Grant Agreement N° 215483

Title:	*Coordinated design knowledge models for software engineering and service-based computing*
Authors:	*CITY, FBK, Lero-UL, POLIMI, Tilburg, UCBL, UniDue, VUA*
Editors:	*Ita Richardson, Stephen Lane (Lero-UL)*
Reviewers:	*Manuel Carro (UPM)*
	Fabrizio Silvestri (CNR)
Identifier:	*Deliverable # CD-JRA-1.1.4*
Type:	*Deliverable*
Version:	*1.0*
Date:	*31 Aug 2009*
Status:	*Final*
Class:	*External*

Management Summary

In this deliverable, we discuss the need for the S-Cube life-cycle and the development of enhancements to support its implementation. We introduce the area of Service-Oriented Systems Engineering (SOSE) and discuss on how it is different from Traditional Software Engineering (TSE), while still recognising that both of these disciplines have important inputs to make to the development of Service-Oriented Systems. We progress with further development of the S-Cube life-cycle focusing on requirements, design and adaptation. From the requirements perspective, we investigate context-aware requirements discovery and specification, exploring whether existing models can be applied to improve requirements specification. From the design perspective, we suggest design principles and guidelines that are suitable to enable adaptation. From the adaptation perspective, we investigate SOSE and TSE to present practices for adaptation. Finally, we propose a unified formal model for dealing with the effects of iterative and localized changes between any two interacting service consumers and providers.

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	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
VU Amsterdam	Netherlands

Published S-CUBE documents

These documents are all available from the project website located at http://www.s-cube-network.eu/results/deliverables

Glossary

ASD - Abstract Service Definition
BPEL - Business Process Execution Language
BPMN - Business Process Modeling Notation
COC- Cross-Organisational Collaboration
ECA - Event Condition Action
KPI - Key performance Indicator
OS - Operating System
QoS - Quality of Service
SBA - Service Based Application
SeCSE - Service Centric Systems Engineering
SED - Service Engineering and Design
SLA - Service Level Agreement
SLDC - The Web Services Development Lifecycle
SOA - Service-Oriented Architecture
SOSE - Service-Oriented Software Engineering
TSE - Traditional Software Engineering
WSDL - Web Services Description Language
XML - eXtensible Markup Language

Contents

Chapter 1

Introduction

Service-oriented systems are developed for constantly changing environments with the expectation that they will evolve over time in a dynamic manner. The dynamism can vary in that the behaviour of services can range from answering simple requests to executing sophisticated processes [1] and service-based applications can change their behaviour in response to context changes by invoking new services in different combinations. Such systems are constructed by integrating heterogeneous services. These heterogeneous services are often developed using various programming languages and running on heterogeneous operating systems from a range of service providers [2].

Service-Oriented Systems Engineering (SOSE) is different from Traditional Software Engineering (TSE). This occurs because when building traditional applications using TSE, we have full control of the execution of the software components, even where they are customised off the shelf systems. However, when developing compositions, we are exploiting services that are run from third parties. This puts the development of the software outside the control of SOSE developers. It also moves the control of execution away from the owners of the service into the hands of others who are now using this service. This means that when we build service compositions we have to always remember that they will have to adapt to changes in the services they are exploiting. Quality, such as that expected from TSE applications, is still required, for example, performance, security, efficiency, usability, but, additionally, other quality characteristics become equally if not more important, for example, flexibility, dynamicity.

1.1 Background

Service-oriented computing enables software engineers to develop applications dynamically by seamlessly integrating software services and processes across communication networks and heterogeneous platforms [1]. Web and software services are operations that users access via the internet through a well-defined interface independent of where the service is executed [3]. Many Service-oriented System engineering (SOSE) methodologies have been proposed in both academia and industry aiming at providing approaches, methods and (sometimes) tools for researchers and practitioners to engineer service-oriented systems (see for instance [4]). SOSE methodologies are more complex than traditional software engineering (TSE) methodologies, having to deal with new challenges while keeping the principles of TSE. The additional complexity results mainly from open world assumptions, co-existence of many stakeholders with conflicting requirements and the demand of adaptable systems [5]. To improve the understandability of a methodology and its guidance, software development process models (describing what activities a development process consists of and how they should be performed) have been often used since they visualize the development process proposed by the methodology. However traditional software process modeling techniques are no longer directly applicable or adaptable in SOSE [6], and a number of service life cycle models have been proposed by both industry and academia (e.g. [7] [6] [4] [8]). However, none of the proposed models has either reached a sufficient level of maturity or been able to fully express the aspects that are peculiar to SOSE.

In the S-Cube project, the Services Engineering and Design research which is carried out allows the project team to build further on TSE and SOSE research. In doing this, we can combine techniques and methodologies from both disciplines to strengthen the process through which service systems will be developed. This is evident in the research presented in this deliverable. For example, in [9] we integrate adaptation from SOSE with maintenance from TSE to extend the process from the development life-cycle. In the first stages of this project [1], S-Cube partners developed a life-cycle for services which consists of two cycles (see Figure 1.1). In cycle 1, the evolution cycle, shown on the right hand side of the diagram, service-oriented systems are developed following the traditional stages of requirements engineering, design, construction and deployment, while also focusing on quality assurance. As stated earlier, services-oriented development must allow for the systems to be dynamic, with the developer considering continuous adaptation of the system. The life-cycle shown here highlights not only the typical design-time iteration cycle (Evolution), but it introduces a new iteration cycle at runtime, Adaptation. This is undertaken in all the cases in which the adaptation needs are addressed. In addition, throughout the process within the complete life cycle, there must also be a focus on operation management and quality.

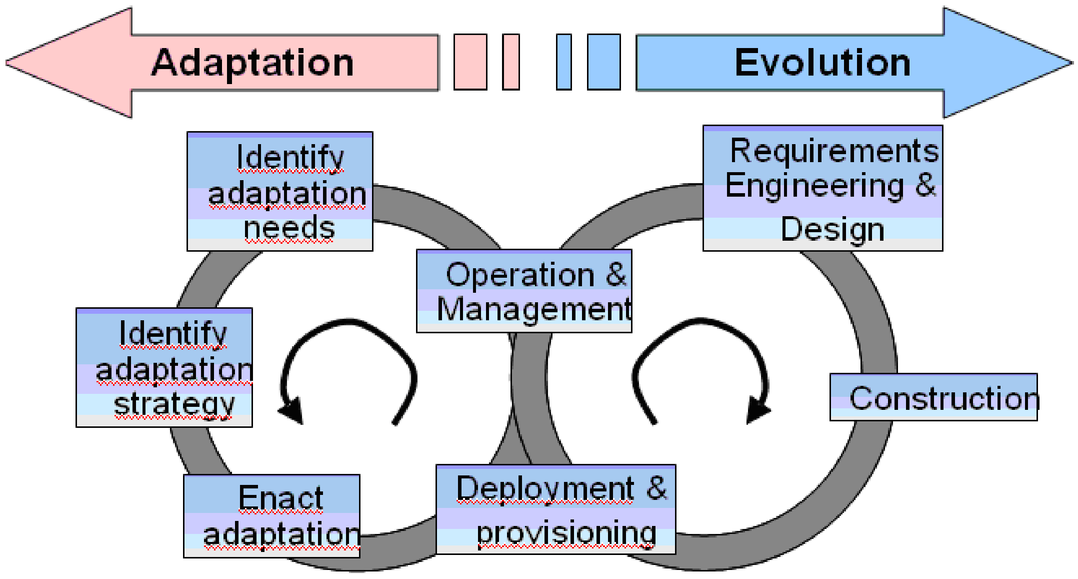

Adaptation
Evolution
Identify adaptation needs
Requirements Engineering & Design
Operation & Management
Identify adaptation strategy
Construction
Enact adaptation
Deployment & provisioning

Figure 1.1: The Life-Cycle of Adaptable Service Based Applications.

The need for 2 cycles arises because the nature of dynamic services development is to ensure that the system evolves over time. The two cycles do not conflict with each other. Instead, they cohexist and support each other during the lifetime of the application. With services development, adaptation of the system is important to, or even essential in, system development and design. It is therefore incumbent upon us, in this era of services development, to ensure that software engineering processes are modified to allow the principles of software services development to be enacted successfully during the product life-cycle.

In a further development of the life-cycle shown in Figure 1.2 [10], we have identified the various adaptation- and monitoring-specific actions carried out throughout the life-cycle of the SBA, the main design artefacts that are exploited to perform adaptation, and the phases where they are used (dotted lines). In 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).

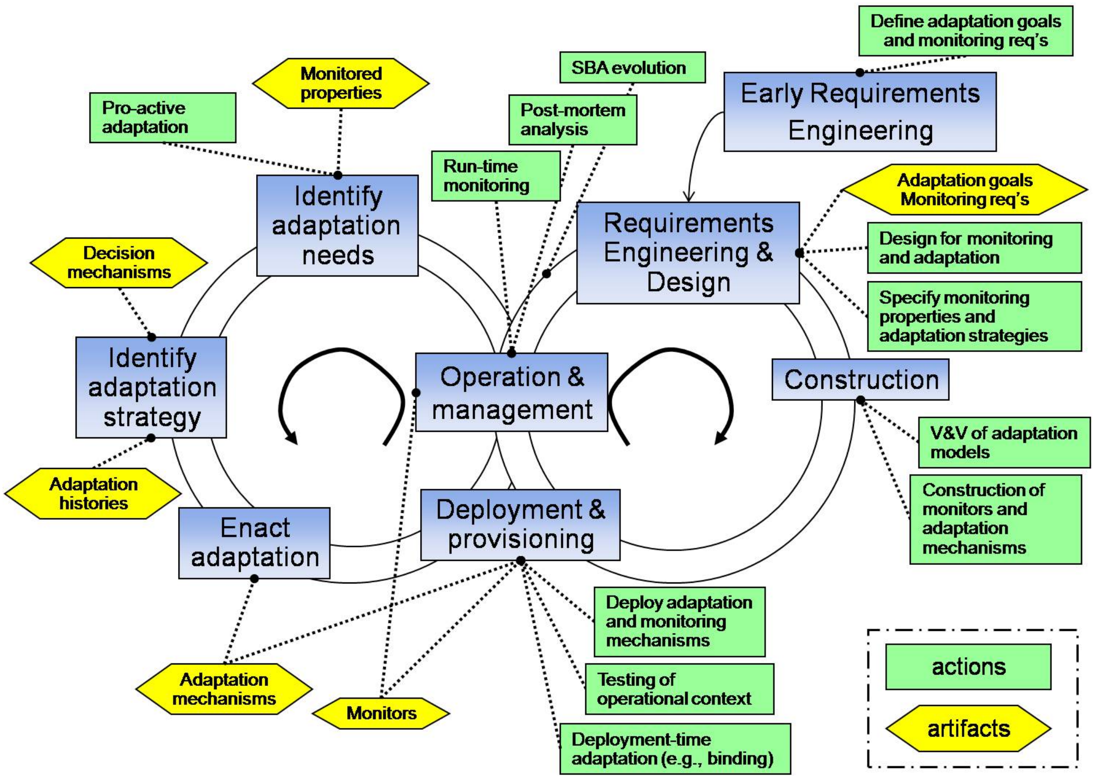

Define adaptation goals and monitoring req's
SBA evolution
Early Requirements Engineering
Monitored properties
Pro-active adaptation
Post-mortem analysis
Run-time monitoring
Identify adaptation needs
Adaptation goals Monitoring req's
Requirements Engineering & Design
Design for monitoring and adaptation
Decision mechanisms
Specify monitoring properties and adaptation strategies
Identify adaptation strategy
Operation & management
Construction
V&V of adaptation models
Adaptation histories
Construction of monitors and adaptation mechanisms
Enact adaptation
Deployment & provisioning
Deploy adaptation and monitoring mechanisms
Testing of operational context
Adaptation mechanisms
Monitors
Deployment-time adaptation (e.g., binding)
actions
artifacts

Figure 1.2: Actions and Artefacts within S-Cube life-cycle

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.

