Grant Agreement N° 215483

Title:	*Separate design knowledge models for software engineering and service based computing*
Author:	*CITY, FBK, Lero-UL, POLIMI, Tilburg*
Editor:	*Vasilios Andrikopoulos (Tilburg)*
Reviewers:	*Qing Gu, Patricia Lago (VUA)*
	Martin Treiber (TUW)
Identifier:	*Deliverable # CD-JRA-1.1.2*
Type:	*Deliverable*
Version:	*1*
Date:	*15 March 2009*
Status:	*Final*
Class:	*External*

Management Summary

This deliverable presents two distinct bodies of knowledge: the first one is for service oriented computing based on a proposed life cycle that incorporates adaptation-specific phases. Each phase is discussed in depth, and methods, techniques and tools for it are presented. Furthermore, cross-phase aspects are investigated. The other body of knowledge concerns more traditional software engineering and business process methodologies, examined from the perspective of service based applications. A number of preliminary results on the synergy between the two areas are also presented as a stepping stone for the following deliverables.

Copyright © 2008 by the S-CUBE consortium – All rights reserved.

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/

Contents

1 Introduction **5**
1.1 Context . . . 5
1.2 Main goals of JRA-1.1 . . . 6
1.3 Interactions with the other workpackages . . . 8
1.4 Deliverable objectives . . . 9
1.5 Document structure . . . 9

2 Preliminary definitions **10**
2.1 Agents and Actors . . . 10
2.2 Service Based Applications . . . 10
2.3 Types of services . . . 11

3 Knowledge Model for Service-Based Applications **13**
3.1 SBA life cycles . . . 13
3.2 Requirement Engineering and Design . . . 15
3.2.1 Requirement Engineering . . . 15
3.2.2 Design for Monitoring and Adaptation . . . 18
3.3 Construction . . . 21
3.4 Deployment and Provisioning . . . 22
3.5 Operation and Management . . . 25
3.6 Adaptation Life-cycle Phases . . . 28
3.7 Cross-cutting Concerns . . . 30
3.7.1 Service Governance . . . 31
3.7.2 Quality Assurance of SBAs . . . 32
3.7.3 Service Discovery . . . 34
3.7.4 Service Level Agreement Negotiation . . . 35
3.8 Summary . . . 36

4 Knowledge Model for Relevant Areas of Software Engineering **37**
4.1 Classical Software Engineering . . . 37
4.1.1 Software Process Quality . . . 37
4.1.2 Component-Based Software Engineering . . . 41
4.1.3 Legacy Systems Re-Engineering . . . 44
4.1.4 Evolution and Maintenance . . . 47
4.2 Business Process Methodologies . . . 48
4.2.1 DMAIC Methodology . . . 49
4.2.2 Supply Chain Operations Reference Methodology . . . 49
4.2.3 Discussion . . . 51
4.3 Summary . . . 52

5 Discussion on Knowledge Models 53

Chapter 1

Introduction

1.1 Context

The evolution of software methodologies and technologies can be seen as a progressive journey from rigid to flexible, static to dynamic, centralized to distributed solutions. The journey started in the late 1960's-early 1970's, with the attempts to discipline the software process through the identification of well-defined stages and criteria to be met to progress from a stage of the process to the next. The goal was to avoid endless iterations of code-and-fix activities, improve predictability, improve product quality, and reduce process costs. Continuous change to remove errors, to meet the customers' expectations, and to improve the implementation were felt as the main culprit of poor quality.

In [1] the term closed-world assumption was used to characterize the implicit hypotheses that underlie these initial approaches to software engineering: the development process and the structure of software products should have been fixed, static, and monolithic; in the cases of complex systems decomposed in modules these last ones were supposed to be statically bound to each other and the resulting frozen application was statically deployed on a physically centralized architecture.

The history of software engineering shows a progressive departure from the strict boundaries of the closed-world assumption toward more flexibility to support continuous evolution. This becomes clear already in the early 70ies when Parnas introduces the idea of design for change [2]. From there on systems progressively evolved from fixed, static, and centralized to adaptable, dynamic, and distributed. Methods, techniques, and tools were developed to support the need for change without compromising product quality and cost-efficient developments. The evolution has been both at the process level and at the product level. Evolutionary process models —such as incremental and prototyping-based— were introduced to achieve a better tailoring of solutions to user needs, and to reduce risks. More recently, these evolved into agile methods, like extreme programming. As processes, product architectures evolved from a static, centralized, and monolithic structure, where changes implied (partial) recompilation and redeployment of the application, to modular and distributed architectures. Design methods were also proposed to support change of the software architecture. The principles of information hiding, encapsulation, and separation of a module's interface from its implementation [2] were eventually embedded in new programming languages to enforce good design practices. The evolution of software technology allowed bindings among modules not only to become dynamic, but also to extend across network boundaries. Another major evolution thread was in terms of the ownership of an application. In the early stage, system development was under control of a single organization, which ultimately owned it completely. Next, component-based software development became dominant. COTS are developed and provided by third parties, who are also responsible for their quality and their evolution. Application development thus becomes (partly) decentralized. At an extreme, application development consists of gluing components together, by using middleware technology to provide an integration and coordination infrastructure.

The demand for software to live in an open world and to evolve continuously as the world evolves (the open world assumption), however, is now reaching unprecedented levels of dynamism. Over the past

years a major step of evolution toward this direction has been made possible by the birth of the concepts of services and service-based applications (more often called service-oriented architectures - SOAs in the literature), and by the development of technologies and proposed standards to support them.

Such evolution needs now to be fully conceptualized and understood in order to identify those methodological and formal means that allow us to build service-based applications with the required level of quality. A detailed discussion on the open issues and research areas can be found in [3].

1.2 Main goals of JRA-1.1

The goal of JRA-1.1 as a whole is to provide a contribution into the identification of the methodological and formal means for engineering service-based applications. In particular, we identify those ingredients that allow service-based applications to adapt and evolve to best fit the situation in which they live, and position them into a coherent lifecycle.

Figure 1.1 shows how we think the life cycle of service-based applications appears when adaptability comes into place [4]. Not only applications can undergo the transition between the runtime operation and the analysis and design phases in order to be continuously improved and updated (we call this part of our life cycle the *evolution cycle*), but they also have intrinsic mechanisms that, during runtime, continuously and automatically a) detect new problems, changes, and requirements for adaptation, b) identify possible adaptation strategies, and c) enact them. These three steps are shown in the left hand side of the figure and define what we call the *adaptation cycle*. The observation of the changes in the environment is obtained through monitoring which is part of the management activities typically performed during execution. This is one of the trigger for the iteration of the adaptation cycle, whose effect is to inject changes directly into the application being operated and managed.

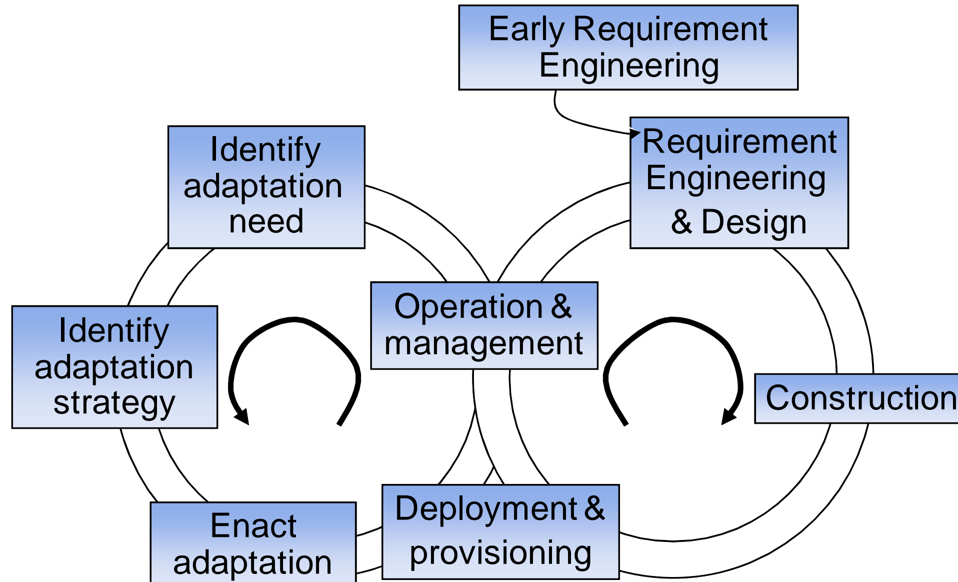

Figure 1.1: The lifecycle for evolvable and adaptable service-based applications.

