Kozlenkov. A service discovery framework for service centric systems. *Services Computing, IEEE International Conference on*, 1:251–259, 2005.
17. M. Trainotti, M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, F. Barbon, P. Bertoli, and P. Traverso. Astro: Supporting composition and execution of web services. In *ICSOC*, pages 495–501, 2005.
18. K. Verma, K. Gomadam, A. P. Sheth, J. A. Miller, and Z. Wu. The meteor-s approach for configuring and executing dynamic web processes. Technical report, University of Georgia, Athens, June 2005.
19. K. Verma, K. Gomadam, A. P. Sheth, J. A. Miller, and Z. Wu. The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. In *Technical report*, 2005.
20. L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.

Using a Lifecycle Model for Developing and Executing Adaptable Interactive Distributed Applications

D. Meiländer[1], S. Gorlatch[1], C. Cappiello[2], V. Mazza[2], R. Kazhamiakin[3], and A. Bucchiarone[3]

[1] University of Muenster (Germany), [2] Politecnico di Milano (Italy), [3] Fondazione Bruno Kessler (Italy)

Abstract. We describe a case study on using the generic Lifecycle Model developed in the S-Cube project for a novel class of Real-time Online Interactive Applications (ROIA), which include distributed simulations (e.g. massively-multiplayer online games), e-learning and training. We describe how the Lifecycle Model supports application development by addressing the specific challenges of ROIA: a large number of concurrent users connected to a single application instance, frequent real-time user interactions, enforcement of Quality of Service (QoS) parameters, adaptivity to changing loads, and competition-oriented interaction between users, other actors, and services. We describe the implementation aspects of the application development and adaptation using the RTF (Real-Time Framework) middleware, and report experimental results for a sample online game application.

Keywords: Service-Oriented Architecture, Service Engineering, Real-Time Online Interactive Applications, Adaptation, Real-Time Framework (RTF)

1 Introduction

Service-oriented applications are developed for constantly changing environments with the expectation that they will evolve over time. Several service-oriented system engineering (SOSE) methodologies have been proposed aiming at providing methods and (sometimes) tools for researchers and practitioners to engineer service-oriented systems. SOSE methodologies are more complex than traditional software engineering (TSE) methodologies: the additional complexity results mainly from open world assumptions, co-existence of many stakeholders with conflicting requirements and the demand for adaptable systems. A number of service lifecycle models have been proposed by both industry and academia. However, none of the proposed models has either reached a sufficient level of maturity or been able to fully express the aspects peculiar to SOSE. The S-Cube project [1] combines existing techniques and methodologies from TSE and SOSE to improve the process through which service based applications will be developed.

This paper describes an industrial-strength case study for the S-Cube Lifecycle Model in the emerging and challenging area of Real-time Online Interactive Applications (ROIA) which include such popular and socially important applications

as multi-player online computer games, high-performance simulations, e-learning, etc. ROIA pose several new challenges: thousands of users connect simultaneously to one application instance and frequently interact with each other, system must adapt to changing loads, maintaining QoS requirements, etc. Within the European edutain@grid project [2], a service-oriented architecture including a novel RTF (Real-Time Framework) middleware was implemented which focuses on the main challenges of ROIA.

The paper studies how the application of the S-Cube Lifecycle Model to the applications on top of the edutain@grid architecture enables the designer to identify suitable adaptation mechanisms and design patterns for the challenging area of ROIA.

We briefly introduce the S-Cube Lifecycle Model for SOSE in Section 2, followed by a description of the edutain@grid architecture and RTF in Section 3. We describe the application of the Lifecycle Model on the case study scenario from edutain@grid in Section 4 and report experimental results of a sample ROIA application based on RTF in Section 5. Related work is finally discussed in Section 6.

2 The Lifecycle Model for Service-Oriented Applications

The S-Cube Lifecycle Model for adaptable Service Based Applications (SBAs) (see Figure 1) comprises two main cycles: (i) a design-time iteration cycle that leads to the explicit re-design of the application in order to adapt it to new needs (i.e., *evolution*), and (ii) an *adaptation* cycle at runtime that is used when the adaptation needs are addressed on-the-fly. The two cycles coexist and support each other during the lifetime of an application [5].