1.2 Deliverable CD JRA 1.1.4

The purpose of this deliverable is to develop knowledge models that associate, compare and coordinate software engineering and service-oriented computing processes, methods, modelling and analysis techniques and tools available for use in these disciplines. In the S-Cube deliverable JRA-1.1.2 [11], we developed the S-Cube life-cycle. The extension of this knowledge model is to focus on particular phases of the life-cycle, and expand our knowledge at to what should occur within each of these phases. Additionally, in deliverable JRA-1.1.2 we presented the aspects of the project on which we would work during year 2 of S-Cube:

(1) Understand how to model and reason on the context from where adaptation/evolution requirements should come.

(2) Consolidate our understanding on adaptation and evolution of service-based applications and identify proper approaches for selecting adaptation requirements and for reason on them in order to handle possible conflicts and inconsistencies.

(3) Integrate the results achieved on the human-computer interaction aspects with the body of knowledge we have acquired on the life cycle for adaptable service-based applications.

The research presented here and in the corresponding papers [9][12][10][13][14] are focused on working towards these aspects, focusing specifically on the Requirements elicitation, Design for adaptation and Extended operation phases. With a view to (1) above, our first step was to develop and understanding of the existing TSE/SOSE models [13], this research is presented in Section 2.1.1. Requirements elicitation, focusing particularly on context is contained in [14] and in Section 2.2 and works towards aspect (3). To consolidate our understanding on adaptation and evolution of service-based applications (2), we carried out research on the topic of design for adaptation (see [10] and Section 2.3) and on the investigation of software practices within the adaptation cycle (see [9] and Section 2.4). The work by [12] also supports this research aspect by considering the formal modelling of services evolution through versioning and its impact to the services environment as an initiator of adaptation. This is described in Section 2.5 of this document. In Chapter 3 we summarise our research our research contribution. In Chapter 4, we discuss how the outcomes from this research contributes towards the S-Cube research vision, along with proposed future research that can be carried out within S-Cube.

Chapter 2

Development of Services Life-cycle Phases

2.1 Guiding the Service Engineering Process: The Importance of Service Aspects

The fundamental change in developing service-oriented systems as opposed to traditional software systems is that software is delivered as a service. As such, users pay for and use services instead of buying and owning software. Consequently, users do not have the control of services, which are controlled by service providers instead. These changes are reflected by three service aspects that we will discuss in detail in this section.

2.1.1 Introducing three service aspects

The relevance of cross-organizational collaboration The focus of SOSE shifts from the development of applications to services that are collaboratively developed by multiple Service-Oriented Architecture (SOA) roles [15, 16], such as service consumer, service provider, service broker. These roles are often distributed in multiple departments or organizations. In this case, interactions between development activities associated with multiple business roles demand for collaboration between multiple organizations. We call this type of collaboration, which crosses the boundaries of the domain of each role, cross-organizational collaboration.

Consequently, the processes of discovering, selecting, composing services require continuous interaction (or collaboration) [17] between the participating roles through the service development life cycle. This makes their relationship tighter but also demanding clearer governance and agreements.

Since collaboration between multiple roles becomes part of the SOSE process, it is of great importance to highlight this collaboration in a SOSE process model. When the collaboration becomes explicit and clear, the need for corresponding agreements or contracts becomes evident. Consequently, appropriate governance can be applied.

Increased importance of the identification of stakeholders We ideally define a stakeholder as a role in a SOSE methodology. Please notice that in the reality, a role can played by different stakeholders (i.e., persons, groups or organizations) and vice versa, the same stakeholder can play different roles. For the purpose of our analysis of service aspects, however, such differentiation can be easily ignored.

Since cross-organizational collaboration becomes more critical in SOSE, the importance of clearly identifying stakeholders increases accordingly. Furthermore, the decision on whether a role should be identified as a separate stakeholder in a SOSE process model depends on what type of interactions the model intends to represent. For instance, if a SOSE development approach intends to emphasize, or elaborate, on how monitoring is provided, the service monitor has to be selected as a stakeholder.

The importance of the identification of stakeholders in SOSE process models also lies in the fact that stakeholders in the SOSE development process do not always assume the same roles as in TSE. For instance, in general a software developer is responsible for implementing software applications, while in SOSE a service developer could be responsible also for composing existing services, depending on specific methodologies. Without specifically associating SOSE activities with stakeholders, one is not able to visualize the corresponding responsibilities as one would do in TSE.

In summary, identifying stakeholders with the appropriate level of detail in a SOSE development approach facilitates the establishment of a corresponding SOSE process model describing associated activities and their interactions at an appropriate level of abstraction.

The need for increased effort at run-/change time In TSE, the main goal is to develop high quality applications that meet the requirements of the end users. Consequently, most of the effort is dedicated to design (collecting requirements, design, and implementation) and change time (maintenance). Runtime activities are hardly addressed if not in specific domains. Furthermore, change time activities are often performed off line (either with or without execution interruption).

Different from TSE, the main goal of SOSE is to deliver high quality, agile, and robust services that are able to meet the ever-changing business requirements. Consequently, much more development effort is shifting from design time to run-/change time. For instance, components identification is a typical design time activity aiming tat analyzing and selecting existing components for constructing software systems. Service identification (or discovery), a SOSE activity equivalent to components identification, is often encouraged to be performed at runtime in order to find the most suitable services dynamically for composing agile SBAs.

Most existing SOSE process models do fail in emphasizing this shift. By explicitly modeling the two stages, a process model can visualize the amount of activities shifted to run-/change time, hence providing useful inputs to resource allocation.

2.1.2 An initial experiment with service aspects

With the aim of gaining insight in the extent to which the modeling of the three service aspects improves the guidance of a SOSE development approach, we modeled them in a concrete and practical context. In particular, we looked for a development approach specifically focusing on SOSE and explicitly considering adaptation and evolution as important aspects. As highlighted also in Deliverable CD-JRA-1.1.2 [11], most of the existing approaches do not take these two elements into account. Therefore, we selected and modeled for our experiment an approach that, even though not fully mature and tested, offers methods and tools to explicitly fulfill the requirements above. This is the methodology developed within the SeCSE project [18].

By focusing on service aspects, our main objective is to discuss *what* has to be modeled rather than *how* to model. For illustration purposes, we use BPMN[1] as process modeling notation to be used to communicate a methodology to its users. This is expressive enough to represent the various inter-dependencies and multiple stakeholders involved in the SeCSE development process.

The resulting model from the SeCSE project is given in 2.1, where the service aspects are explicitly modeled in terms of graphical patterns. These graphical patterns are a set of pre-defined notations that we used to capture the service aspects. More specifically, we modeled the stakeholders in terms of the *stakeholders pattern* to highlight the increased importance of the identification of stakeholders. Their associated activities and inter-dependencies are modeled in terms of the *cross-organizational collaboration pattern* to highlight the interactions that might go beyond the organizational boundaries. The increased effort at run-/change time becomes obvious when they are separated from the design time effort using the 2-stage pattern.

In this section, for each service aspect we first explain its associated graphical pattern in a SOSE process model in general; and then we discuss how the SeCSE methodology addresses the service aspect by observing the SeCSE process model against the graphical pattern. In this way, we show how each service aspect both emphasizes the characteristics of the SeCSE methodology itself and facilitates better SOSE guidance.

[1]www.omg.org/spec/BPMN/

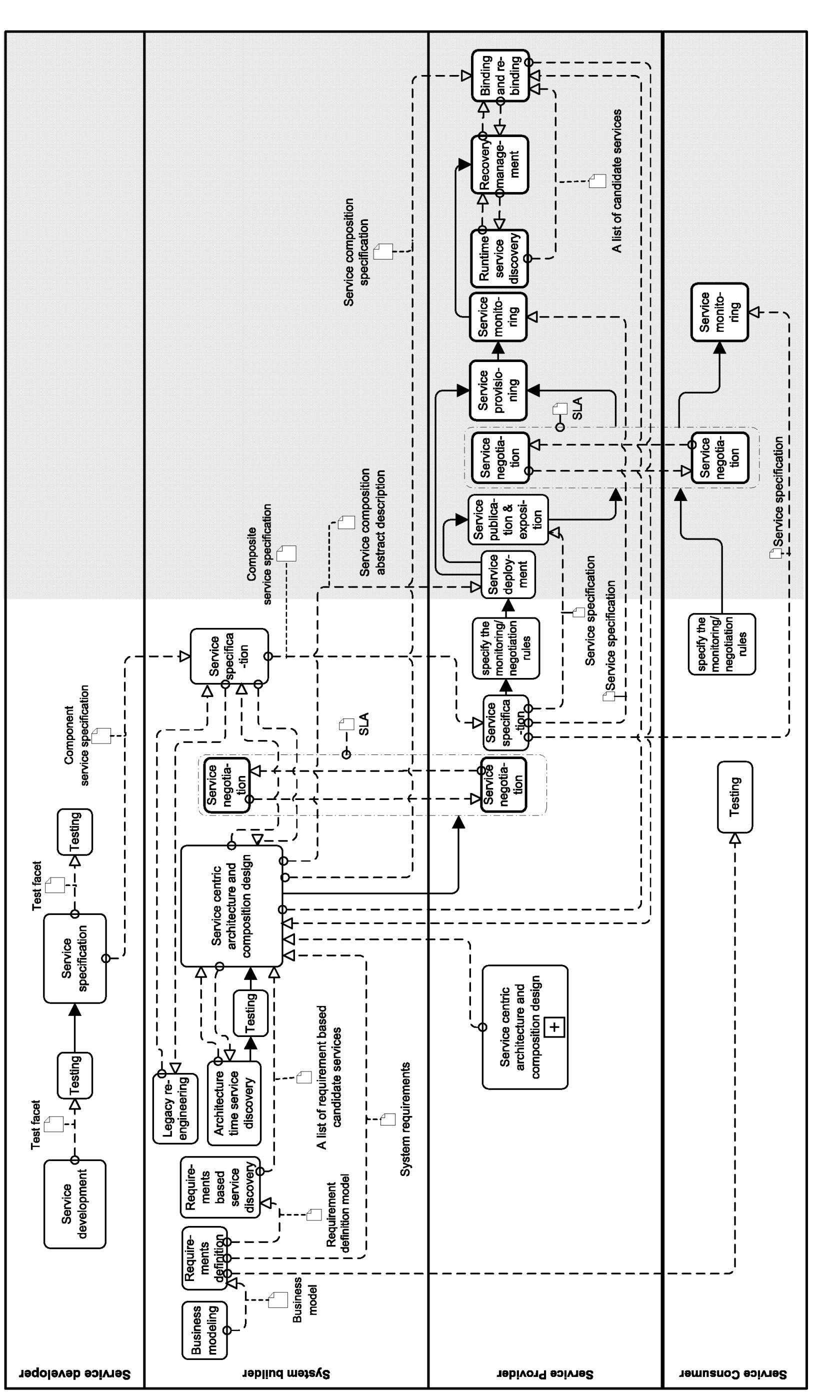

Design time
Runtime / change time
Service developer
System builder
Service Provider
Service Consumer
Service development
Test facet
Testing
Service specification
Component service specification
Business modeling
Business model
Require-ments definition
Requirement definition model
Require-ments based service discovery
Architecture time service discovery
Legacy re-engineering
A list of requirement based candidate services
System requirements
Service centric architecture and composition design
Service negotia-tion
SLA
Service specifica-tion
Composite service specification
Service composition abstract description
Service composition specification
specify the monitoring/ negotiation rules
Service deploy-ment
Service publica-tion & exposi-tion
Service provisio-ning
Service monito-ring
Runtime service discovery
Recovery manage-ment
Binding and re-binding
A list of candidate services
Service specification

Figure 2.1: The SOSE process model for the SeCSE methodology.