This lifecycle of course should be refined and disambiguated, but it has the merit to be the first one explicitly taking both evolution and adaptation of applications into account. For the moment, we use it as the reference against which we compare other processes, lifecycles, and methods that are available in the literature with the objective of identifying what is missing and the aspects we should focus more. Also, we detail each of its activities and we study how the contributions from the other S-Cube workpackages and from other areas of software engineering could be framed within these activities. In particular,

similarly to what happens in the development of other kinds of systems, we need to understand how the life cycle and all the related processes can be measured, controlled, evaluated and improved.

Adaptation and evolution are triggered by the occurrence of some events that result in the emergence of some *requirements for adaptation/evolution*. These requirements can either be raised, more or less directly, by the human beings involved in the execution of service-based applications or they can be generated by the technological environment in which the system is running. In general, we say that the *context* in which the system and its actors are immersed has an important impact on the emergence of requirements for adaptation/evolution. The context has been characterized in the literature in various ways, depending on the application domains in which it has been studied. An important issue for S-Cube is to understand how to characterize the context of service-based applications and how to ensure that these applications are able to use it to identify the adaptation requirements.

Assuming that adaptable service-based applications are able to identify adaptation requirements, they should also be able to decide if and when to take them into consideration. There could be application states in which some adaptation requirements could not been used as they would lead the application into an inconsistent and unrecoverable case. Also, some requirements could be conflicting with each other and could require some reconciliation to take place before one of them is selected. The literature so far has addressed these issues only partially and with ad hoc solutions. The main challenge here is to identify proper modeling means that enable the automatic identification and analysis of adaptation requirements and the solution of the potential inconsistencies that can arise.

While the software engineering literature has provided through the last fifthy years proper approaches to design evolvable systems, a consolidated understanding on what to do to design adaptable systems is still to come. In the SOA literature some approaches have been identified to perform some limited adaptation, often on the basis on a hard-coded logic. We are interested in investigating the principles that enable the ability of *design for adaptation* and in defining a body of knowledge and methodological support that helps in this task.

As humans have a very important role in the open world either as users of service-based systems or as service providers themselves, we consider the aspects concerned with the so called Human-Computing Interaction (HCI) with particular attention. In particular, our main research challenge is to select and codify human-computer interaction knowledge that delivers new capabilities to the development and use of service-centric systems. Examples of that knowledge include user knowledge, user task knowledge, accessibility knowledge, and organizational culture knowledge. The identification and the refinement of the research method for such an activity is part of the research itself.

The last goal of JRA-1.1 is to try out the methods and approaches that will be developed not only in the typical information systems and B-2-B context, but also in pervasive domains such us the one of the Internet of things. By this term it is meant the possibility for special purposes devices such as navigation systems, PDAs, cellular phones, sensors, actuators, and the like to operate and be visible and accessible through the Internet. Such possibility that is being realized quite fast thanks to the technological advances in the areas of hardware and telecommunication systems is opening new very interesting challenges. While in the past relatively complex computations running on things were not possible, now these are being experimented in research. This, of course, opens up a huge number of new possibilities in terms of systems that pervasively influence the life of people and help them in several tasks and situations. For instance, through these devices we can imagine users access complex information systems, but also, in the opposite direction, information systems could access software services available on these devices to actuate local-scope operations such as the execution of a temperature monitoring function on some critical patient or the invocation of a “turn red for 5 min” service on all the semaphores on some critical paths. We could even imagine some systems where the computation is entirely in charge of devices that cooperate to achieve a common goal without a direct control of any centralized complex system. The literature of service-based applications so far has been mainly focusing (with some exceptions in the OSGI domain) on more traditional settings where devices (e.g., the car system) were used as a mechanism for the user to interact with services and, in limited cases, as data sources (this is the case of the GPS

that provides the position of the car to the service). Now, the challenge would be to understand how to have services living within devices and be accessed by other consumers running anywhere else, how to be aware and control the execution context of these services, how to handle the intrinsic limitations and peculiarities of devices that, being quite limited in terms of resources, surely require a high level of adaptability.

The goals we have briefly described here represent those that have been identified during the first year of the project. The proposal of solutions for some of them is an ongoing activity, while others will be addressed in the following years. In particular, in year two of the project we start tackling the following aspects:

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

On the longer term, we will focus on assessing the life cycle we have identified and on understanding how all related processes can be measured, controlled, evaluated, and improved.

In order to experiment with all engineering aspects of service-based applications, we will try to apply our approach in some concrete application domains more or less traditionally associated to SOA, and, possibly, to the internet of things setting.

1.3 Interactions with the other workpackages

The engineering and design workpackage interacts with all others by providing design principles and methods and receiving specific techniques and approaches. In particular, the approaches for Agile Service Networks (ASNs) and business process definition developed by WP-JRA-2.1 will be incorporated into the requirement engineering and high level design phase of the life-cycle. The same workpackage will also offer inputs on the identification of the needs for adaptation that typically arise at the level of the business processes. Vice versa, WP-JRA-2.1 will receive from the engineering workpackage hints and suggestions on how the ASNs and business process aspects are incorporated into the comprehensive life cycle.

The service composition metamodel and techniques provided by WP-JRA-2.2 will be incorporated into the construction phase. Vice versa, WP-JRA-1.1 will provide to the other workpackage new requirements for extending the composition metamodel and techniques to account for the design for monitoring and design for adaptation principles. For instance, a way to incorporate into the composition approach a proper model of the execution context should be identified. Other minor interactions with WP-JRA-2.2 concern the other phases of "deployment and provision" and all the phases of the adaptation life cycle (left hand-side of the figure).

The infrastructural services offered by WP-JRA-2.3 will be exploited and properly incorporated within the life cycle, partially, in the design phases (e.g., the discovery mechanisms) and, partially, in the runtime phases (e.g., the low-level adaptation mechanisms and, again, the discovery mechanisms).

Finally, the relationships between WP-JRA-1.1 and the other WP-JRA-1.* are quite strict, and therefore, the work will be, in many cases, conducted in strict collaboration with these workpackages. More in detail, WP-JRA-1.2 owns the left hand-side of the life cycle and coordinates with JRA-1.1 to receive developed applications that are ready to be monitored and adapted. WP-JRA-1.3 offers all those techniques that are needed to provide some quality guarantees on the service-based applications. Such quality guarantees can be provided at various different levels of abstraction and can concern various, if not all, phases

of the life cycle. The task of WP-JRA-1.1 is then to incorporate these techniques within the life cycle and to apply those software engineering principles that lead to the design of verification-ready applications.

1.4 Deliverable objectives

In this deliverable we focus on the analysis of the life cycle for adaptable and evolvable service-based applications. Thus, the objectives of the current deliverable are to:

- Compare the life cycles that have been proposed in the literature for service-based applications with our reference skeleton to identify missing parts and aspects that it is worth to analyze in a deeper way in the future.
- Identify the main concepts, issues, and challenges concerning the various phases of our reference life cycle as they have been identified in the literature.
- Analyze the areas of software engineering and business methodologies that can be relevant to service-based applications with the objective of identifying experiences and approaches that can be useful for service-based applications.