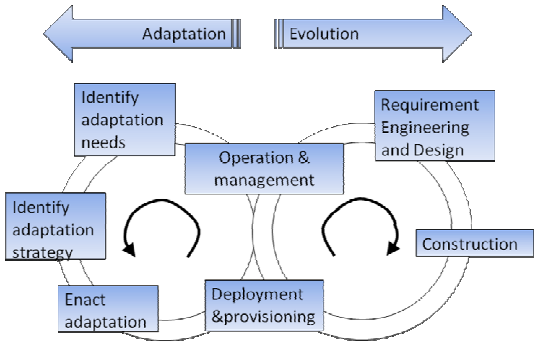

Figure 1: The S-Cube Lifecycle Model.

The development of an SBA starts with the former cycle that inherits some common aspects from the traditional software lifecycle but is modified in order to deal with specific adaptation issues. In case of ROIA, already in the *Requirements Engineering & Design* phase it is necessary to identify monitoring and adaptation requirements and methods to guarantee high update rates.

The *Construction* phase of an SBA is often performed in the form of a service composition. The construction can be manual (service integrator defines an executable process composed of concrete and abstract services), model-driven (service orchestration models are generated by abstract models) or automated (starting from service models, the executable SBA is automatically generated). For ROIA, it is necessary to implement suitable parallelization, adaptation and scalability mechanisms. Then, after the *Deployment and Provisioning* phase in which the application is introduced to customers, the *Operation and Management* phase relies on the monitoring activities that use the monitored properties to derive the status of the application and detect changes in the context or in the system that require adaptation or evolution. Starting from this phase, ROIA developers can decide to execute the right-hand side of the lifecycle if an evolution of the application is required (redesigning the application offline, making it temporarily unavailable to customers), or otherwise the ROIA is managed online by enacting adaptation actions at runtime (executing the left-hand side). E.g., an iteration of the evolution cycle may become necessary if the application is facing new attacking mechanisms by fraudulent users or in case of changing user requirements, since there may be a need to define additional sensors and monitors, as well as to change adaptation strategies.

In the adaptation cycle, it is important to define the adaptation needs that can be caused by: changes in the functional and non-functional aspects (e.g., unreliable hoster resources cannot preserve QoS requirements) or changes of the context in which the application is running (e.g., increasing user numbers in the evening hours creating peak load). In the domain of ROIA, adaptation mechanisms need to be proactive and transparent to users in order to adapt the application during runtime.

3 A Service-Oriented Architecture for ROIA

In this section, we describe the specific features of Real-time Online Interactive Applications (ROIA) and express their major design and execution aspects in the context of the S-Cube Lifecycle Model. ROIA pose many new challenges for SOSE including: large number of concurrent users connecting to a single application instance, frequent real-time user interactions, enforcement of precise QoS parameters, adaptivity to changing loads, and competition-oriented interaction between users and services.

Within the edutain@grid project, a distributed service-oriented architecture (see Figure 2) was implemented that is based on the interaction of four actors [10]: (1) *End-user* accesses ROIA sessions through graphical clients, typically purchased as a DVD; (2) *Scheduler* negotiates on behalf of the end-user appropriate ROIA sessions based on the QoS requirements (e.g. connection latency); (3) *Hoster* is an organisation that provides a computational and network infrastructure for running ROIA servers; (4) *Resource broker* provides a mechanism for application Schedulers and Hosters (and possibly other actors) to find each other in a large-scale environment and negotiate QoS relationships.