The relevance of cross-organizational collaboration

Fig. 2.2 graphically illustrates the cross-organizational collaboration (COC) service aspect. The left-hand side of the figure shows three collaboration types (COC patterns): the **peer activities group** models same activities carried out in parallel across multiple partner enterprises; the **main-sub activities** model the same activities carried out partially by one partner enterprise and completed by another; the **distributed activities** model inter-dependent activities carried out across multiple partner enterprises. These patterns are exemplified in the right-hand side of Fig. 2.2.

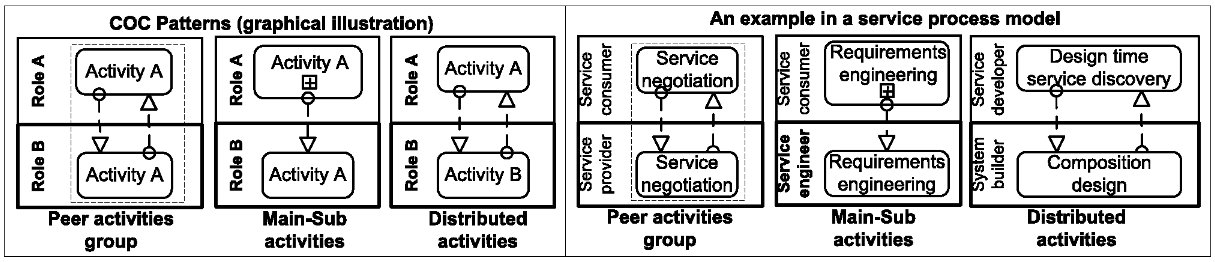


Figure 2.2: Service aspect: cross-organizational collaboration (COC).

By observing the SeCSE process model (shown in Fig. 2.1) against the three COC patterns defined in Fig. 2.2, attention is brought to specific types of cross-organizational collaboration.

- **Peer activities group:** `Service negotiation` occurs twice in the SeCSE development process. One is carried out by a service provider and a service consumer; another is carried out by a system builder and a service provider. By nature, each service negotiation must be performed in parallel by both its stakeholders as peers. The results of the collaborations (indicated by the data objects attached to the peers) are SLAs. Different from TSE where contracts are often established after software is built, SLAs in SOSE often precede final service products (service composition in the case of the SeCSE methodology). These SLAs are also potentially useful to other activities such as service monitoring.

- **Main-sub activities:** `Service centric architecture and composition design` are carried out by a system builder and service provider in a cooperative manner. A system builder has the main responsibility for this activity, whereas a service provider focuses only on a subset of its tasks.

  `Service specifications` are modeled as three activities with related service specification as data objects. They are carried out by a service developer, system builder and service provider independently but on related artifacts. In general, a service developer creates service specifications for a component service, which influences a composite service carried out by a system builder. The system builder has to make sure that the QoS characteristics defined in the specification of the component services are compatible with those of the composite service. When a service composition or a single service is deployed, the service provider may add information to the corresponding specification known at deployment time.

- **Distributed activities:** `Service centric architecture and composition design` is carried out by a system builder at design time and `binding and re-binding` is carried out by a service provider at run-/change time. Cross-organizational collaboration occurs when new substituting services are discovered at run time (e.g., due to a new requirement) and service composition needs to update its bindings to accommodate the change.

Only when the collaboration is explicitly captured, the stakeholders of service-oriented systems can gain insight on the impact between their own responsibilities and the others'. Each stakeholder has a clearer view on at what time ("when") which activity ("what") has to be carried out in cooperation with which stakeholder ("who") and in which manner ("how"). In the SOSE development process, external enterprises often continuously play important roles throughout the service life cycle. By looking for the cross-organizational collaboration patterns in a service process model, enterprises are brought to focus on the points needing strategic business agreements that should regulate such tight collaboration.

Increased importance of the identification of stakeholders

Fig. 2.3 graphically represents the stakeholders pattern (and an example), which makes the stakeholders in a SOSE process model explicit. The figure aims at highlighting the fact that each horizontal bar is associated to a role or stakeholder and therefore includes those activities that are executed by it. By observing the SeCSE process model (shown in Fig. 2.1) against this stakeholder pattern, we can see that the SeCSE methodology involves mainly four stakeholders, namely: `service developer, system builder, service provider` and `service consumer`). These stakeholders, potentially representing partner enterprises, play common SOA roles from the perspective of service implementation, integration provision, and consumption.

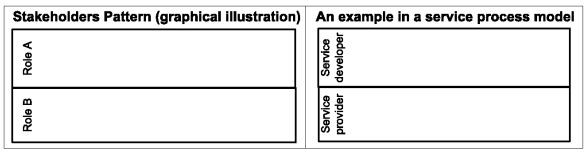

Stakeholders Pattern (graphical illustration)
Role A
Role B
An example in a service process model
Service developer
Service provider

Figure 2.3: Service aspect: increased importance of the identification of stakeholders.

Explicitly modeling the identified stakeholders improves the guidance of the SeCSE methodology as follows. Firstly, by placing the SOSE activities in the corresponding swimlanes, the SeCSE process model naturally shows the responsibilities and collaborations of and among stakeholders. This is especially crucial in SOSE where cross-organizational collaboration occurs in almost all activities. In this way, the business dependencies requiring contractual/SLA agreements are made explicit, and project managers can better plan the allocation of development activities based on the skills and responsibilities of the internal and external stakeholders.

Secondly, service composition centered characteristic of the SeCSE methodology is well captured by the SeCSE process model when identified stakeholders are explicitly modeled. Fig. 2.1 shows that most of the development activities are associated to the system builder and the service provider (stakeholders that carry out service composition activities). Furthermore, the model shows that the system builder and the service provider are tightly linked; the service developer and the service consumer are instead loosely linked. Due to focus of service composition, the service consumer in the SeCSE process model is considered as the consumer of composite services, rather than the consumer of component services. Therefore, the service consumer does not have direct interaction with the service developer, and the system builder must cooperate with the service provider in multiple activities. In this way, the SeCSE process model very well captures the fact that service composition is the main focus of the SeCSE methodology and consequently provides better guidance in that the stakeholders are able to gain Fig. 4. Service aspect: increased effort at run-/change time (2-Stages) better understanding of the focus of the (SeCSE) methodology.

The need for increased effort at run-/change time

Our approach of separating the design and run-/change time activities in a SOSE process model is presented in Fig 2.4, where run-/change time activities are placed in a gray box. The left-hand side of the figure shows the 2-stages pattern, exemplified in the right-hand side of the figure. In this example, it is visually evident that `service design` is carried out at design time; while `service discovery` is performed at run-/change time.

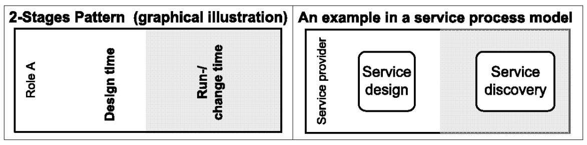


Figure 2.4: Service aspect: increased effort at run-/change time (2-Stages)

By observing the SeCSE process model (shown in Fig. 2.1) against the 2-stages pattern defined in Fig. 2.4, we are now able to easily distinguish the design time activities (falling in the left-hand side of the figure) from the run-/change time activities (in the shadowed area at the right-hand side of the figure). Consequently, the guidance for applying the SeCSE methodology is improved in that the process model shows its support for adaptation, service composition and facilitates critical project plan decisions.

Firstly, we notice that about one third of the development effort, simplistically computed in terms of number of activities, is dedicated to run-/change time. In particular, `runtime service discovery` and `service negotiation` are supported by the SeCSE methodology with the objective of increasing the adaptability and agility of resulting systems to meet on-the-fly requirements. Thereby, related activities such as runtime `service monitoring, recovery management`, and `binding and re-binding` are also in place.

Secondly, we notice that the development effort dedicated to run-/change time activities is not evenly distributed among the stakeholders in the SeCSE methodology. Instead, the service provider carries out most of the run-/change time activities; while the system builder and service developer do not perform run-/change time activities at all. We have discussed in Section 2.1.1 that the roles of system builder and service provider are extensively developed due to the service composition centered approach. The process model illustrates and emphasizes further the separation of design and run-/change time activities: the system builder focuses on the design of service compositions; while the service provider focuses on the provision of service compositions.

Thirdly, knowing which activities are executed at which stage is also crucial in SOSE. Project managers should be able to adjust project plans based on the criticality of the activities since runtime activities are directly related to executing services real-time and therefore more critical than design time activities. For instance, as shown in Fig. 2.1 service negotiation is supported by the SeCSE methodology at both design time and runtime. The difference is that at design time, service negotiation occurs between a system builder and a service provider for component services that are selected for service composition; at runtime it occurs between a service provider and a service consumer for a composite service that fulfills business requirements. This difference results in different levels of business commitment. At design time, the failure of reaching service level agreements or the failure of collaborating with a service provider does not have huge business impact on the system builder; the system builder can always decide to look for an alternative service. However, at runtime if the composite service fails to execute or does not reach the quality it promises, the system builder faces risks to loose its customer and even its business market. Here, the business commitment is much higher than at design time. Being aware of this difference, a project manager is able to decide which actions to take for activities with various levels of criticality.

2.2 Using Services to Discover Requirements for Service-Based Applications

2.2.1 Introduction

Context-aware computing has already a long tradition, so that the need arises to introduce context-aware features in SBAs as well. Whilst the development of context-aware systems has been the subject of considerable research, the challenges of discovering and specifying requirements on such systems have not been sufficiently addressed. Therefore, this Section reports research dedicated to requirements elicitation for context-aware SBAs, and is given in detail in [14].

The basis for this research are scenarios that have been shown to be an effective technique for eliciting requirements [19, 20, 21]. In particular, we extend the existing ART-SCENE scenario walkthrough method and its related tools to cover relevant context-aware information. Scenario walkthroughs are structured workshops in which stakeholders step through each scenario event in turn, discovering requirements to handle each event in turn. ART-SCENE is a method and associated tools to support a systematic scenario walkthrough. The method is based on a normal course scenario, which is automatically extended with alternative course events during the walkthrough. These alternative course events trigger the elicitation of alternative and exceptional scenarios, which complement the initial set of requirements. Consequently, the major challenge is to elicit relevant context factors, which might impact the behaviour and/or the quality of the SBA. These context factors in turn resemble a set of events, which may trigger the elicitation of context-dependent alternative course scenarios.

The elicitation of context factors and the related alternative course scenarios are based on codified context knowledge and its inclusion in the ART-SCENE method. To achieve this integration, the context knowledge was organised according to different types (e. g. knowledge about the location, knowledge about the user, etc.). Classes of abnormal behaviour, which describe an event that triggers an alternative course scenario, were assigned to these context types. These classes of abnormal behaviour allow us to describe events of alternative course scenarios in sufficient detail without overloading the stakeholder when s/he performs the scenario walkthrough.

2.2.2 Extending ART-SCENE

We extended the ART-SCENE environment with 3 types of knowledge:

(1) new classes of abnormal behaviour and state specific to service-based applications;

(2) new rules to generate alternative course events in scenarios that instantiate these new classes of abnormal behaviour and state, and;

(3) guidelines for specifying requirements in response to these generated alternative course events. The new classes were generated using an exploratory three-stage process based on the types of context reported in [22].