The analysis that we perform in this deliverable leads to the definition and consolidation of terms and knowledge that will be incorporated into the S-Cube Knowledge Model as part of the IA-1.1 work.

1.5 Document structure

Consistently with its objectives, the deliverable is structured as follows. Section 2 introduces some basic definitions that will be used and in some cases extended in the rest of the deliverable. Section 3 presents the life cycles for service-based applications and details the various activities in the life cycle. Section 4 summarizes the knowledge acquired from the fields of software engineering and business processes and discusses on how it can be exploited in the engineering of service-based applications. Finally, Section 5 draws the conclusions.

Chapter 2

Preliminary definitions

The goal of this chapter is to give a short overview of the main basic concepts that are relevant for the current deliverable and, in general, for the JRA-1.1 work. In particular, we identify the main actors that have a role in the context of service based applications, the main concepts concerned with the definition of service-based application, and the various kinds of services that have been identified in the literature so far. The terms will be described using UML as this way we can identify the relationships among them. These terms will be further detailed in the forthcoming sections of this deliverable and we will continue using UML class diagrams as vehicles for their representation. The diagrams reported in this section are an extension and clarification of some of those belonging to the conceptual model proposed in the SeCSE project [5]. In particular, as the concepts of Service-Based Application and Adaptable Service-Based Application are the focus of S-Cube, we have introduced these two within the diagrams showing the relationships between them and the other concepts.

2.1 Agents and Actors

The model of Figure 2.1 [5] identifies a set of Agents and Actors and the relationships between them. The model exploits the UML2.0 features that allow for the development of orthogonal inheritance hierarchies. In particular, it expresses the fact that Agents are entities of the real world and Actors are the roles the Agents may play. Agents in the diagram are Person, Organization and Systems (that may be Legacy Systems and Software Systems). They can act as Providers, Service Developers, Service Integrators, Consumers, Monitors, and Negotiation Agents. Providers can offer any kind of resource (including a whole application). Service Providers are those that specifically offer one or more services. Similarly, a Consumer can consume or exploit any kind of resource while Service Consumers consume, in particular, services.

The classification of Actors is overlapping, this means that, for instance, a Person, an Organization, or a System can act both as a Service Consumer and a Service Integrator. On the contrary, the classification of Agents is disjoint. This means that, for instance, a Service Consumer can be either a Person, an Organization, or a System. In the diagram, the human characters represent those Agents that are human beings and all Actors as all of them represent roles that can be potentially taken by human Agents.

2.2 Service Based Applications

A service-based application is obtained by composing various Services in order to satisfy the desired functionality. It is an Adaptable Service Based Application when it is able to react autonomously to changes and to self-adapt to them. Such kinds of service-based applications will be the main focus of this deliverable. Service-based applications are often implemented in terms of an *orchestration*, that is, a centralized logic that describes the order in which the various services are called and the way their

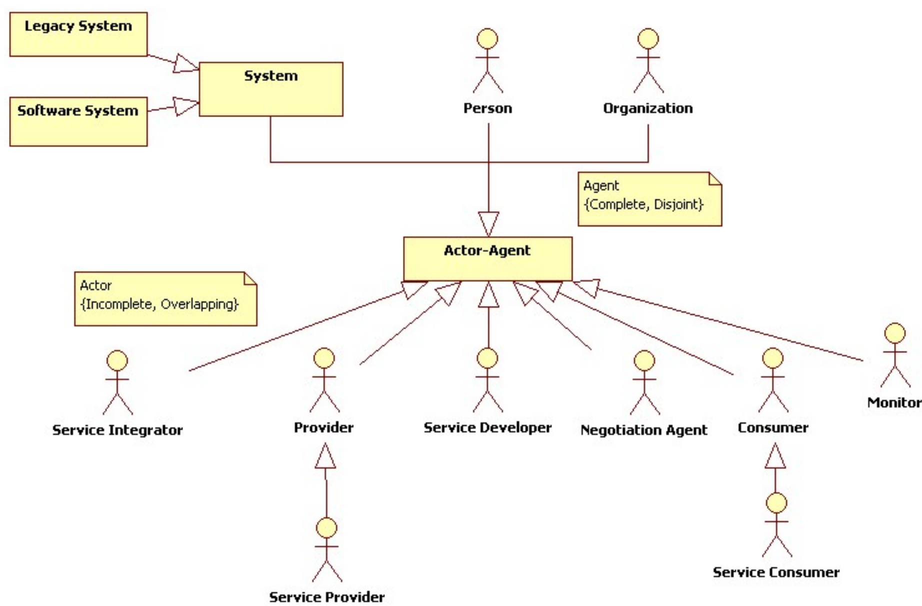

Figure 2.1: The Agent-Actor diagram.

parameters are formed and used for further calls. This orchestration is also called Service Process. The Service Integrator (see Figure 2.1) is the actor that is in charge of developing a service-based application, while the Provider is the one that offers (provides) it and the Consumer is the one that exploits it. Figure 2.2 shows the terms that we have defined and highlights the relationships among them.

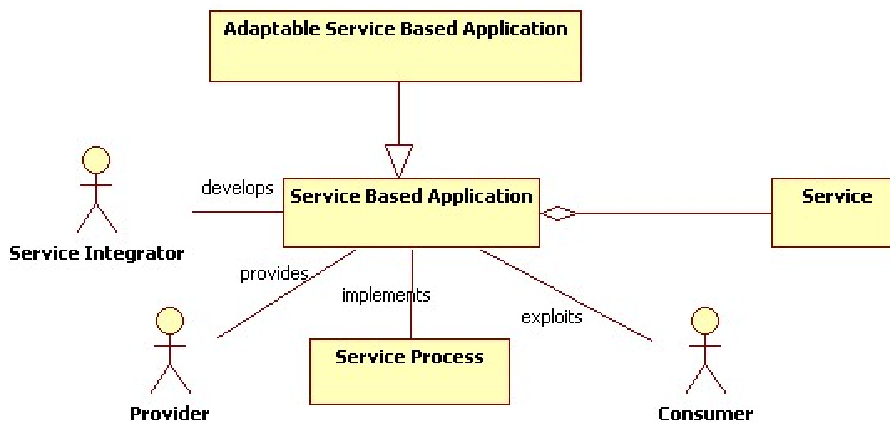

Figure 2.2: The Service Based Application diagram

2.3 Types of services

Services exploited in a service-based application can be offered by various different agents, as highlighted in Section 2.1 (for instance, they can be offered by Persons or by Organizations), or they can simply be software services exploiting some specific technology, e.g., web services.

Besides for the agent that is providing them, services may also differ for their nature. They can be abstract when they do not have a concrete implementation but only represent an idea that could correspond, possibly in the future, to various implementations. Of course, they are concrete when they are actually provided by some actor. This distinction is quite relevant when developing adaptable service-based applications as a Service Integrator at design time may reason even in the absence of Concrete Services simply by exploiting Abstract Services. Clearly, in this case, the resulting application will be executable only in those cases when at runtime some Concrete Service implementing the abstract ones exists and these services are selected in some adaptation step.

Orthogonally to this classification, Services can also be distinguished in Simple and Composite. Composite Services are service-based applications being accessible as services. The current technology for building service-based applications, BPEL, actually, only supports the development of Composite Services.

The last orthogonal classification refers to the statefulness of services. A special kind of Stateful Services are the Conversational Services. These store the state of the conversation with a single specific stakeholder, but keep the states of different conversations separate from each other.