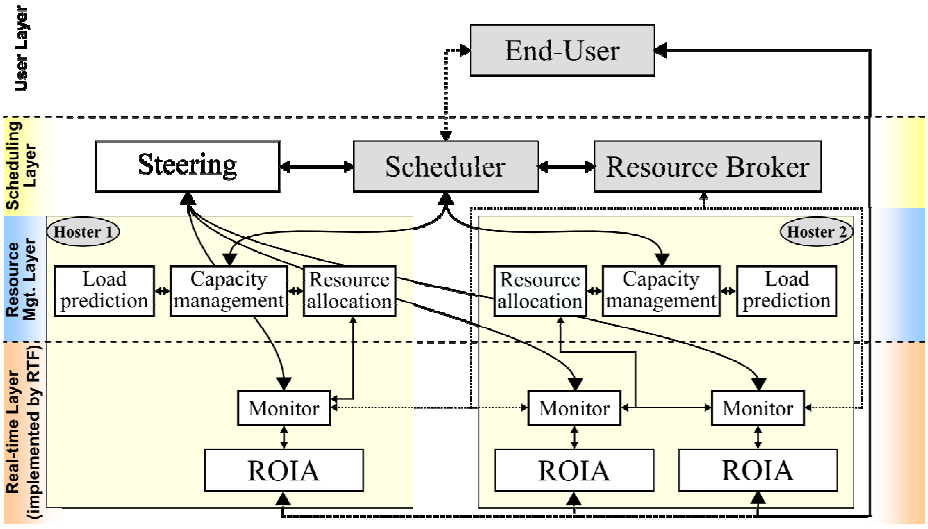

Figure 2: The edutain@grid architecture for ROIA.

The service-oriented edutain@grid architecture encompasses stateful web services, ontologies, business and accounting models, as well as a flexible and generic communication API for ROIA. In the following, we use the application area of online gaming to identify suitable services for the implementation of the adaptation cycle in the S-Cube Lifecycle Model (Figure 1).

Scheduling Service. The web service-based Scheduler receives from the user QoS requirements which can be performance-related (e.g., maximum allowed latency, minimum bandwidth) or application-specific (e.g., game genre, minimum number of users) and negotiates with existing hosters. The result is a contract, called *service level agreement (SLA)*, for the interaction of the end-user with the application. The mapping of users to game servers, as well as the allocation of Hoster resources to game servers, takes place as a distributed negotiation between the Scheduler and Hosters. The result is a performance contract that the Scheduler offers to the end-user and which does not necessarily match the original QoS request. The user can accept the contract and connect to the proposed session, or reject it.

Runtime Steering Service. During the game session, situations may occur which affect the performance, such that the negotiated SLAs cannot be maintained. Typical perturbing factors include external load on shared resources, or overloaded servers due to an unexpected concentration of users in "hot spots". The steering component is a web service which interacts at runtime with the monitoring service of each Hoster for preserving the negotiated QoS parameters for the duration of the session. A violation of a QoS parameter triggers appropriate adaptive steering or rescheduling using the API of the real-time layer. Thereby the Runtime Steering Service contributes to the "Identify Adaptation Needs" and "Identify Adaptation Strategy" phases of the adaptation circle of the Lifecycle Model.

Resource Allocation Service. Typically, each Hoster owns a Resource Allocation service responsible for allocating local resources to the clients. This web service receives from the Scheduler connection requests formulated in terms of QoS requirements, such as minimum latency or maximum bandwidth, and returns a positive answer if it can accomodate them. Online games are characterized by a large number of users that share the same application instance and interact across different game servers. The atomic resource allocation units for users are, therefore, no longer coarse-grain processes, but rather fine-grained threads and memory objects, which are more sensitive to external perturbations.

Capacity Planning Service. The load of a game session depends heavily on internal events, e.g. interactions of avatars. Also external events may occur, such as the user fluctuation over the day or week [9]. Hence, it is crucial for a Hoster to anticipate the future game load. The Capacity Planning web service predicts future load of Hoster resources by employing neural networks [10] and thereby contributes to the "Identify Adaptation Needs" and "Identify Adaptation Strategy" phases of the Lifecycle Model.

The real-time layer of edutain@grid is implemented by the *Real-Time Framework (RTF)* [3] that distributes game state calculations among the participating servers. Instead of using web services and SOAP-encoded communication messages which are not suitable for fast real-time communication, RTF uses TCP/IP sockets internally. RTF provides efficient parallelisation and adaptation concepts and implements suitable monitoring capabilities, which we explain in the following.