In the first stage we walked through one normal course scenario that described a driver's use of an in-car navigation system. The context types were used to generate instances of alternative courses in the scenario that could generate or refine requirements on the in-car navigation system. In the second stage we generalized and aggregated the new alternative course classes to remove duplicates. In the third stage we validated the new alternative course events by walking through a new normal course scenario, describing remote management of home appliances, and instantiating the same alternative course classes to each scenario event in it. We retained alternative course class that could be instantiated to generate alternative course events and which generated new and valid requirements in the new domain. The alternative courses were then validated against the 40 classes of abnormal behaviour in the original database, removing duplicates that were associated with event timings.

The result was a prototype extension of the ART-SCENE took with which to explore the usefulness of the extensions during formative evaluations. The next sections describe the extensions in more detail.

2.2.3 Introduced New Classes of Abnormal Behaviour

A sample of the new classes of abnormal behaviour classes grouped by their context type, are reported in Table 2.1. A complete table is available in [14]. We generated 8 new classes and sub-classes of abnormal behaviour for the computing context, divided into behaviours and states related to services and devices. We generated 17 new classes and sub-classes for the user context, 9 for the physical context, and 2 for the time context. The lower number of time context classes reflected the overlap with the original ART-SCENE knowledge already encoded in its database. Therefore, some of the identified classes of abnormal behaviour relevant for context-aware SBAs were removed to avoid duplicates.‘

ID	**Abnormal behaviours and states**
Computing Context Abnormal behaviours and states	
SE1	What if there is an exception with the service?
SE1.1	What if the service is not available?
SE1.2	What if the time to complete the service request is too long?
SE1.3	What if the service is busy?
User Context Abnormal behaviours and states	
CO1	What if there is an exception with the consumer?
CO1.1	What if the consumer is busy or not available?
CO1.8	What if the consumer is working collaboratively?
CO1.16	What if the consumer has a different nationality?
Physical Context Abnormal behaviours and states	
LO1	What if there is an exception about the location?
LO1.1	What if there are legal consequences of the (current or future) location?
LO1.4	What if the (current or future) location prevents communication?
LO1.7	What if the (current or future) direction that the consumer is pointing changes?
Time Context Abnormal behaviours and states	
TI1	What if there is an exception relating to the time?
TI1.1	What if this activity is happen at different time, e.g. time of day?

Table 2.1: Classes of abnormal behaviour and state by context type

For each of these 36 new classes of abnormal behaviour and state, we sought to specify one rule that would generate candidate alternative course events by instantiating it for each normal course event. No rules were identified and generated for classes CO1.14, CO1.15 and CO1.16 because we were unable to associate ART-SCENE action and agent types to alternative course events related to privacy, religion and nationality. The remaining 33 rules were specified in a condition-action form. For example rule Context 8 specifies that:

IF the event is start of action of type (system or communication) undertaken by agent type (device)

THEN generate alternative course instantiating class DE1.3 (What if the device has already undertaken other behaviour?)

This rule was specified to generate alternative course events that describe the situation when a device is busy because of its internal processing or external communication with other devices or human agents.

2.2.4 Specifying More Precise Requirements

Generating possible alternative courses specific to context-aware SBAs can increase the coverage of the requirements process but, on its own, it is unlikely to increase the precision of the resulting requirements. To achieve such a higher precision, we need to differentiate the way in which the requirements are elicited. Our solution was to provide different requirements authoring guidelines associated with different classes of abnormal behaviour and state. We aligned different requirements types with different classes of abnormal behaviour and state, then prescribed different authoring guidelines based on the requirement type, as supported in the VOLERE [23] and Planguage [24] approaches. This alignment was achieved using the 3 broader requirements classes of Quality-related requirements, Requirements linked to user characteristics and context-specific requirements. The type of requirement to specify for different classes of abnormal behaviour and state is given in Table 2 of [25].

To guide requirements analysts to specify the different types of requirements we offer a different template with tailored guidelines for each requirement type, similar to the established VOLERE [23] and Planguage [24] techniques. For example an instance of the class of abnormal behaviour SE1.2—the time to complete the service request is too long—might be mitigated or avoided by a user-type requirement that specifies different permitted service consumer behaviours and/or a performance-type requirement on the system specified using metrics such as throughput and latency [26]. Table 2.2 reports two possible requirements—one of each type—that can be specified in response to an alternative course event that instantiates class SE1.2. The first specifies behaviour that the service consumer shall be able to undertake: if the service is taking too long to respond to the request, the service consumer shall be able to select alternative services that achieve the same result. The second specifies a required performance characteristic of the service-based application: the application shall be able to support different communication networks that make the application more reliable and, therefore, improve performance times that will can increase the likelihood of the alternative course event not occurring.

Abnormal Behaviour Class	What if the time to complete the service request is too long? (SE1.2)
Requirements Type	Performance
Service consumer The service consumer can undertake actions to avoid or overcome the request-response delay	The passenger shall be able to select between services based on the predicted performances of relevant and available services
Measured by Latency The time between the service invocation and the service reply.	The system shall be able to use a UMTS or WiFi connection.

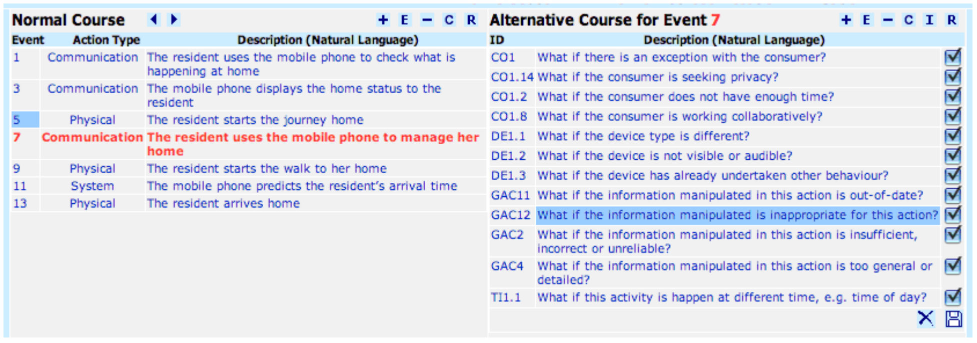
Table 2.2: Example template to elicit Performance requirements

To enable a validation, we are currently extending ART-SCENE with the complete set of requirement-type templates and guidelines listed which will be linked to its VOLERE shell for documenting requirements.

2.2.5 Example Scenarios

We used the extended version of ART-SCENE to generate new scenarios. One of these, which describes how a resident remotely manages their home when returning from work, is shown in the ART-SCENE web client in Figure 3. The normal course on the left-hand side describes events that are start of actions such as the mobile phone displays the home status to the resident and the resident uses the mobile phone to manage her home. Most of the alternative courses on the right-hand side are instances of the new abnormal classes of behaviour of state generated for the action the resident uses the mobile phone to manage her home. An example of a consumer alternative course event is what is the consumer does not have enough time? One device alternative course event is what if the device type is different? One time alternative event is what if this activity happens at different time, e. g. time of day? Figure 2.5 also shows alternative course events starting with code GAC generated from the original ART-SCENE classes, e. g. what if the information manipulated in this action is out-of-date?

Requirements generated from the new alternative course events might introduce functionality to guide the resident to manage her home when she has time, prompt the resident to undertake different tasks during the day, and support the functionality on different types of mobile device. Furthermore, the specification of these requirements might include preconditions related to different user, device, location and time contexts.



Normal Course

Event	Action Type	Description (Natural Language)
1	Communication	The resident uses the mobile phone to check what is happening at home
3	Communication	The mobile phone displays the home status to the resident
5	Physical	The resident starts the journey home
7	**Communication**	**The resident uses the mobile phone to manage her home**
9	Physical	The resident starts the walk to her home
11	System	The mobile phone predicts the resident's arrival time
13	Physical	The resident arrives home

Alternative Course for Event 7

ID	Description (Natural Language)	
CO1	What if there is an exception with the consumer?	✓
CO1.14	What if the consumer is seeking privacy?	✓
CO1.2	What if the consumer does not have enough time?	✓
CO1.8	What if the consumer is working collaboratively?	✓
DE1.1	What if the device type is different?	✓
DE1.2	What if the device is not visible or audible?	✓
DE1.3	What if the device has already undertaken other behaviour?	✓
GAC11	What if the information manipulated in this action is out-of-date?	✓
GAC12	What if the information manipulated is inappropriate for this action?	✓
GAC2	What if the information manipulated in this action is insufficient, incorrect or unreliable?	✓
GAC4	What if the information manipulated in this action is too general or detailed?	✓
TI1.1	What if this activity is happen at different time, e.g. time of day?	✓

Figure 2.5: Segment of a remote home control scenario generated with the extended version of ART-SCENE

2.2.6 A First Formative Evaluation

To evaluate the ART-SCENE extension we have published the automatically-generated scenario depicted in Figure 2.5 on the ART-SCENE web-site, and asked experienced requirements analysts to comment remotely on the perceived usefulness and usability of the alternative course events for discovering and specifying requirements using a structured questionnaire. Results are pending: we plan to report them as part of the S-Cube IA-3.2 workpackage. S-Cube builds on the ESPRIT IV CREWS Long-term research project, which led to the ART-SCENE software environment.

2.2.7 Further Evaluation

The paper written on this research [14] reports results that demonstrate the integration of context-aware computing concepts into existing requirements engineering techniques and tools. Scenarios, which are one technique for representing and reasoning about context in requirements processes, are extended with alternative course events that instantiate different classes of context. As such context information described in both the scenario normal course and course events is presented in different combinations to describe different possible contexts for which requirements on service-based applications might be specified. The successful development of ART-SCENE demonstrates that the extension is tractable.

The next stage of the reported research is to undertake a more complete formative evaluation of the extended ART-SCENE environment. We will investigate whether the extended environment will result in the specification of a larger number of requirements, requirements that are more complete, and requirements that are more precise. Evaluation data will be collected from a series of face-to-face scenario workshops including 6–8 stakeholders each at different European sites, following the ART-SCENE workshop process reported in [27]. These sites include an Austrian IT systems integration organization, a German systems consultancy specialising in support for the German federal states, and a small consortium of experienced requirements analysts based in London.

The evaluation in be in 2 major stages. In the first a facilitator will run each workshop by walking the stakeholders through 2 versions of a scenario: alternative course events generated from the standard ART-SCENE classes of abnormal behaviour and state, and alternative course events generated from the news classes extended for service-based applications. A scribe will document each requirement's description and rationale generated by the stakeholders in ART-SCENE. After each workshop the stakeholders will be divided into 2 groups of 3 or 4 stakeholders each. The first will use ART-SCENE's requirements specification template to complete the specification of a subset of the requirements generated from the original alternative course events, whilst the second will use the extended requirements specification template described in section 3.3 to specify requirements from the new alternative course events. As a result the workshops will generate a corpus of requirements specifications that will be reviewed in the second stage of the evaluation.

During the second stage an independent requirements expert will analyze and score each of the stakeholder requirements for its completeness and precision blind to their source. We will then compare the requirements' scores to investigate whether requirements generated from the extended ART-SCENE environment generate a larger number of requirements, requirements that are more complete, and requirements that are more precise. We hope to report results from this evaluation in the near future.

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

2.3.1 Design for Adaptation

Modern service-based applications have to be adaptable to unforeseen changes in the functionality offered by component services, to their unavailability, to decreasing performances, to the current context of use as well as to specific requirements and needs of the specific users. Most of the existing approaches address this issue by hard coding in the infrastructure supporting the execution of service-based applications a limited number of adaptation strategies that are triggered only when some specific and known events happen. Adaptation of service-based applications works properly only in the case the application is designed to be adaptable. Furthermore, different application scenarios and domains may expose different characteristics and therefore pose different requirements on the adaptation activities. As it is shown in the reference paper [10] with the help of automotive supply-chain, wine production system, and mobile application scenarios, these characteristics may include the properties of the application context, dynamicity of involved services, human involvement, etc. Furthermore, service-based applications have to face very different adaptation needs, including customization, recovery and repair, self-optimization and context-driven adaptation.