The three classifications are shown in Figure 2.3. Given their orthogonality, they lead to the definition of eight possible types of services, all considered relevant and worth of being supported by proper service engineering technologies.

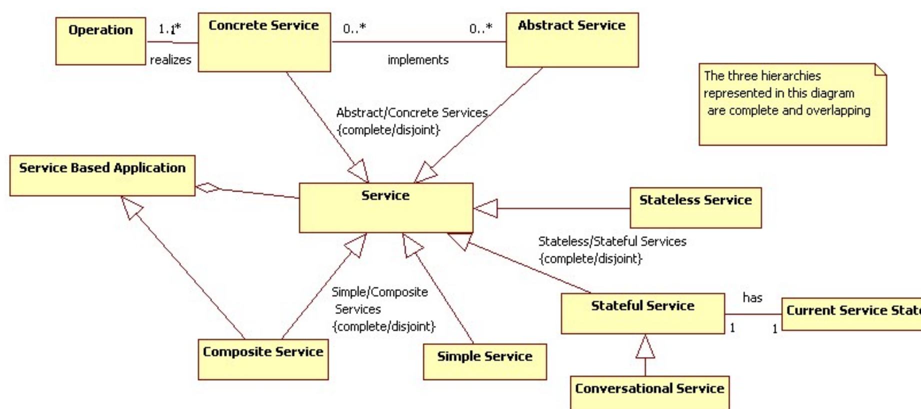

Figure 2.3: The service type diagram.

Chapter 3

Knowledge Model for Service-Based Applications

The purpose of this section is to elaborate on the reference life-cycle model for SBAs introduced in Section 1 (Figure 1.1), discussing the scope and purpose of each phase of the life-cycle and developing a knowledge model for SBA engineering by presenting a collection of processes, methodologies, techniques, and tools for each of those phases.

More specifically, Section 3.2 discusses the first phase in the life-cycle, i.e. requirement engineering and design. Consequently, Sections 3.3 and Section 3.4 build on that section to discuss the construction and the deployment and provisioning phases of the service life-cycle respectively. The final phase of that cycle, covering operation and management is discussed in Section 3.5. Section 3.6 summarizes the adaptation aspect of the proposed life-cycle (the left circle in Figure 1.1). Finally, Section 3.7 presents issues that are *cross-cutting* through a number of phases and deserve to be discussed separately.

3.1 SBA life cycles

As discussed in deliverable PO-JRA-1.1.1 [4], the distinguishing feature of the adaptable applications, in contrast to classical SBAs, is the ability to accommodate and manage various changes at runtime. This, in turn, requires the capability to observe the changes relevant to the application execution, and the capability to enact the corresponding adaptation strategy.

The following sections summarize the various service and SBA life-cycle methodologies covered more extensively in PO-JRA-1.1.1, and focus on presenting which functionalities (in the form of phases) from the reference life cycle of Figure 1.1 are provided by each methodology, and, in turn, what each of them has to offer in terms of concepts, methodologies, and processes.

The Web Services Development Lifecycle (SLDC)

The Web Services Development Lifecycle (SLDC) [6] is a continuous and iterative approach to the development, implementation, deployment and maintenance of software services. It utilises standards, reference architectures, and run time environments that are required to provide guidance during the design, development and production phases of the software service lifecycle. The development lifecycle model includes an introductory phase along with eight key lifecycle phases throughout the development of the service: planning, analysis, design, construction, testing, provisioning, deployment, execution and monitoring. The reference life cycle of Figure 1.1 is based on this methodology; for that reason SLDC covers all the phases of the right side of the life cycle in a more fine-grained way, but does not discuss the adaptation phases. Nevertheless, [7] discusses the way that SLDC has to be extended to manage the evolution of services by supporting a *change-oriented life cycle* that allows for re-configuration, alignment, and control of cascading through the service network changes in an orderly fashion.

Rational Unified Process (RUP) for SOA

The Rational Unified Process (RUP) is a framework that aims to support the analysis and design of iterative software systems. It was created with component based development (CBD) and object oriented analysis and design (OOAD) in mind [8], so it is not easily transferable to serviced-based applications. With some adjustments however, several of its milestones can be adjusted to fit in SOA solutions [9, 10]. RUP for SOA is based on IBM's *Business-Driven Development for SOA* model that breaks down the iterative development of a service into phases of: modeling the business (process), definition of the requirements, analysis & design, implementation, testing, deployment, management, and optimization. For each of these phases special provision is taken for the asset and change management. In that sense, RUP for SOA is covering the same aspects of service lifecycle management as SLDC does, but instead of not dealing directly with adaptation issues, it depends on project and change management capabilities to allow for flexibility.

Service Oriented Modeling and Architecture (SOMA)

Service Oriented Modeling and Architecture (SOMA) is a framework developed by IBM to help practitioners effectively model the architecture of service based applications. Service oriented modeling is a service oriented analysis and design (SOAD) process for modeling, analysing, designing and producing a SOA that aligns with business goals [11]. The SOMA method of analysis and design attempts to make practitioners think about business goals in a top-down manner from the beginning of the project, and then develop the services from a bottom-up perspective once high level business goals have been satisfied. At a high level, SOMA consists of three primary activities (inter-related phases): identification, specification and realization. Service identification refers to the business process of identifying services to help realise business goals. Service specification aims to classify services and specify the details of the components that implement the services. This classification categorizes services into a service hierarchy reflecting the composite services that can be created by combining other, finer grained services in the hierarchy. Finally, service realization aims to assign the specified services to the legacy or custom built components that can realize them. These primary activities correspond to the requirements engineering and construction aspects of the reference lifecycle. In addition to the three primary activities, SOMA also provides activities for business modeling and transformation, existing solution management, implementation by building or assembling and testing of services, deployment, monitoring and management of services, and governance to support SOA [12]. The left hand side of the lifecycle that deals with adaptation does not appear to be represented by any of the SOMA activities.

Service Oriented Analysis and Design/Decision Modeling (SOAD)

Service oriented analysis and design (SOAD) is a structured approach to the analysis, design and realisation of quality SOAs [8]. The foundation for the SOAD model originates from fields such as Enterprise Architecture (EA), Object-Oriented Analysis and Design (OOAD - particularly RUP) and Business Process Management (BPM). While SOAD process and notation have yet to be defined in detail [8], key elements such as conceptualization (or identification), service categorization and aggregation, policies and aspects, meet-in-the-middle process, semantic brokering and service harvesting (for reuse) can already be identified. SOAD, like SOMA, focuses primarily on the analysis, design and architecture of SBAs and does not include provisions for service adaptation. SOAD, therefore satisfies only the right hand side of the reference lifecycle.

ASTRO

ASTRO is a methodology that is focused primarily on service composition or service orchestration. This is a process that combines existing third party services in order to provide new services with novel functionality [13, 14]. The methodology also provides verification functionality, so that the correct operation of services can be verified, in both on-line and off-line mode [15, 16]. The ASTRO methodology covers all the life cycle of service-based systems, starting from their requirements specification, going

	SLDC	RUP for SOA	SOMA	SOAD	ASTRO	BEA Services Lifecycle
Analysis, Design & Implementation						
Early Requirements Engineering	X	X	X	X	X	X
Requirements Engineering and design	X	X	X	X	X	X
Construction and Quality assurance	X	X	X	X	X	X
Deployment and Provisionining	X	X			X	X
Operation Management and Quality Assurance	X	X			X	X
Adaptation		Relies on project			X	(X)
Identify Adaptation Requirements		management to			X	(X)
Identify Adaptation Strategy		allow for			X	(X)
Enact Adaptation		adaptation			X	(X)

Figure 3.1: Lifecycles vs. Reference Lifecycle

through their realization and deployment, and finally to their execution, possibly looping back to re-design, triggered by the monitoring of unexpected/unwanted run-time behaviors. In addition, it supports the evolution and adaptation of SBAs by allowing the design of service compositions directly from requirements and by enforcing an incremental development approach that iteratively refines the behavioral requirements.