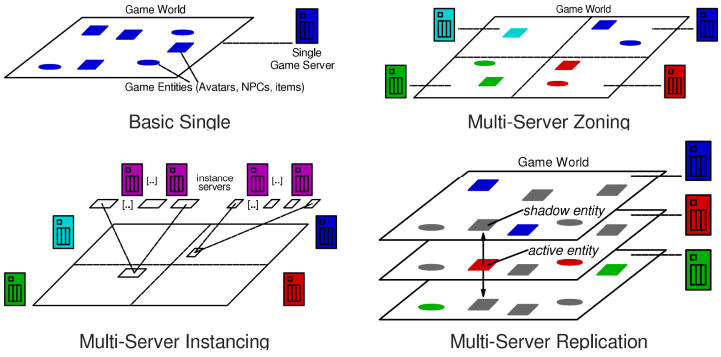

Figure 3: Adaptation strategies of RTF via (re)distribution.

Real-Time Adaptation Service. To enable the service scalability to a high number and density of users, RTF distributes game sessions adaptively, based on several adaptation strategies illustrated in Figure 3. *Zoning* [8] distributes the game world into disjoint zones, each zone being assigned to one server. *Instancing* uses multiple, independently processed copies of highly frequented subareas of the game world, each copy processed by one server: if an avatar moves into one frequented subarea, it is assigned to one of the available copies. Different copies are not synchronized since users in disjoint copies cannot interact with each other. *Replication* [13] assigns

multiple servers to a zone with a high load; the responsibility of computing in that zone is divided equally among the servers. RTF's adaptation strategies are used in the "Enact Adaptation" phase of the Lifecycle Model as explained in Section 4.

Monitoring Service observes the QoS parameters negotiated by the Hoster. Several monitoring parameters are summarized in profiles which support monitoring of low-level QoS parameters, as well as of game-related metrics (like entity positions, messages sent or received, or end-user activity information) crucial for an adequate game experience of the users. RTF's monitoring services are used by the steering and capacity planning services.

4 Using the Lifecycle Model for Developing ROIA

In this section, we demonstrate how the Lifecycle Model of Section 2 is applied to ROIA applications by exploiting the edutain@grid architecture of Section 3. The lifecycle is based on various adaptation- and monitoring-specific actions and related design artifacts. The main aspects for designing an adaptive application are: the *application requirements*, the *adaptation strategies*, and the *adaptation triggers*.

Our analysis of ROIA applications has identified the application requirements shown in Table 1. Besides the functional and non-functional application requirements described in Section 3, it is also important to identify the requirements to be considered for the design of adaptation actions. In the case of ROIA, the mechanisms for monitoring and adaptation should be non-intrusive, i.e., take place in parallel with the application execution, and users not aware of changes inside the application. Since the considered system must guarantee the QoS requirements, proactive adaptation should be supported in order to prevent QoS violations.

Functional Requirements	- Correct execution
Non Functional Requirements	*Client related requirements* - Short response time (< 100 ms) - Frequent interactions between users
	Game session-related requirements - Resources use for a variable # of users - Suitable parallelization concepts - High number of concurrent users in a single application instance - High update rate (5-100 updates/s)
Adaptation Requirements	- Transparency of the adaptation - Instance-specific adaptation during runtime - Need for proactive adaptation - Autonomy (self-adaptability) - Efficiency of adaptation actions

Table 1: Application requirements for ROIA.

The application requirements drive the selection of the adaptation strategies. According to the Lifecycle Model, we identify the following five adaptation strategies for ROIA applications on top of the edutain@grid architecture:

(1) *User migration*: users are migrated transparently from an overloaded server to an underutilized server which is replicating the same zone.

(2) *Zoning*: new zones are added during runtime and assigned to additional game servers. Zoning provides the best scalability of all adaptation strategies, but is not transparent to users (since the geography of the virtual world is changed), so it is generally used for high numbers of users that cannot participate otherwise.