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

The following subsection explain different elements of the design for adaptation process, as well as the guidelines for the identification, selection, and realization of the adaptation strategies and their relation to the adaptation triggers.

2.3.2 Adaptation and evolution in service-based applications

Figure 2.6 (see the appendix for the details about the way UML diagrams are used through the document) shows the main ingredients that are needed for building and operating Adaptable Service-Based Applications (Adaptable SBAs or simply SBAs from now on). An Adaptable Service Based Application also poses new requirements in terms of monitoring and adaptation aspects. *Monitoring Requirements* concern the need for detecting (part of) those situations that may trigger the need for adapting an SBA. *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. The *Context* includes users and execution properties.

Generally, adaptation requires some temporary modification permitting to respond to changes in the requirements and/or in the application context or to faulty situations. 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*.

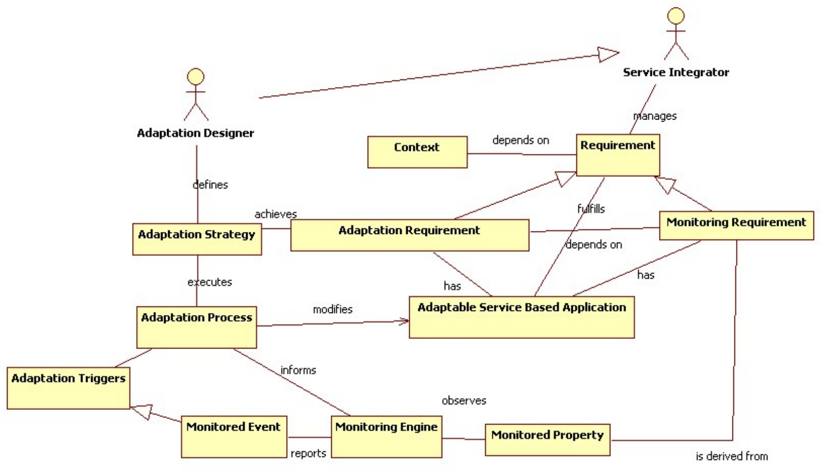


Figure 2.6: Main concepts related to the definition and operation of Adaptable SBAs.

2.3.3 Design for adaptation: main ingredients

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

Adaptation strategies

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

Adaptation triggers

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

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

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

Table 2.3: Relationships between triggers and adaptation strategies

possibility of the application still to accomplish its current task). Depending on these parameters different strategies may apply (Table 2.3).

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

- **Built-in adaptation**. If possible adaptation needs and possible adaptation configurations are fixed and known a priori, it is possible to completely specify them at design time. The focus is on specifying situations, under which adaptation is triggered, and the concrete actions to be performed.

The specification may be performed by extending the standard SBA notations (e.g., BPEL) with the adaptation-specific tools using ECA-like (event-condition-action) rules , variability modeling, or aspect-oriented approaches. Typical strategies suitable for such adaptations are: service substitution (by selection from predefined list of options), re-execution, compensation, re-composition (by using predefined variants), fail.

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

Reference paper [10] demonstrates the application of these considerations and guidelines to the motivating scenarios (Automotive, Wine-Production, Mobile).

2.4 Adaptation of Services: A Maintenance Process?

2.4.1 Adaptation of Services

There have been many approaches proposed for development of SBAs, however few of them address the adaptability phases in the S-Cube reference life-cycle in Figure 1.1. In order to address this problem we have mined adaptation-specific practices from two complementary sources identifying practices which can be carried out within the Adaptation phase of the S-Cube life-cycle [9]. For the purposes of this research we have focused both on service oriented architecture (SOA) approaches, and on software engineering processes. While the S-Cube life-cycle shows the Adaptation Cycle and breaks it into three constituent parts - Identify adaptation needs, Identify adaptation strategy, Enact adaptation - we need to define what practices should occur within each of these. In the Phase I development, we identified the adaptation-related activities within existing service oriented architecture (SOA) approaches, thus providing practices which should be included in the S-Cube Adaptation life-cycle as defined by SOA. During

Phase II development, we carried out a gap-analysis comparing SOA adaptation and the software engineering maintenance process, focusing particularly on ISO/IEC 14764. ISO/IEC 14764 is a software engineering standard which outlines a detailed software maintenance process. Through this analysis, we identified that there are other practices, which, for effective implementation of Adaptation of Services, should be included within the more detailed level of the life-cycle.

2.4.2 Phase I: Adaptation-Related Practices

Our first step in the identification of the Adaptation practices proposed was to examine current SOA approaches as well as adaptation aspects from existing S-Cube deliverables (CD-JRA-1.1.2, CD-JRA-1.2.1, CD-JRA-1.2.2 [28]), analysing these to identify where they would fit into the S-Cube life-cycle model.

S-Cube Deliverables

One of the objectives is to identify adaptation requirements that are either raised by the human beings involved in the execution of service-based applications or generated by the technological environment in which the system is running. As soon as the need for adaptation is identified, the methods and strategies for adaptation should be decided in the step of *Identify adaptation strategy*. The actual adaptation execution takes place in the step of *Enact adaptation*. Another objective is to decide if and when to take these requirement into consideration in that some requirements might be conflict with each other. With this objective, relevant information of the behavior of the system has to be collected and evaluated. Hence, monitoring is also required in this step. Within S-Cube deliverables, we identified nine practices from the three adaption steps. These are:

- Identify adaptation need
  - *Define adaptation requirements:* Identify the aspects of the SBA model that are subject to change, and the expected outcome of the adaptation process.
  - *Define requirements to the monitoring subject:* In order to satisfy the adaptation requirements, this practice focuses on specifying what artifacts are expected to be monitored
  - *Define monitored property:* Specify which properties of the monitoring subject should be monitored.
  - *Provide monitoring functionality:* Monitoring functionalities that satisfy the monitoring requirements are provided through monitoring realization mechanism.
  - *Collect monitoring results for adaptation:* Results of monitoring are collected and analyzed.
  - *Trigger adaptation:* Evaluate the results from the monitoring analysis against adaptation requirements. If the need for adaptation is identified, send a request to trigger adaptation process.
- Identify adaptation strategy
  - *Design adaptation strategy:* Design the ways through which the adaptation requirements are satisfied.
  - *Select adaptation strategy:* Decide which particular adaptation strategy to be chosen based on the specific adaptation needs.
- Enact adaptation
  - *Perform adaptation:* The actual adaptation process is performed through adaptation realization mechanisms based on the selected adaption strategy.

These adaptation-related practices are presented in [9, Figure 2].

SOA Engineering Approaches

The next stage of this project was to identify what adaptation practices are identified within SOA engineering approaches. For the research presented here, we analysed 16 SOA approaches [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [4] [42] [43], and note that only five approaches explicitly mentioned some activities or tasks that are related to adaptation. These five approaches and the activities identified to be related to adaptation are illustrated in [9, Figure 3].

Identification of Adaptation Practices

Through our analysis of the S-Cube literature and the SOA approaches, we took two steps to map the adaptation practices identified from SOA approaches to the ones from S-Cube deliverables:

– Step 1: Cluster adaptation-related practices identified from SOA approaches that share similar objectives in the analysis performed, we identified that different practices identified from SOA approaches share similar objectives. For instance, monitoring messages sequences from ASTRO, monitor service, application, middleware, OS, hardware, and network from BEA, monitor workloads from SDLC, and monitor services from the SeCSE methodology are all about performing monitoring but on different subjects. Therefore, we clustered them into a group called *monitoring*.

– Step 2: Map the clustered practices to the S-Cube life cycle when analysing S-Cube literature, we were also able to identify those practices which existed in both defined SOA approaches and were previously identified in S-Cube deliverables and were subsequently clustered together and mapped to the S-Cube lifecycle phases. For instance, detect protocol violations from ASTRO aims at catching the misbehaviors by external partners according to the business process protocol; evaluate SLA QoS metrics aims at assessing quality attributes of the system of interest based on the corresponding SLAs. Both of these two practices assess the execution of the system against pre-defined requirements by collecting and analyzing monitoring results. Therefore, we mapped these two practices to *collect monitoring results for adaptation* in the S-Cube reference model as shown in Figure 2.7.

  During this step, we noticed that some adaptation practices we identified belong to the Adaptation phase, there are others which, while coming under Adaptation within SOA approaches and S-Cube life-cycle literature, actually belong to the Evolution phase of the S-Cube life-cycle. For instance, KPIs and management policies (from BEA) as well as service properties (from the SeCSE methodology) are defined at the requirement engineering phase. They are not directly used by adaptation practices but are relevant to adaptation process in that specifying these attributes makes corresponding monitoring and assessment possible.

Progression through these steps allowed us to develop a modified S-Cube life-cycle model for the Adaptation phase as shown in Figure 2.7.

2.4.3 Phase II: Software Maintenance Practices

For the second phase of adaptation practice identification, we focused on the software maintenance process. As adaptation is the modification of software in a dynamic environment, we expected that practices from static modification (maintenance) would or could also be of importance during services evolution. During Phase II development, we mapped maintenance practices from ISO/IEC 14764 to the adaptation activity groups illustrated in Figure 2.8. Some of the maintenance practices are also relevant to the evolutionary phases of the reference lifecycle. However we are focusing on adaptation for this phase.