BEA Services Lifecycle

The BEA Service lifecycles specifies the stakeholders, the tools, the deliverables, the processes and the best practices for each stage of the services lifecycle [17]. The BEA methodology covers the phases of the services development and maintenance lifecycle in three primary stages: requirements and analysis, design and development, and IT operations. The lifecycle also has a business dashboard phase which provides business stakeholders with business intelligence data. The entire lifecycle is underpinned by a governance process which promotes the interoperability, discoverability and standardisation of services and leverages the adaptation of services to new requirements. In that sense the BEA Service lifecycles indirectly handle the left-hand cycle in Figure 1.1.

Summary

Figure 3.1 shows how each of the discussed lifecycles compare to the reference lifecycle introduced in Figure 1.1. It is evident that the ASTRO and BEA lifecycles (to a lesser extent) are the ones which bear closest resemblance to the reference lifecycle. The following sections investigate each phase in more depth.

3.2 Requirement Engineering and Design

In this section we address the phase of Requirement Engineering and Design. In particular, section 3.2.1 explains the phase of Requirement Engineering, while section 3.2.2 reports the phase of Design focusing on adaptation and monitoring.

3.2.1 Requirement Engineering

Before developing any system, the developers must understand what the system is supposed to do and how its use can support the goals of the individuals or business that will pay for that system.

This means understanding the application domain and the specific functionality required by the stakeholders. Requirements Engineering (RE) is the name given to a structured set of activities that help developing this understanding.

Requirements are derived from documents, people, and the observation of the social context of people expressing them. In fact, requirements are expressed by the stakeholders using concepts strictly related to their social world. Stakeholders may be different and numerous; they include paying customer, users, indirect beneficiaries, and developers. Their social worlds may be distinct and they may express goals, often conflicting, depending on the different perspectives of the environment in which they work. This, together with the fact that often stakeholders are not able to make explicit their tacit knowledge, makes the elicitation of requirements a very critical and difficult to accomplish activity.

Not only requirements have to be elicited, but they have also to be documented, possibly in a formal way. Also, their evolution needs to be managed and kept under control in order to guarantee that the implemented system can evolve with them.

In general, the activities that belong to the RE process varies depending on the complexity of the application being developed, the size and culture of the companies involved. Large systems require a formal RE stage to produce a well documented set of software requirements. For small companies developing innovative software products, usually the RE process might consist of brainstorming sessions leading to a short vision statement of what the software is expected to do.

Regardless of the process used, some activities are fundamental to all RE processes [18]:

Elicitation Identify sources of information about the system and discover the requirements from these.

Analysis Understand the requirements, their overlaps, and their conflicts.

Validation Check if the requirements are what the stakeholders really need.

Negotiation Try to reconcile conflicting views and generate a consistent set of requirements.

Documentation Write down the requirements in a way that stakeholders and software developers can understand.

Management Control the requirements changes that will inevitably arise.

These activities constitute a cyclic process performed during the system lifecycle.

The components of the process are sketched in Figure 3.2. The activity of *Requirement Elicitation* consists of gathering and clarifying the needs of the purchaser and the business goals. Several techniques may be used for Requirement Elicitation, such as focus group, interviews, questionnaires, contextual observation and others approaches. In the *Requirement Analysis* the goals are decomposed in sub-goals and the best strategies to satisfy each goal are individuated; this activity produces a set of functional requirements that will be validated to check potential conflicts among the requirements (*Requirement Validation*). If conflicts are detected, a negotiation activity (*Requirement Negotiation*) is required to obtain a set of consistent requirements then formalized in a *Requirement Documentation*. Obviously a continuous management requirement activity (*Requirement Management*) is performed to manage changes.

The main outcome of the RE process is a requirements document that defines what is to be implemented. The industry increasingly recognizes the importance of using good RE processes and appropriate RE techniques when developing software systems to achieve high software quality. Researchers emphasize the necessity of adopting proper requirements engineering techniques in order to derive high quality specification. Davis [19] states that knowing which technique to apply to a given problem is necessary for effective requirements analysis. Requirements are often written in natural language and are often vague descriptions of what is wanted rather than detailed specifications: this could be the best choice in domains where requirements change quickly.

Requirements change is inevitable, because the business environment where the software is executing is highly dynamic: new products emerge, businesses reorganize, restructure, and react to new opportunities. Boehm [20] argues that in order to deliver systems rapidly that meet customer needs, a key challenge is to reconcile customer expectations with developer capabilities. He developed an approach to RE called the WinWin approach, in which negotiation between customers and software suppliers is central.

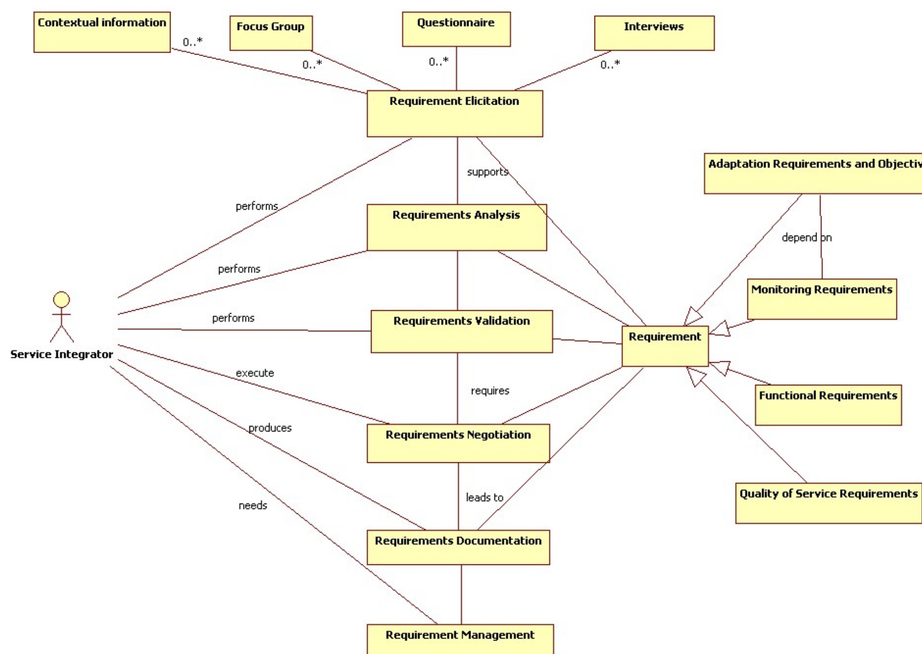

Figure 3.2: Requirement Engineering Process diagram

Service Oriented Requirement Engineering

Service-Oriented Requirements Engineering (SORE) specializes RE for service-based applications. SORE is an important topic in Service-Oriented System Engineering (SOSE) and an emerging research area; it assumes applications developed in an SOA framework running in an SOA infrastructure. SORE shares with traditional requirement engineering the same activities. However, some of them are conducted in a different way. The most remarkable difference is that service and workflow discovery has a very significant role in SORE as part of the requirement elicitation and analysis activities. While, usually, traditional RE activities do not deal with pre-existing software, the availability of services and of service descriptions in registries allow System Integrator to exploit this knowledge to enable reuse [21].

Maiden [22] focuses on the availability of services in registries during SORE, and suggests that existing services guide requirement elicitation; moreover results and information of the queries in the registries can be reused in the forthcoming searches.