(3) *Replication*: new game servers are added transparently during runtime in order to increase computation power for highly frequented zones. When replicating a zone, a number of users are migrated to the replica and initiate workload redistribution. However, replication implies an additional inter-server communication and thus, its scalability is limited. To address the demand for autonomic and self-adaptable applications, the number of active replicas for a particular zone is monitored to decide whether activating additional replicas is feasible.

(4) *Instancing*: creates a copy of a zone which is processed by a different server than the original zone. Since users in different copies of the same zone cannot interact with each other, replication is generally preferred to instancing in order to support high interactivity between users. However, instancing as an adaptation strategy is useful if the overhead of replication would be too high.

(5) *QoS negotiation* with several distributed hosters which includes (i) adaptation of existing contracts, or (ii) negotiation of contracts with new hosters. Typical scenarios include the usage of stronger resources of the same hoster (QoS adaptation) or leasing cheaper resources from a new hoster. QoS adaptation can be an alternative to replication by allocating more powerful resources to serve more users. Since QoS adaptation needs a longer time, replication and instancing provide better scalability, but QoS adaptation can be used to overcome small peaks in resource shortage.

For designing adaptable SBAs, adaptation strategies must be related to adaptation triggers. Adaptation triggers and suitable trigger rules are defined considering all scenarios at runtime in which application requirements may be violated. For ROIA, adaptation triggers are related to changes in Service Quality, in the computational context and in users' requirements, unexpected increment of the users' accesses, and specific users' needs. In the edutain@grid architecture, proactive adaptation is planned on the basis of predicted values from the capacity planning service. For example, load balancing may anticipate increasing user numbers in the evening hours and request appropriate resources. Then, the adaptation trigger (predicted increase in user numbers) is related to the change in the context (i.e., time period).

Adaptation triggers provide to the application designers the variables to be monitored at runtime and thereby drive the design of the monitoring mechanism. In ROIA, monitored properties include CPU/memory load on hoster resources, the number of concurrent users in an application instance, bandwidth capacity etc.

By considering the application requirements and the adaptation strategies, we distinguish the following scenarios for triggering adaptation in ROIA applications:

(1) *Change in Quality of Service*: QoS violations which were not expected and predicted, e.g., caused by unreliable hoster resources. In this scenario, user migration, replication or instancing is used for adaptation in order to overcome performance bottlenecks. We decide which of the three adaptation mechanisms to use depending on the amount of free resources and number of active replicas: in order to minimize costs for the application provider, migrating users to underloaded resources is

preferred if the additional load can be compensated by running resources; otherwise replication is preferred to instancing in order to support a high level of interactivity between users; if activating additional replicas implies too high communication overhead, instancing is used.

(2) *Change in computational context*: a change in the costs of calculating the application state, e.g., caused by increasingly or decreasingly frequent interactions between users. To prevent QoS violations, one of the adaptation strategies is used: user migration, replication or instancing (depending on free resources and number of replicas).

(3) *Change in business context*: a change in user preferences which was not predicted in advance, e.g., many new users connecting to the application. In this scenario, user migration, replication or instancing (depending on the amount of free resources and number of replicas) are used for adaptation.

(4) *Prediction results*: The capacity planning service gathers information about the users' preferences and triggers adaptation proactively and autonomously. Depending on the predicted number of additional users, either QoS negotiation, replication or adding new zones at runtime are used for adaptation since predicted adaptations can be planned ahead. If the maximum number of replicas is already reached, then instancing can be chosen.

Table 2 shows how the described adaptation triggers and adaptation strategies are linked together, and provides examples for trigger rules and monitored values:

Adaptation Trigger	**Monitored variable**	**Adaptation Trigger rule**	**Adaptation Strategy**
Change in Quality of Service	response time, throughput, resource usage, average packet loss, connection latency, update rate, service availability	update rate < 25 updates/s	user migration, replication or instancing
Change in comput. context	CPU and memory load, incoming/outgoing bandwidth	CPU load > 90%	user migration, replication or instancing
Change in business context	number of concurrent users, number of requests per application	number of concurrent users > Σ user capability of application servers	user migration, replication or instancing
Prediction values from capacity planning service	number of users/hour, number of requests per application	predicted users > current users + △ (threshold)	QoS negotiation, replication or instancing/zoning

Table 2: Relationship between Adaptation Triggers and Adaptation Strategies.