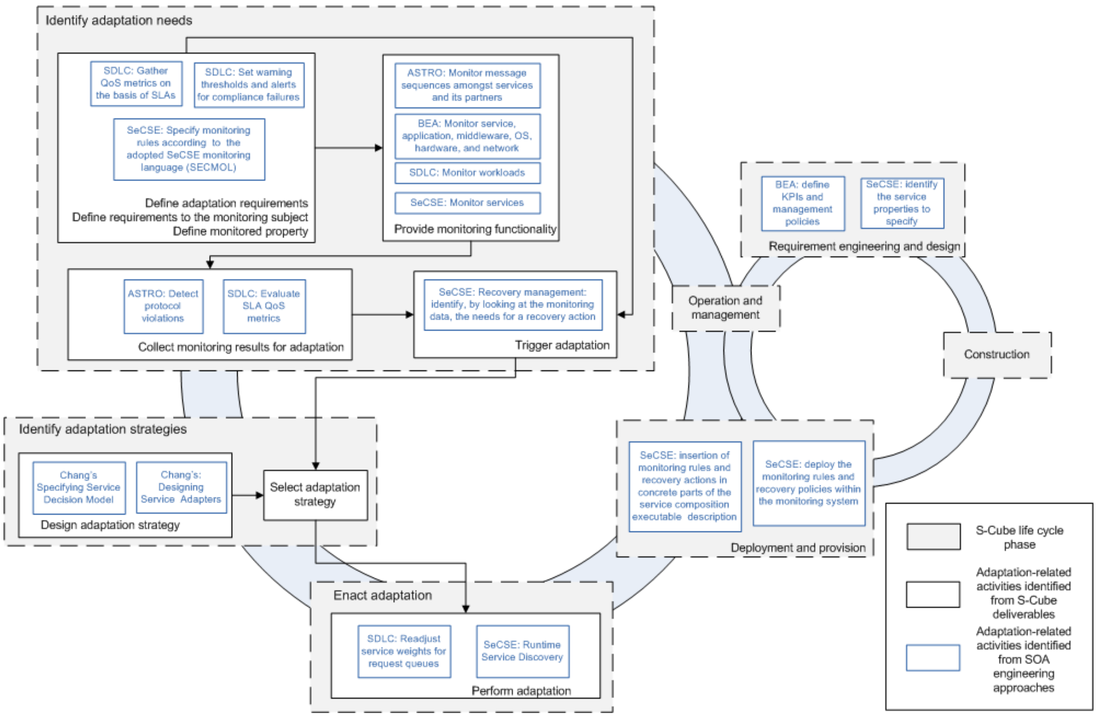

Identify adaptation needs
SDLC: Gather QoS metrics on the basis of SLAs
SDLC: Set warning thresholds and alerts for compliance failures
SeCSE: Specify monitoring rules according to the adopted SeCSE monitoring language (SECMOL)
Define adaptation requirements
Define requirements to the monitoring subject
Define monitored property
ASTRO: Monitor message sequences amongst services and its partners
BEA: Monitor service, application, middleware, OS, hardware, and network
SDLC: Monitor workloads
SeCSE: Monitor services
Provide monitoring functionality
ASTRO: Detect protocol violations
SDLC: Evaluate SLA QoS metrics
Collect monitoring results for adaptation
SeCSE: Recovery management: identify, by looking at the monitoring data, the needs for a recovery action
Trigger adaptation
BEA: define KPIs and management policies
SeCSE: identify the service properties to specify
Requirement engineering and design
Operation and management
Construction
Identify adaptation strategies
Chang's Specifying Service Decision Model
Chang's: Designing Service Adapters
Design adaptation strategy
Select adaptation strategy
SeCSE: insertion of monitoring rules and recovery actions in concrete parts of the service composition executable description
SeCSE: deploy the monitoring rules and recovery policies within the monitoring system
Deployment and provision
Enact adaptation
SDLC: Readjust service weights for request queues
SeCSE: Runtime Service Discovery
Perform adaptation
S-Cube life cycle phase
Adaptation-related activities identified from S-Cube deliverables
Adaptation-related activities identified from SOA engineering approaches

Figure 2.7: Mapping between adaptation-related activities identified from the SOA approaches and S-Cube lifecycle phases (from the S-Cube life cycle model perspective).

Maintenance Practices which are Applicable to Adaptation Activities

Of the 19 maintenance practices identified from the software engineering literature (ISO/IEC 14764), 13 of them can be mapped to adaptation activities. Table 2.4 shows which maintenance practices can be mapped to which adaptation activities.

Many of the mappings in Table 2.4 are apparent, for example "Maintenance plans and procedures " to "Design adaptation strategy" or "Modification Request/Problem Report procedures" to "Provide monitoring functionality". Some of the other mappings however are not so apparent, such as "Maintenance Review/Acceptance" practices that map to the "Collect monitoring results for adaptation" phase. Here we will explain the mappings from Table 2.4.

(1) **Process Implementation** The process implementation process area has activities such as maintenance planning and configuration management that would be beneficial to adaptation of services. Maintenance plans would be quite similar to an adaptation plan or strategy for a SBA. Similarly it is also worth noting that when an SBA adapts, the new service configuration could be tracked with a configuration management system.

(2) **Problem and Modification** Analysis Practices from the Problem and Modification Analysis process area can be mapped quite intuitively to the requirements, design and monitoring activities for SBAs.

(3) **Modification Implementation** In the Modification Implementation process area the Analysis practice could be part of service requirements gathering, and the Development Process practices could be leveraged for performing SBA adaptations.

(4) **Maintenance Review/Acceptance** The review practices could be beneficial to monitoring activities for a number of reasons. Reviews involve the collection/monitoring of data, after which further

Maintenance Practices	Adaptation Activities
1 Process Implementation	
Maintenance plans and procedures	Design adaptation strategy
MR/PR procedures	Provide monitoring functionality
Configuration management	Perform adaptation
2 Problem and Modification Analysis	
MR/PR analysis	Define adaptation requirements
Verification	Collect monitoring results for adaptation
Options	Design adaptation strategy
Approval	Select adaptation strategy
3 Modification Implementation	
Analysis	Define adaptation requirements
Development process	Perform adaptation
4 Maintenance Review/Acceptance	
Reviews	Collect monitoring results for adaptation
Approval	
5 Migration	
Migration	Define adaptation requirements
Migration plan	Design adaptation strategy
Post-operation review	Collect monitoring results for adaptation

Table 2.4: Maintenance Practice to Adaptation Activity Mapping

action may need to be taken. Similarly reviews are carried out continuously to ensure that software systems are performing acceptably to predefined measures such as Service Level Agreements (SLAs).

(5) **Migration** At first it is difficult to see how Migration practices may be mapped to the adaptation activity groups, however if Migration is seen as adaptive maintenance then the comparison becomes more clear. Adaptive maintenance shares many of the same characteristics as service adaptation in that the adaptation needs requirements, strategy and monitoring.

Maintenance Practices which are not Applicable to Adaptation Activities

Not all of the maintenance practices from ISO/IEC 14764 could be mapped to the adaptation activities because they are too specific to the maintenance process. The documentation practice from the maintenance process does not get included or is paid very little attention to in any of the adaptation activities covered in the literature. The 4 migration practices: notification of intent, implement operations and training, notification of completion and data archival are specific to the the maintenance of traditional software and cannot be leveraged for service adaptation.

As a result of the mapping that we carried out, we have developed practices within the S-Cube lifecycle as shown in Figure 2.8.

2.5 A Unified Formal Model for Controlled Evolution of Services

During the life time of a service a series of changes can be applied to it. These changes may have an impact on the environment of the services i.e., on the clients and other services that are using them as part of a service composition, and may require their adaptation in turn. This can create a propagating effect throughout the network that the service participates in. The research presented in [12] focuses on

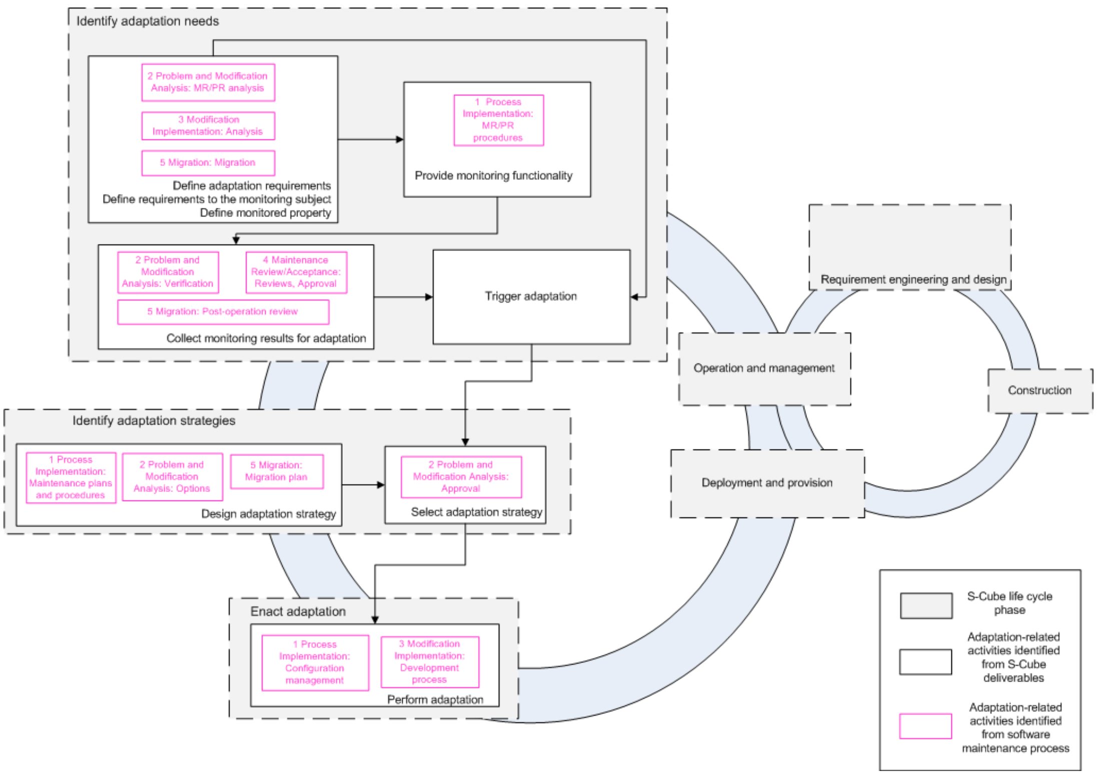

Identify adaptation needs
2 Problem and Modification Analysis: MR/PR analysis
3 Modification Implementation: Analysis
5 Migration: Migration
Define adaptation requirements
Define requirements to the monitoring subject
Define monitored property
1 Process Implementation: MR/PR procedures
Provide monitoring functionality
2 Problem and Modification Analysis: Verification
4 Maintenance Review/Acceptance: Reviews, Approval
5 Migration: Post-operation review
Collect monitoring results for adaptation
Trigger adaptation
Identify adaptation strategies
1 Process Implementation: Maintenance plans and procedures
2 Problem and Modification Analysis: Options
5 Migration: Migration plan
Design adaptation strategy
2 Problem and Modification Analysis: Approval
Select adaptation strategy
Enact adaptation
1 Process Implementation: Configuration management
3 Modification Implementation: Development process
Perform adaptation
Requirement engineering and design
Operation and management
Construction
Deployment and provision
S-Cube life cycle phase
Adaptation-related activities identified from S-Cube deliverables
Adaptation-related activities identified from software maintenance process

Figure 2.8: Mapping maintenance processes to S-Cube reference lifecycle.

identifying the criteria under which changes to service interfaces can be said to constitute *controlled evolution* of a service by not having an impact on the service environment. In this sense this work provides the service developers with the means to decide and evaluate the impact of a proposed modification of a service either as part of the Evolution or the Adaptation cycle. It also allows SBA stakeholders to check whether a change to one of its composed services requires the initiation of the Adaptation cycle or it can be resolved as part of the Evolution cycle (e.g. by re-binding the SBA to the new version of the service).

The enforcement of controlled evolution is a non-trivial problem: a robust and flexible framework for service description, together with a set of mechanisms for reasoning about the effect of changes to the services' environment are required. In particular:

2.5.1 Service Description

A service description as defined in the OASIS Reference Architecture [44] is an information artifact containing not only the description of the service interfaces but also information like conversation protocol schemes and QoS characteristics. This information is required by the potential consumers to decide whether the service is appropriate for their needs and consequently on how to consume it. Previous approaches to the problem of service description have focused on either the structure or the behavior of services and have not integrated the two aspects seamlessly. Our goal in [12] is to provide a unified framework for modeling these two aspects of services that will allow service providers and consumers to reason about both in terms of interoperability.

Furthermore, the fact that the producer and consumer roles are not mutually exclusive but mutable in a service ecosystem has to be acknowledged and reflected in the description of a service. The relationships between what the service consumes from other providers and what it provides to its consumers have to be identified. A service provider can then reason about whether a change to one of his providers will affect his consumers and act accordingly. Service description should also be technology-agnostic to avoid different assumptions and discrepancies due to the implementation details of various technological standards. At the same time it should map easily to existing (Web) services standards so that it is directly applicable to existing service implementations. Service providers must also have the means to reason about and control the impact of evolution of their services to their consumers. Mechanisms must be put in place to guarantee that - at least in terms of description - a change to a service is shallow [45].

At the heart of the proposed service description model is the Abstract Service Description (ASD) that represents a particular version of a service. An ASD consists of elements - informational constructs representing the building blocks of the service - and their relationships. The elements have types in the sense of classic typing theory [46] defining their structure. In that respect, each element and relationship is considered an evaluation of its respective type. Furthermore, each ASD is generated from a meta-model that contains generic concepts and their relationships - "classes" of element and relationship with common characteristics in all service descriptions.