In recent years, researchers have begun to develop techniques that could be employed by requirements engineers to identify service requirements specified in SLAs. For example Bohmann et al. [23] argue that one of the key features of applications hosting services is the heterogeneity in customer requirements. Their aim is to assist service providers to address heterogeneity in customer requirements through matching and mapping required service features and factors during RE and design phases. In traditional RE Macaulay [24] identified poor communication between stakeholders as the key factor of limiting or enabling effective RE. As technology and systems are embedded within socio-organizational contexts and processes, strong socio-technical approaches to RE are required. Lichtenstein et al. [25] suggest that in the new IT services era new techniques and approaches are needed for eliciting and determining provider and customer requirements; moreover it is required to involve key stakeholder groups to negotiate the sometimes-conflicting provider and customer service needs.

Among SBAs, Adaptable SBAs play a significant role. In such applications the phase of RE must take in account the mechanisms of reaction to critical conditions or changes in the environment or in the user needs. One of the main challenges in the RE for Adaptable SBAs is the difficulty to know in advance

all the possible adaptations since it is unfeasible to anticipate requirements for all the possible critical conditions that may happen. While RE for traditional systems reports what the system "shall do", RE for adaptable systems reports what the system "can do if something happens". RE for adaptive systems is an open research area, offering a limited number of approaches. Some research was conducted to use a goal models approach in describing the requirements of an adaptable system. Goldsby et al. [26] proposed an approach to modeling the requirements of an adaptable system using i^* goal models. In particular, using i^* goal models, they represent the stakeholder objectives, the business logic, the adaptive behavior, and the adaptive mechanism needs.

3.2.2 Design for Monitoring and Adaptation

An Adaptable Service-Based Application is a service-based application augmented with a control loop that aims to continuously monitor the execution and evolution of the application and its environment and to modify it on the basis of application-dependent strategies designed by system integrators. In general, the adaptation may be caused by different reasons: it may be a necessary tool for the application to recover from unexpected problem or failure; to customize the application in order to better fit the current execution context or to better satisfy the needs of a particular application user; it may be required in order to improve the application performance or to optimize the way the application resources are managed.

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

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

- *Identify appropriate adaptation and monitoring capabilities*. When the adaptation and monitoring requirements are defined, there is a need to identify the possible candidates for their implementation. These refer to the existing adaptation and monitoring frameworks and tools provided at different functional SBA layers and to various mechanisms enabled at different layers for more general purposes, such as online testing [27], data and process mining for monitoring purposes, and service discovery, binding and automated composition [28] for adaptation purposes.

- *Define monitoring properties and adaptation strategies*. Requirements and capabilities identified in previous steps are used to provide concrete monitoring and adaptation specification for a given SBA. These specifications may be given implicitly when they are hard-coded within the given approach or explicitly. For instance, when one deals with the recovery problem, a typical implicit

monitored property refers to the failures and exceptions not managed by the application code [29]. Accordingly, the self-optimization approaches often rely on the predefined threshold for certain quality of service properties for triggering adaptation need; the corresponding adaptation strategy (e.g., re-composition) is also often predefined [30].

- *Incorporate adaptation and monitoring mechanisms*. Based on the above specifications, the adaptable SBA is extended with the corresponding monitors and adaptation mechanisms. Depending on the mechanisms, this extension may require integrating the monitoring and/or adaptation functionalities into the SBA code or into the underlying execution platform. A typical example of the former approach is presented in [31]: the underlying BPEL process is augmented with the calls to the a special proxy that evaluates the monitored properties. In [32] analogous code modification is applied in order to inject the necessary adaptation actions. On the other side, monitoring approaches presented in, e.g., [28, 33], as well most of approaches to Business Activity Monitoring, rely on the mechanisms for generating monitors independent from the application and on the specific tools respectively.

Conceptual Model

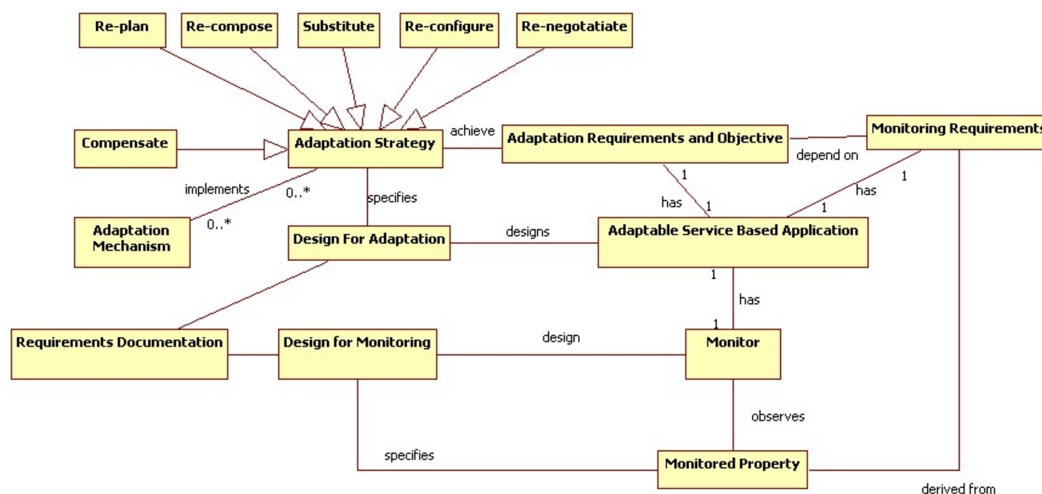

Figure 3.3: The design for adaptation diagram

The presented design for adaptation and monitoring concepts are represented in Figure 3.3. The *Adaptable Service-Based Application* is associated with the *Monitoring Requirements* and *Adaptation Requirements*, which define when the changes in the application functionality or environment become critical and what we should achieve in that case respectively. From these requirements one should derive the *Monitored Properties* and *Adaptation Strategies* achieving them. In order to be monitored, the application is associated with the *Monitors* that continuously observes the monitored properties representing critical changes in the SBA functionality or environment. The *Adaptation Mechanisms* are identified and incorporated in order to achieve the defined strategies.

There exists a wide range of adaptation strategies to be used by different approaches. In a simple case, adaptation targets modification of the application parameters (e.g., re-configuration, re-negotiation of the SLAs, substitution of one failed or underperforming service with another one) without changing its structure. In more complex cases, the adaptation involves also modification of the application structure (e.g., re-compose the services, re-plan the underlying process, or introduce specific activities that compensate the incorrect results achieved by the faulty execution).

We remark that the different forms, approaches, and methodologies used for the SBA monitoring and adaptation, as well as the ways the corresponding mechanisms are realized, is out of scope of the presented knowledge model; they are discussed and classified in the corresponding deliverables of other S-Cube workpackages, in particular in CD-JRA-1.2.2 [34]. Below we will only present some aspects relevant for the specification of the adaptation strategies and monitoring properties.

Adaptation and Monitoring Specification

As we already mentioned, the monitoring specification defines the moment and conditions "when" adaptation activities should be triggered, while the adaptation specification prescribes "how" the adaptation should be performed. Both these specification may be given either explicitly, or implicitly.

Explicit monitoring specification is defined using standard notations (such as WS-Agreement or WS-Policy) or specific languages (such as RTML [28], WS-CoL [31]). In the first case the specification is first translated into some internal representation specific monitoring framework, and then is given as input to the corresponding monitoring tool.

Explicit adaptation specification may have different forms:

- *goal-based* specification, where the adaptation activities are described in a higher-level form that contains only objectives to be reached, leaving the system or the middleware to determine the concrete actions required to achieve those objectives. This goals may have the form of certain utility function to be maximized [30], declarative functional goal specification [35], etc.
- *action-based* specification, where the activities are defined explicitly. In the corresponding languages the strategies are specified using high-level action specification, where actions correspond to re-binding, terminating, selection of alternative behavior, rolling back to some previous stable state, etc [32, 36].
- approaches based on *explicit variability* modeling. In such approaches the identified variation point is associated with a set of alternatives (variants) that define different possible implementations of the corresponding application part. In business processes this corresponds, for example, to a nominal sub-process, and a set of potential customized flows [37].