We observe that for each adaptation trigger, various adaptation strategies may be suitable. A ROIA application is realized in the *Construction* phase of the Lifecycle Model in order to include all the monitoring features and technical infrastructure needed for proactive adaptation. The design patterns and distribution concepts offered by RTF support developers designing and implementing efficient ROIA; application

development on top of RTF was described in [18]. The need for proactive adaptation has a strong impact on the design and realization of the monitoring mechanisms. Since ROIA applications are very dynamic, not all failures/critical situations can be identified at design time. Therefore, there is a need to continuously update the knowledge about the behavior of the system.

After the construction phase, the application is deployed, i.e. it is ready to be executed and managed. During the *Operation and Management* phase, the application is running and all the previously designed adaptation triggers are monitored to detect changes in the context or in the system that could require adaptation or evolution. Starting from this phase, the application is managed online by enacting adaptation actions during runtime; if evolution is required, the right-hand side of the lifecycle is executed again conforming to the specific implementation issues for ROIA.

5 State of Implementation and Experimental Results

In order to test the effectiveness of system design using the Lifecycle Model, we perform some experiments with the central part of the edutain@grid architecture, responsible for real-time services for ROIA. In particular, we study the suitability of the zoning and replication adaptation strategies for ROIA. We have implemented the Real-Time Framework (RTF) [18] as a C++ based library, since C++ is currently the language of choice for ROIA applications in industry. Using RTF as a development framework and runtime middleware, we developed several applications, some of them jointly with industrial partners, from three areas: online computer games [16], e-learning systems [11], and crowd simulation [19]. We use one such application for our experiments.

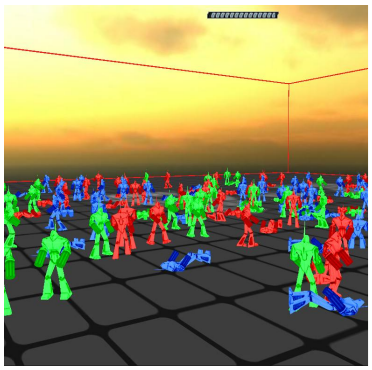

Fig 4: Screenshot of RTFDemo (avatars managed by different servers have different colours).

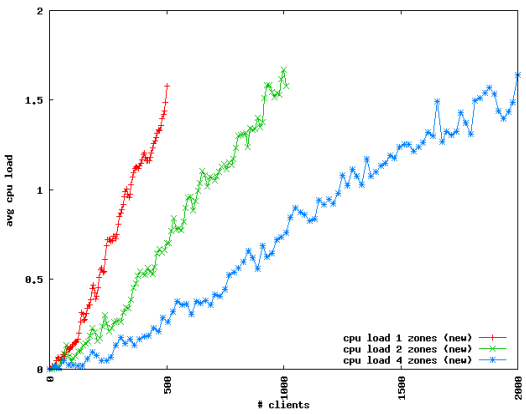

Figure 5: CPU load for zoning adaptation.

The *RTFDemo* application is an industrial-strength, fast-paced online game that takes place in a zoned 3D world and is built on top of RTF. RTFDemo is a

representative of the first-person shooter game category, the most demanding class of online games requiring a very high state update rate, interactivity and frequent message exchange. A user participates in RTFDemo by controlling a robot avatar in a 3D virtual world and interacting with avatars controlled by other users. The characteristics of RTFDemo correspond to a modern commercial online game: the game state is updated 25 times per second, both seamless migration between different zones and interactions across zone borders are supported. Fig 4 shows a screenshot of the RTFDemo game.

We conducted experiments that test the zoning and replication adaptation strategies implemented in RTFDemo. We use a pool of homogeneous PCs with 2.66 GHz, CoreDuo 2 CPUs, and 4 GB RAM in a LAN. A static setup of zones and servers was started with multiple computer-controlled clients (bots) that continuously sent inputs to their servers. The average CPU utilization was measured on the servers as the metric to be evaluated. Clients were allowed to move between zones and thereby generate higher load on some of the servers.