In [47] we introduced a formal reference model for the technology agnostic description of services that made a similar distinction between concepts and elements. In [12] we modify this model so we can apply typing theory principles to it. More specifically, we apply the *subtyping* relation $\leq$ as defined in [**?**] to compare element types: an element type is a subtype of another iff it contains all the constituent parts of the latter element type or subtypes of them (and possibly more). This property allows for replacing one element with another under certain conditions, investigated in the later sections of the paper. Comparing relationships to check for replaceability requires a different approach since relationships can only differ in terms of their multiplicity domains in order to be comparable. For that purpose we are extending the *subset* $\subseteq$ relation in a similar to subtyping fashion.

2.5.2 Behavioral Aspects of Service Description

The concepts and elements discussed in the previous section focus on the syntax and structure of services and they are not suitable for behavioral aspects like the data and control flow of operations, messages and message content. For that purpose, based on the work on Web service contracts and following the notation of [48] and [49] we introduce the notion of orderings.

First though we need to introduce a way to partition the element set $\mathcal{E} := e_i, i = 1, ..., n$ of an ASD based on the purpose of each element. More specifically, we define the sets $Required\ \mathcal{E}^{req}$ that contains the input-type elements of an ASD and $Provided\ \mathcal{E}^{pro}$ that contains the output-type elements of an ASD [50]. It holds that $\mathcal{E}^{req} \cup \mathcal{E}^{pro} = \mathcal{E}$. The relationship set $\mathcal{R}^e$ can be segmented in a similar way: $r \in \mathcal{R}^{e,view} \Leftrightarrow \exists e \in \mathcal{E}^{view}/r.e_{name} \in \mathcal{E}$ where $view = req, pro$. In other words, the relationship belongs to the view that the source element belongs to. Based on this view on the elements set we define the orderings set as:

Definition Orderings $\mathcal{O}$ is the set of (possibly infinite) terms that can be generated from the set $N = \{e_{name} \mid \forall e \in \mathcal{E}\}$ containing all the element names from set $\mathcal{E}$ using the following grammar:

$$\begin{aligned} &a = n \mid \bar{n}, n \in N \\ &o := \mathbf{0} \mid a.o \mid o + o \mid o \oplus o \mid o \parallel o \mid o^{\otimes} \end{aligned}$$

where:

- with n we denote the name of an element that belongs to the $\mathcal{E}^{req}$ set, and $\bar{n}$ to the $\mathcal{E}^{pro}$ set respectively,
- **0** is the empty ordering, containing no action,
- . the sequence operator: $x.y$ means that y succeeds x,
- + the external choice: the consumer of the service chooses (exactly one) between the options x and y in the ordering $x + y$,
- $\oplus$ the internal choice: as the external choice, but this time the provider chooses,

– $\parallel$ the parallel operator: $x \parallel y$ means that both orderings occur in parallel,

– $\otimes$ the internal Kleene star: $x^{\otimes}$ denotes a finite sequence of x orderings.

We therefore extend the work on service contracts in [49] in three important dimensions:

(1) Service contracts assume the existence of a unique naming scheme for their elements and focus on the behavioral description of a service, ignoring the syntax and structure of the service description. not only provides the required naming scheme but at the same time it provides an explicit connection between behavioral and structural description aspects.

(2) Based on the connection between these aspects we are able to apply the same mechanism for describing not only the ordering of messages and operations but also the ordering of the content of the messages themselves.

(3) The work on contracts focus on identifying compatible providers and consumers; our emphasis is on the evolutionary aspect of the service description.

As in the case of elements and relationships, service evolution is also expressed through changes to orderings. This calls for a way to compare versions of orderings and check whether one can be replaced safely by another. For that purpose we will be using the weak subcontract relation $\preceq$ [49] which behaves as the (inversed) subtyping for orderings. Very briefly, this relation holds between $\sigma \preceq \tau$ if every client that works with σ can also work with τ. $\preceq$ has two basic properties: it allows the replacement of one ordering with a “more deterministic” one (e.g. $a\bar{c}_1 \oplus a\bar{c}_2.ac_3 \preceq a\bar{c}_1$ since a client that expects either $a\bar{c}_1$ or $a\bar{c}_2.ac_3$ can work also with simply ac_1). It also allows for the replacement of an ordering with one that offers more options (e.g., $a\bar{c}_1 \preceq a\bar{c}_1 + a\bar{c}_2.ac_3$ since the second option can simply be ignored) either in terms of width (more available actions) or depth (longer orderings). For further theoretical details on the definition see [49].

2.5.3 Controlled Evolution of Services

The use of elements and their relationships in conjunction with the orderings provides us with the means to describe both structural and behavioral aspects of a service using the ASD:

Definition ((Unified) ASD) An Abstract Service Description (ASD) S of a service s is the triple $(\mathcal{E}, \mathcal{R}^e, \mathcal{O})$ of its elements, relationships and orderings.

This definition of an ASD as S encompasses both aspects and describes both structure and behavior of a service independently of the technology used. Furthermore the application of the segmentation of $\mathcal{E}, \mathcal{R}^e$ using views (but not of $\mathcal{O}$ since it contain orderings that belong to both views) provides us with the means to distinguish between the purpose of the description element, relationship or ordering with respect to the service role in a particular interaction.

In addition, in [12] we discuss how existing Web services standards can be mapped to the presented technology-agnostic theoretical framework. Some of the artifacts (building blocks) of the standards have been omitted for reasons of brevity and relevance (e.g. “syntactic sugar”) but the crucial constructs of each of the major standards for Web services, i.e., XML Schema, WSDL and BPEL have been included.

The application of shallow changes to a service according to the classification of [45] means that a change to a service will not afect its consumers and as such it is possible to replace one version of the service with another. In that case we are characterising the new ASD as conformant to the older version. Conformance can be defined (and checked) using the subtyping, subsetting and subcontracting relations introduced in the previous sections as follows:

Definition (ASD Conformance) An ASD $S' = (\mathcal{E}', \mathcal{R}', \mathcal{O}')$ of a service is conformant w.r.t. to another ASD $S = (\mathcal{E}, \mathcal{R}, \mathcal{O})$, and we write $S \leq S', if$

1.$\forall e \in \mathcal{E}^{req}, \forall r \in \mathcal{R}^{req}, \exists e' \in \mathcal{E}'^{req}, \exists r' \in \mathcal{R}'^{req} / e \leq e' \wedge r \subseteq r'$(contravariance of input)
2.$\forall e \in \mathcal{E}^{pro}, \forall r \in \mathcal{R}^{pro}, \exists e' \in \mathcal{E}'^{pro}, \exists r' \in \mathcal{R}'^{pro} / e \leq e' \wedge r \subseteq r'$(contravariance of output)
3.$\forall o \in \mathcal{O}, \exists o' \in \mathcal{O}' : o \preceq o'$, (invariance of behavior)

This modular definition of conformance provides a formal foundation to discuss backward- and forward-compatible service versions that have been previously documented as best practices and design guidelines in a series of industry articles. It can be shown that the techniques discussed for compatibility in these articles are a subset of the possible conformant-respecting changes that can occur to an ASD.

Chapter 3

Research Contribution

The research by [13] and discussed in Section 2.1.1 has allowed the project team to attain a greater understanding of TSE and SOSE process models and characteristics of SOSE methodologies. This aids the establishment of guidelines and principles for engineering large scale and complex SBAs. It points out aspects that differentiate between service engineering and traditional software engineering and highlights issues which need to be focused on when developing of SBAs. The study of the link between the SOSE development process and delivered SBAs may further point out whether the emphasis on service aspects may eventually facilitate the delivery of high quality SBAs or whether it only aids the development process itself. The research in this deliverable and its associated papers, are working towards aspects of the S-Cube project as presented in Chapter 1:

(1) Understand how to model and reason on the context from where adaptation/evolution requirements should come.

(2) Consolidate our understanding on adaptation and evolution of service-based applications and identify proper approaches for selecting adaptation requirements and for reason on them in order to handle possible conflicts and inconsistencies.

(3) Integrate the results achieved on the human-computer interaction aspects with the body of knowledge we have acquired on the life cycle for adaptable service-based applications.

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. We have presented research which contributes to the early requirements stage of the S-Cube life-cycle. It enables the documentation of requirements that are discovered using alternative course events specific to the characteristics of service-based applications. These requirements are specified using guidelines that are tailored to the required characteristics of such applications.

We have also demonstrated that 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 and the whole application functional and non-functional requirements. This suggests that the identification of the most suitable adaptation strategy to activate can be a complex issue since different system characteristics have to be considered and that this complex issue should be considered when designing service-oriented systems for adaptation.

We have shown that there are subsequent levels that we need to specify in the S-Cube life-cycle. We have considered actions and artefacts which should be part of the life-cycle and identified phases where these are used. In addition, through the investigation of software engineering maintenance and software-oriented processes, we have commenced the definition of practices for use in services development to ensure adaptation.

Additionally, we investigated under which criteria can the evolution/adaptation of a service, expressed as a modification of the service interface, be safe i.e., impact-free for its environemnt (users and other services). Since SBAs and Adaptable SBAs compose and depend on a number of services for their operation then they are subject to changes that are the result of either the Adaptation or the Evolution cycle of each service in turn. Being able to predict as a service provider the impact of a (new) version of a service to its environment allows for deciding on the versioning strategy to be followed (maintain one version, only major versions, major and minor versions) which is crucial for the deployment and provisioning and operation and management phases of the life-cycle. Furthermore, it provides the (Adaptable) SBA with a mechanism for checking whether a change to one of its composed services requires the initiation of the Adaptation cycle or it can be covered within the Evolution cycle by e.g. re-binding.

We have presented which ingredients are needed for building and operating Adaptable Service-Based Applications. An Adaptable SBA not only is usually able to satisfy some requirements, but it also poses new requirements in terms of monitoring and adaptation aspects. For this reason we have introduced adaptation and monitoring requirements. The former concern the need for detecting situations that may trigger the need for adapting an SBA. The latter 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. Moreover we have introduced the context aspect that includes users and execution properties. Execution properties are those that concern the conditions under which the SBA and its component service execute. We have further developed the Adaptation cycle, investigating the identification of adaptation needs, identification of adaptation strategy and the enacting of adaptation at a practice level.

The need of a holistic approach for the development of adaptable service based applications arises due to the complexity and the dynamism of such systems. The research for this deliverable has allowed us to further define the S-Cube life cycle, demonstrating the importance of context for requirements, specifying the adaptation- and monitoring-specific actions and the main design artefacts that are exploited to perform adaptation and the phases where they are used, and defining the practices for the Adaptation phase.

Chapter 4

Conclusion

4.1 Contribution to S-Cube Vision

The research presented in this S-Cube deliverable has allowed the project team to work towards the S-Cube research vision from a number of different perspectives. We have commenced the integration of theories and knowledge from a variety of disciplines - mainly, at this point, services and software engineering. The research also contributes to the wider S-Cube vision for codifying knowledge to improve the specification of service-based applications. The research has deepened our understanding of how SOSE and TSE can be integrated to develop a more detailed S-Cube life-cycle. We see this being a significant input to IA 3, as we have researched developed principles for engineering and adaptation of service-oriented systems. The knowledge gained during the research presented here will be input into the Convergence Knowledge Model being developed in IA 1.1. Additionally, we envisage that the S-Cube life-cycle will be evaluated by industry, thus contributing to work package IA 2.2. It is important that in future activities that the research from this work-package is integrated with output from JRA 2, as the business models from this work package should be supportive of any life-cycle developed within the S-Cube project.