With implicit adaptation specifications the decisions when the system has to be changed and which actions to perform are predefined by the adaptation framework. This is a typical situation for dynamic service compositions, where the services are selected and composed dynamically upon, e.g., unavailability of some of them. This is also the case for many self-healing systems, where the recovery activities are somehow hard-coded. The role of the design activities in case of implicit adaptation is to provide possibly richer and more complete descriptions of the services and compositions in order to support and simplify the decisions made at run-time automatically. In case of dynamic composition, for example, these decisions correspond to the discovery and selection of the candidate services. Implicit adaptation specification may have different forms that we shortly introduce in the following and that are already explained in detail in the deliverable PO-JRA-1.1.1 [4]:

- *Quality driven* specification that supports dynamic composition of services with the goal of optimal valuation of service qualities. In this way the composed process (e.g., in BPEL) is designed as a workflow composing elementary tasks. At run-time a concrete elementary service is selected to perform a particular task from a community of services that provide the same functionality, but have different quality characteristics. The description of the services, therefore, should include not only functional aspect, but also non-functional properties that are required in the selection process. The predefined goal of this kind of specification is, therefore, at run-time optimize the values of characteristics;

- *Reputation-based* specification approaches that target the problem of maintaining dynamic service compositions, when the component services fail or become defective. If the service invocation was successful, the reputation is positive, while in case of failure the value degrades. They allows improving the quality of selection;

- *P2P self-healing* approaches support the dynamic look-up and replacement of elementary services that failed during the execution of the process. The key idea is that they use peer-to-peer resource management for publishing and discovery and binding of the necessary services.

- *Adapters-based* approaches have the objective to automatic generate mediators based on predefined requirements (e.g., deadlock freeness) or semi-automated methodologies for identifying and modelling instructions and procedures for adapting the specification (transformation templates or commands);

- *Local knowledge-based* approaches allow run-time adaptation of the system configuration according to changes in the context. The key idea is to define properties of a system starting from the local knowledge, defined as the knowledge about its immediate structure. Local knowledge is used to reconfigure the structure of the system when a change in the context is found, and is propagated upward when needed.

- *Semantic Web-based* approaches specify protocol mediation allowing for the automatic adaptation of the service requester behaviour meeting the constraints of the providers interface, by abstracting from the existing interface description. A shared ontology is used to understand the semantics of the domain actions.

3.3 Construction

The construction of SBAs is based on top of the design phase, where the model of the future SBA is defined and described. The construction of SBA assumes the definition and specification of the executable code of the corresponding Service-based Application on top of the existing services or service templates. In the latter case, the abstract service definitions are used, which are bound to concrete services at deployment/provisioning time.

The construction of an SBA as an executable service composition may be achieved in several ways (Figure 3.4). Note that since a Service Based Application isn't always exposed as a service, we could have a Provider (at not necessarily a Service Provider). At the highest level of abstraction we distinguish between:

- *Manual construction* of a service composition. In this case the goal of the service integrator is to define an executable process composed of concrete or abstract services using an appropriate service composition specification language. In deliverable PO-JRA-2.2.1 [38] a variety of languages for the construction of SBA are presented. Among them Business Process Execution Language (BPEL for short, [39]) is one of the prominent standard languages supported by industry and accepted by the community. It supports loosely coupled composition of Web services described in a standard WSDL notation. Besides BPEL, there exists a variety of notations for the construction of composed service compositions and service-based business processes such as JOpera [40], jPDL [41], etc.

- *Model-driven service composition*, which copes with generating service orchestration models from more abstract models, which are often abstract business process models created by business analysts. Notations like BPMN [42] or WS-CDL [43] may be used for these purposes.

- *Automated service composition*. Here the goal is to automatically generate the executable SBA using available service models (abstract and concrete, stateful and stateless) and predefined composition goals that restrict the behavior, functionality, and QoS parameters of the future SBAs. The composition goals are usually defined during the SBA design phase, and are specified in high-level notations (see, e.g., [44, 45]).

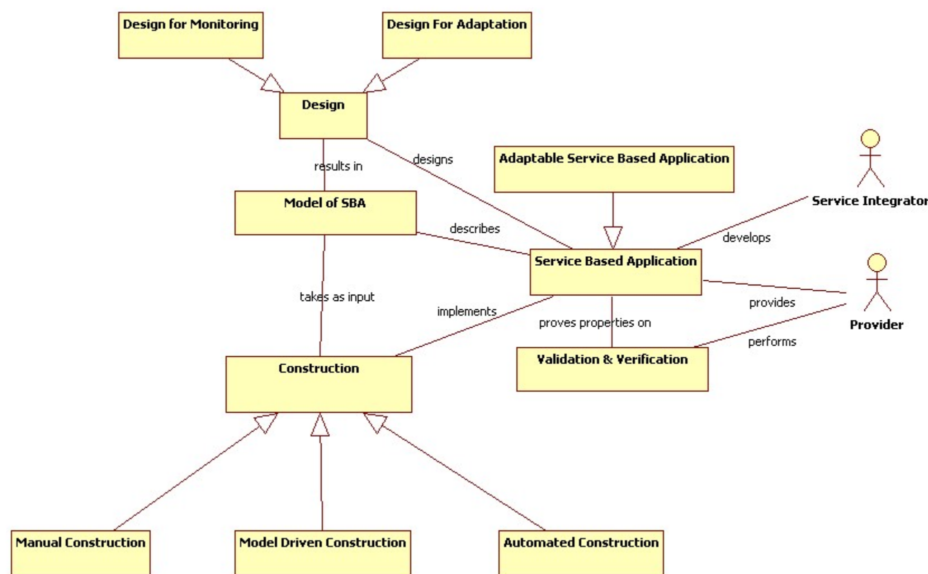

Figure 3.4: The construction diagram

Another important activity that should be accomplished during the construction phase as well as in other phases of the development and operation process is the *verification and validation* of the SBA against various requirements and constraints. Section 3.7.2 provides a summary of this issue that is tackled more extensively in deliverable PO-JRA-1.3.1 [46] and CD-JRA-1.3.2 [47].

3.4 Deployment and Provisioning

The phase "Deployment and Provisioning" of the lifecycle in Figure 1.1 comprises all the activities related to the publication and deployment of a Service Based Application; this section discusses the major concepts related to this phase.

Service Description

As reported in the deliverable PO-JRA-1.1.1 [4], a Service Description allows the users to access a service regardless of where and whom it is actually offered. It specifies all the information needed to the potential consumers to access and use the service. Web services transform the Web from a distributed source of passive information into a distributed source of active services. When a consumer decides to acquire the use of a service, he would be sure the service fulfills his/her expectations both in terms of offered functionality and of non functional characteristics. Thus, it is important to have an expressive service description that does not only report the syntactical aspects of the service, but also describes its meaning in a human readable format, describes its QoS, the way its operations should be used, and the like. The description is provided by the service provider during the service publication (see next section), and it is used by the service consumer to choose the correct service during the service discovery.

The standard description for Web service is provided by the Web Service Description Language (WSDL) [48]: it is an XML language describing the public interface of the service. It offers a syntactical description of the service permitting the consumer to interact with it; WSDL, among the other things, gives information about the location, and the types of input/output messages of the service. Such information, though essential, isn't often enough to provide the full understanding of the service.