In the experiments with adaptation by zoning, each zone was assigned to a different server. Figure 5 shows the measured number of players that were able to participate fluently in the game for one, two and four zones, respectively. We observe that the zoning adaptation scales almost linearly.

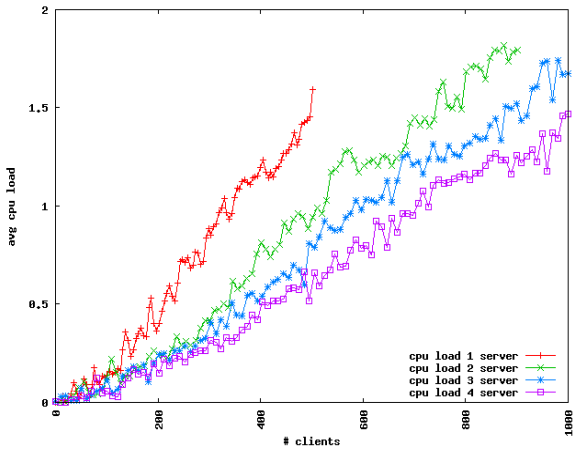

Figure 6: CPU load for replication adaptation.

Our second set of experiments aims at the replication adaptation strategy: we replicate the computation of a single zone on up to four servers. Figure 6 shows the measured results for the CPU load. One server is able to serve up to 450 clients at a CPU load of 120% (each core has a load below 100%), which is similar to the results of zoning. But if more servers are added to the processing of a large zone, the client numbers can be increased from the previous limit of 450 clients to up to 1000 clients in a four-server setup. This shows that the replication adaptation strategy allows games to provide a higher level of interactivity.

6 Related Work

Initial SOSE methodologies directly derived from traditional software engineering (TSE) methodologies, but over time, the need for dealing with new challenges led to the development of specific approaches (e.g., [5],[14],[15]).

Several approaches (e.g., [4],[6]) deal with the design and realization of service-based applications. Most of them are not flexible since they base the execution of service-based applications on static rules that trigger the execution of a pre-defined adaptation action only when some specific and known events happen. In fact, applications could address adaptation issues by using built-in adaptation or dynamic adaptation, but it is suitable when all the adaptation configurations are known at priori. In such kind of applications, specification is performed by extending standard notations or using ECA-like rules [7] or aspect-oriented approaches [12]. The main disadvantage of such adaptation approach is the impossibility to react to unforeseen events. This paper studies challenging cases in which the adaptation needs are dynamic and not all the system characteristics are known a priori, i.e. the adaptation actions cannot be completely defined at design time. Some approaches address this issue by proposing an abstraction-based adaptation through which at design time, the adaptation strategies are defined while the concrete mechanism is defined only at run-time. For example, in [17] the design of the application is based on the abstract definition of service. Only at run-time the services are effectively selected on the basis of the situation and context in which the execution is required. The S-cube lifecycle enables both the built-in and the abstraction adaptation, but it also addresses the dynamic adaptation, for which it is possible to provide mechanisms that select and instantiate adaptation strategies depending on a specific trigger and situation [5].

7 Conclusions and Future Work

The main contribution of this paper is the industrial-strength case study in which we applied the generic Lifecycle Model for developing adaptive service-based applications to the novel, emerging class of ROIA (Real-Time Interactive Applications). We demonstrated how the specific, challenging features of ROIA can be met during the evolution and the adaptation cycles of the application development. In particular, we identified the adaptation triggers and adaptation strategies that help to develop high-quality, scalable ROIA applications. We show the effetiveness of the proposed lifecycle for ROIA applications in which proactive adaptation is mandatory.