4.2 Future Research

We plan to study the extent to which service aspects play a role in the SOSE development process by carrying out a survey on the existing SOSE methodologies. We also plan to evaluate the impact of incorporating service aspects on SOSE process models by conducting industrial case studies to investigate opinions from real stakeholders involved in SOSE projects. By doing so, we may gain insight in how to emphasize the characteristics of SOSE methodologies and link them to the delivered SBAs. This research will support our working toward the goal of defining a holistic design method for adaptable SBAs. Possible future research includes:

(1) a refinement of guidelines, principles and practices;

(2) a formalization of guidelines, principles and practices;

(3) a definition of more precise criteria to decide on the patterns that are most appropriate for a given adaptation need;

(4) the development of mechanisms and tools that support the methodology proposed. We envisage an empirical evaluation of the proposed methodology by applying it to the real-world scenarios.

In designing for adaptation, we need to take the Adaptation phase into account. Our research to date on each of these topics will contribute to further investigate the connection between these two phases of the S-Cube life cycle. In addition, we reported proposed formative evaluations of the environment in use with requirements analysts in different organizations across Europe. The most obvious evolution of this research is to link classes of abnormal behaviour and state to different adaptation and evolution strategies that are being developed in JRA1.1.

4.3 Conclusion

In this S-Cube deliverable, the focus has been on refinement of the phases within the S-Cube life cycle. We have researched aspects of requirements, design for adaptation and monitoring, and the adaptation cycle, while also developing a formal model for the controlled evolution of services and have presented the results in detail in [9][12][10][13][14]. This research will be developed further in the next phases of the S-Cube project.

Appendix A

Notes on the usage of UML

UML has been used in this deliverable to describe part of our knowledge model. From the rich set of modeling constructs available for UML class diagrams, we chose the following subset with the meaning described below (see [51] for the rationale of using UML for knowledge modeling):

- *Actors:* An actor represents an entity of the application domain having some active role in the knowledge model. The role an actor has, could be played by a human or even by a system.
- *Class:* A class is used to document a concept of our knowledge model. Intrinsic properties, i. e., properties belonging only to this concept, are modelled as attributes. Mutual properties, i. e., properties representing a relation between concepts are represented as associations or aggregations (for the distinction between intrinsic and mutual properties, please refer to [52, p. 222]).
- *Association:* An association represents a relation between two (or more) concepts (mutual property). The label of the association describes the relation verbally.

Bibliography

[1] V. Andrikopoulos, P. Bertoli, S. Bindelli, E. D. Nitto, A. Gehlert, L. Germanovich, R. Kazhamiakin, A. Kounkou, B. Pernici, P. Plebani, and T. Weyer, "State of the art report on software engineering design knowledge and survey of HCI and contextual knowledge," 2008.

[2] M. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, vol. 10. NW Washington: IEEE Computer Society, 2003.

[3] P. Tetlow, J. Z. Pan, D. Oberle, E. Wallace, M. Uschold, and E. Kendall, "Ontology driven architectures and potential uses of the semantic web in systems and software engineering," http://www.w3.org/2001/sw/BestPractices/SE/ODA/, 2006. [Online]. Available: http://www.w3.org/2001/sw/BestPractices/SE/ODA/

[4] M. P. Papazoglou and W. V. D. Heuvel, "Service-oriented design and development methodology," *Int. J. Web Eng. Technol.*, vol. 2, no. 4, pp. 412–442, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1358575.1358582

[5] L. Baresi, "Toward open-world software: Issue and challenges," *Computer*, vol. 39, no. 10, pp. 36–43, 2006.

[6] M. Blake, "Decomposing composition: Service-oriented software engineers," *IEEE Software*, vol. 24, no. 6, pp. 68–77, 2007.

[7] G. McBride, "The role of SOA quality management in SOA service lifecycle management," 2007. [Online]. Available: http://www.ibm.com/developerworks/rational/library/mar07/mcbride/

[8] Q. Gu and P. Lago, "A stakeholder-driven service life cycle model for SOA," in *Foundations of Software Engineering*. ACM New York, NY, USA, 2007, pp. 1–7.

[9] S. Lane, Q. Gu, P. Lago, and I. Richardson, "Adaptation of services: A maintenance process?" in *(to be submitted)*, 2009.

[10] 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 *(to be submitted)*, 2009.

[11] "Separate design knowledge models for software engineering and service based computing," 2009. [Online]. Available: S-Cube,http://www.s-cube-network.eu/results/deliverables/wp-jra-1.1

[12] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou, "A unified formal model for controlled evolution of services*," in *(submitted to ICSOC09)*, 2009.

[13] Q. Gu, P. Lago, and E. D. Nitto, "Guiding the service engineering process: the importance of service aspects," in *2nd IFIP WG5.8 Workshop on Enterprise Interoperability (IWEI 2009)*. Valencia, Spain: Springer, 2009.

[14] N. Maiden, K. Zachos, A. Metzger, A. Gelhert, K. Karlsen, and N. Seyff, "Using scenarios to discover requirements for Service-Based applications," in *(to be submitted to REFSQ10)*, 2009.

[15] W.-T. Tsai, X. Wei, R. Paul, J.-Y. Chung, Q. Huang, and Y. Chen, "Service-oriented system engineering (SOSE) and its applications to embedded system development," *Service Oriented Computing and Applications*, pp. 3–17, 2007.

[16] M. Colombo, E. D. Nitto, M. D. Penta, D. Distante, and M. Zuccala, "Speaking a common language: A conceptual model for describing service-oriented systems," in *3rd International Conference on Service Oriented Computing (ICSOC)*, 2005.

[17] W. T. Tsai, "Service-oriented system engineering: A new paradigm," in *Service-Oriented System Engineering, 2005.*, Beijing, China, 2005, pp. 3– 6.

[18] M. D. Penta, L. Bastida, A. Sillitti, L. Baresi, G. Ripa, M. Melideo, M. Tilly, G. Spanoudakis, N. Maiden, J. G. Cruz, and J. Hutchinson, "SeCSE - Service Centric System Engineering: an overview," *At your service: Service Engineering in the Information Society Technologies Program*, 2009.

[19] T. Alspaugh, A. Anton, T. Barnes, and B. Mott, "An integrated scenario management strategy," in *RE99: IEEE Fourth International Symposium on Requirements Engineering*, 1999, pp. 142–149.

[20] P. Haumer, P. Heymans, M. Jarke, K. Pohl *et al.*, "Bridging the gap between past and future in RE: a scenario-based approach," in *Proceedings of the 4th IEEE International Symposium on Requirements Engineering*. IEEE Computer Society Washington, DC, USA, 1999, pp. 66–73.

[21] A. Mavin and N. Maiden, "Determining socio-technical systems requirements: experiences with generating and walking through scenarios," in *Proceedings 11 th International Conference on Requirements Engineering, IEEE Computer Society Press*, 2003, pp. 213–222.

[22] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, Tech. Rep., 2000.

[23] S. Robertson and J. Robertson, "Mastering the requirements process," 1999.

[24] T. Gilb and L. Brodie, *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Elsevier, 2005.

[25] N. Maiden, D. Lam, O. Gotel, X. Zhu, and S. Jones, "Validating a Model of Scenario-Based Discovery of Requirements," Technical Report, Centre for HCI Design, City University London, Tech. Rep., 2009.

[26] G. Dobson, R. Lock, and I. Sommerville, "Quality of service requirements specification using an ontology," in *SOCCER Workshop, Requirements Engineering*, vol. 5. Citeseer, 2005.

[27] N. Maiden, *'Systematic Scenario Walkthroughs with ART-SCENE', in 'Scenarios, Stories and Use Cases', Eds Alexander & Maiden*, 1st ed. Wiley, Oct. 2004.

[28] "S-cube deliverables." [Online]. Available: http://www.s-cube-network.eu/results/deliverables

[29] P. Allen, "SOA best practice report : The service oriented process." [Online]. Available: http://www.cbdiforum.com/secure/interact/2007-02/service_oriented_process.php

[30] A. Arsanjani, "Service-oriented modeling and architecture," Nov. 2004. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/

[31] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Gariapathy, and K. Holley, "SOMA: a method for developing service-oriented solutions," *IBM Syst. J.*, vol. 47, no. 3, pp. 377–396, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1466610.1466613

[32] ATOS, "SeCSE methodology, version 3," Tech. Rep., Mar. 2007.

[33] A. Brown, S. K. Johnston, G. Larsen, and J. Palistrant, "SOA development using the IBM rational software development platform: a practical guide," *Rational Software*, 2005.

[34] S. Durvasula *et al.*, "Introduction to service lifecycle. SOA practitioners guide. part 3," 2007.

[35] C. Emig, J. Weisser, and S. Abeck, "Development of soa-based software systems-an evolutionary programming approach," in *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, 2006, p. 182182.

[36] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J. P. Richter, M. Vo\ss, and J. Willkomm, "A method for engineering a true Service-Oriented architecture," in *To appear: Proceedings of the 10th International Conference on Enterprise Information Systems. Barcelona, Spain*, 2008.

[37] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: an architectural framework for service definition and realization," in *Services Computing, IEEE International Conference on*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 151–158.

[38] S. Jones and M. Morris, "A methodology for service architectures," *Capgemini, U.K.*, Oct. 2005.

[39] H. Karhunen, M. Jantti, and A. Eerola, "Service-oriented software engineering (SOSE) framework," in *International Conference on Services Systems and Services Management*, vol. 2. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 1199–1204 Vol. 2.

[40] S. Lamparter and Y. Sure, "An interdisciplinary methodology for building service-oriented systems on the web," in *Services Computing, 2008. SCC '08. IEEE International Conference on*, vol. 2, 2008, pp. 475–478.

[41] kunal mittal, "Service oriented unified process." [Online]. Available: http://www.kunalmittal.com/html/soup.html

[42] S. H. Chang, "A systematic analysis and design approach to develop adaptable services in service oriented computing," in *Services, 2007 IEEE Congress on*, 2007, pp. 375–378.

[43] 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 *Lecture notes in computer science*, vol. 3826, 2005, p. 495.

[44] J. A. Estefan, K. Laskey, F. G. McCabe, and D. Thornton, "Reference architecture for service oriented architecture version 1.0," 2008. [Online]. Available: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf

[45] M. P. Papazoglou, "The challenges of service evolution," in *CAiSE*, 2008, p. 115.

[46] L. Cardelli, "A semantics of multiple inheritance." *Inf. Comput.*, vol. 76, no. 2, pp. 138–164, 1988.

[47] V. Andrikopoulos, S. Benbernou, and M. Papazoglou, "Managing the evolution of service specifications," in *Advanced Information Systems Engineering*. Springer, 2008, pp. 359–374.

[48] S. Carpineti, G. Castagna, C. Laneve, and L. Padovani, "A formal account of contracts for Web Services," *Lecture Notes in Computer Science*, vol. 4184, p. 148, 2006.

[49] G. Castagna, N. Gesbert, and L. Padovani, “A theory of contracts for web services,” in *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM New York, NY, USA, 2008, pp. 261–272.

[50] V. Andrikopoulos, S. Benbernou, and M. Papazoglou, “Evolving Services from a Contractual Perspective*,” 2009.

[51] M. S. Abdullah, I. D. Benest, R. F. Paige, and C. Kimble, “Using unified modeling language for conceptual modelling of knowledge-based systems,” in *Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007), November 5–9, 2007, Auckland, New Zealand*, ser. Lecture Notes in Computer Science, C. Parent, K.-D. Schewe, V. C. Storey, and B. Thalheim, Eds., vol. 4801. Springer, 2007, pp. 438–453.

[52] Y. Wand and R. Weber, “On the ontological expressiveness of information systems analysis and design grammars,” *Journal of Information Systems*, vol. 3, no. 4, pp. 217–237, 1993.