Our experimental results confirm the feasibility of applying the S-Cube Lifecycle Model and identified adaptation mechanisms to developing demanding applications. In addition to the RTFDemo game described in the paper, we have implemented several industrial applications using RTF: a multi-server port of the commercial action game Quake 3 [16], the 3D game *Hunter* developed by the game company *Darkworks,* and the remote e-learning framework *edutain@grid Virtual Classroom* [11] developed by the environmental consulting company *BMT Cordah Ltd.*

Our future work will include implementing the adaptation triggers identified in Section 4, developing design patterns for the described adaptation scenarios, as well as further joint experiments with industrial partners. The adaptation capabilities of our system can be naturally complemented by the advantages of Cloud Computing offering virtually instantly available, pay-per-use compute resources. We plan to further enhance our system in order to provide ROIA on Clouds.

References

[1] The S-Cube project, http://www.s-cube-network.eu/, 2010.

[2] The edutain@grid project, http://www.edutaingrid.eu, 2009.

[3] The Real-Time Framework (RTF), http://pvs.uni-muenster.de, 2010.

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

[5] A. Bucchiarone, C. Cappiello, E. Di Nitto, R. Kazhamiakin, V. Mazza and M. Pistore, Design for Adaptation of Service-Based Applications: Main Issues and Requirements. In *Proc. of the Fifth International Workshop on Engineering Service-Oriented Applications: Supporting Software Service Development Lifecycles (WESOA)*, 2009.

[6] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *GECCO '05: Proc. of the 2005 conference on Genetic and evolutionary computation*, 2005.

[7] M. Colombo, E. Di Nitto, and M. Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In *ICSOC*, 2006.

[8] W. Cai, P. Xavier, S. J. Turner, and B.-S. Lee. A scalable architecture for supporting interactive games on the internet. In *16th Workshop on Parallel and Distributed Simulation*, pages 60-67, Washington, DC, USA, 2002. IEEE Computer Society.

[9] W.-C. Feng, D. Brandt, and D. Saha. A long-term study of a popular MMORPG. In *NetGames'07*, ACM Press., 2007.

[10] S. Gorlatch, F. Glinka, A. Ploss, J. Müller-Iden, R. Prodan, V. Nae, and T. Fahringer. Enhancing Grids for Massively Multiplayer Online Computer Games. In *Euro-Par 2008 - Parallel Processing*, volume 5168 of *Lecture Notes in Computer Science*, 2008.

[11] S. Gorlatch, F. Glinka, H. Roreger, and C. Rawlings, Distributed e-Learning using the RTF middleware. In *Proc. of the 2nd Annual Forum on e-Learning Excellence*, 2009.

[12] W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati. An Aspect-Oriented Framework for Service Adaptation. In *ICSOC'06*. Springer-Verlag, December 2006.

[13] J. Müller-Iden and S. Gorlatch. Rokkatan: Scaling an RTS game design to the massively multiplayer realm. In *Computers in Entertainment*, 4(3):11, 2006.

[14] M. P. Papazoglou, and W. J. Van Den Heuvel. Service-oriented design and development methodology. In *International Journal of Web Engineering and Technology*, 2(4):412-442, 2006.

[15] B. Pernici. Methodologies for Design of Service-Based Systems. In *International Perspective of information Systems Engineering*, edited by S. Nurcan, C. Senesi, C. Souveyet, J. Ralyté. Springer, 2010.

[16] A. Ploss, S. Wichmann, F. Glinka, and S. Gorlatch, From a Single- to Multi-Server Online Game: A Quake 3 Case Study Using RTF. In *ACE'08: Proccedings of the 2008 Int. Conference on Advances in Computer Entertainment Technology*, 2008.

[17] K. Verma, K. Gomadam, A. P. Sheth, J. A. Miller, and Z. Wu. The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. In *Technical report*, 2005.

[18] F. Glinka, A. Ploss, S. Gorlatch, J. Müller-Iden. High-Level Development of Multiserver Online Games. In *Int. Journal of Computer Games Technology*, 2008(5):1-16, 2008.

[19] O. Scharf, S. Gorlatch, F. Blanke, C. Hemker, S. Westerheide, T. Priebs, C. Bartenhagen, A. Ploss, F. Glinka, and D. Meiländer. Scalable Distributed Simulation of Large Dense Crowds Using the Real-Time Framework (RTF). In *Euro-Par 2010 - Parallel Processing*, volume 6271 of *Lecture Notes in Computer Science*, 2010.