Hence, the need for more expressive service descriptions arises: such an attempt was carried out by the Semantic Web initiative. The Semantic Web has added machine-interpretable information to Web content in order to provide intelligent access to heterogeneous and distributed information. In a similar fashion, Semantic Web concepts are used to define intelligent Web services, i.e., services supporting automatic discovery, composition, invocation and interoperation. This joint application of Semantic Web concepts and Web services in order to realize intelligent Web services is usually referred as Semantic Web Services. A lot of proposals addressing the semantic Web services try to improve the current technologies such as SOAP, WSDL and UDDI because they provide very limited support in mechanizing service recognition, service configuration and combination, service comparison and automated negotiation. Among them, an important solution is represented by OWL-S [49] that enriches the service descriptions with rich semantic annotations facilitating automatic service discovery, invocation and composition.

An important framework for service description is the Web Service Modeling Framework (WSMF) [50] whose aim is to provide an appropriate conceptual model for developing and describing services and their composition. The WSMF consists of four different main elements: ontologies that provide the terminology used by other elements, goal repositories that define the problems the Web services should solve, Web services descriptions that define various aspects of a Web service and mediators to bypass interoperability limits. WSMF's aim is to enable fully flexible and scalable e-commerce based on Web services providing an architecture characterized by:

- Strong *de-coupling* of the various components that realize an e-commerce application.
- Strong *mediation* service enabling anybody to speak with everybody in a scalable manner.

Among other proposed approaches we can include BPEL4WS [51] and BPML [52] /WSCI [53]: they offer similar functionalities; in fact they define a language to describe process models, offer support for service choreography and provide conversational and interoperation means for Web services. They focus on the composition of services, permitting the description of services interactions. The need to have an exhaustive service description is examined, among the others, by the SeCSE project. The view in Figure 3.5 [5] focuses on the way SeCSE views Service Description. A *Service Description* comprises a *Service Specification* and, if available, some *Service Additional Information*. A Service Specification is usually defined by the *Service Developer* and may include both functional and non-functional information such as information on the service interface, the service behavior, service exceptions, test suites, commercial conditions applying to the service (pricing, policies, and SLA negotiation parameters) and communication mechanisms. Service Additional Information may include information such as user ratings, service certificates, measured QoS and usage history. Both Service Specification and Service Additional Information could be specified by means of different *Facets*. Each Facet is the expression of one or more *Service Properties* in some specification language. A Facet represents a property of a service such as, for example, binding, operational semantics, exception behavior. Within a facet, the property can be encoded in a range of appropriate notations. So each service in the SeCSE environment is described by an undefined set of Facet permitting to the consumer to gain understanding of the service.

Service Publication

Service providers can make their services accessible via Web service interfaces. In order to make a Web service usable by other parties, a provider will publish the Web service description at some network location reachable by target users. It is a common practice to publish syntactic WSDL descriptions of Web services at UDDI (Universal Description, Discovery, and Integration) [54] repositories, which act

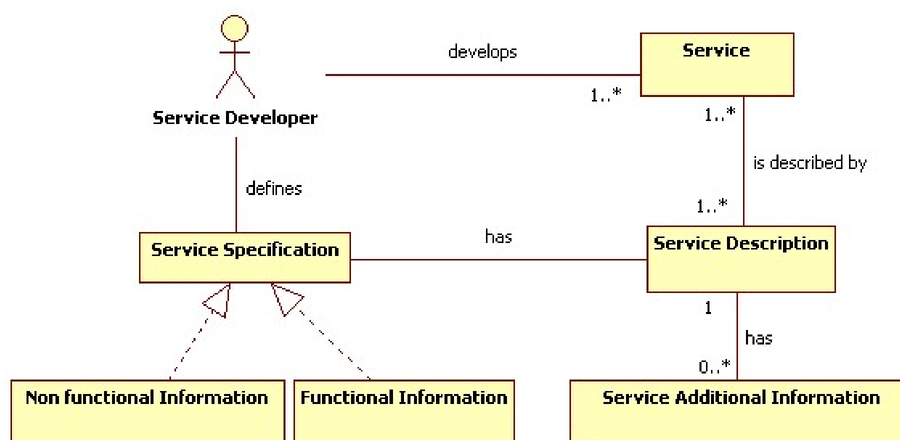

Figure 3.5: The SeCSE Service Description diagram

as a common entry point for the location of Web services and provide keyword-based search facilities, as well as searching based on categories in taxonomies such as UNSPSC (Universal Standard Products and Services Classification) [55]. A UDDI registry is similar to a CORBA trader, or it can be thought as a DNS service for business applications. A UDDI registry has two kinds of users: businesses that want to publish a service description (and its usage interfaces), and clients who want to obtain services descriptions of a certain kind and bind to them. The UDDI entry contains the following elements:

- The Business entity, which provides general data about a company such as its address, a short description, contact information and other general identifiers. This information can be seen as the white pages of UDDI.
- A list of Business services. These contain a description of the service and a list of categories that describe the service, e.g. purchasing, weather forecast etc. These can be considered as the yellow pages of UDDI.
- One or more binding templates define the green pages: they provide the more technical information about a Web service [56].

The main goal of UDDI was to speed interoperability and adoption for Web services through the creation of standards-based specifications for service description and discovery, and the shared operation of a business registry on the Web.

Another solution addressing the service repository is provided by ebXML [57]. Like its predecessor, UDDI, ebXML Registry also facilitates seamless and automatic inter-enterprise collaborations. This feature enables integration between trading partners permitting the communication and functionality sharing among SOA applications without human interaction. An ebXML registry can have a persistence mechanism for enterprises, allowing to share and store information as registered content: XML artifacts can be stored, maintained, and automatically discovered, increasing efficiency in XML-related development efforts. There are two general ways in which an e-business registry may be used: for discovery and for collaboration: while, UDDI is focused exclusively on this discovery aspect, ebXML Registry is focused on both discovery and collaboration. Due to its focus on storing and maintaining XML artifacts, an ebXML registry can used for a collaborative development of XML artifacts within an organization and for a run-time collaboration between trading partners. Note that there is the possibility of run-time interoperability between UDDI and an ebXML registry. For example, it is possible to discover an ebXML registry from UDDI, and vice versa.

The publication of WSDL descriptions at UDDI repositories is characterized by two limitations: a) manual assignment of Web services to categories, and b) the use of syntactic descriptions does not allow for advanced search based on formal semantics. An evaluation and comparison of the Web services registry was led in 2005 by Dustdar et al [58]. Actually, UDDI specification has not received a lot of support from industry and many products implement. In literature, a lot of proposals to enable the retrieval of Web services based on the semantic description can be found [59]. The METEOR-S project [60] proposes an environment for federated Web services publication and discovery among multiple registries: it uses an ontology-based approach to organize registries, enabling semantic classification based on domains. Each registry supports semantic publication of the service, used during discovery process. Several works exist in the literature that extend UDDI or ebXML and propose federated architectures usually based on the P2P paradigm (for example [61], [62]).

Deployment of Service-Based Applications

The term Deployment is used to refer to the process of concretely associating services to devices in the real world system, and all the activities that must be executed to achieve it.

Dynamic deployment, in particular, is related to the body of techniques that are needed to apply such a process in a dynamic context, where changing conditions in the environment must be taken into consideration, together with changes in the requirement, QoS, and other aspects. Dynamic deployment is particularly important in a service-based context where new services or new versions of the same services need to be deployed without stopping or interfering with the normal execution of the others. Some of them, concentrating on the possibility of dynamically deploying services, are also dealing with the degree of reusability of services, and how flexibly they can be configured. The main goals of these approaches are indeed both to provide a high level of QoS and to enable dynamic deployment. A deployment infrastructure for service-based applications should offer the following elements: ways to describe the services that are required for the execution (if any) and ways to describe the software components to deploy (both of the two above aspects belong to “the what” cathegory); where to deploy these services/components, a strategy for deployment, and an infrastructure for executing the deployment strategy. Tawlar et al. [63] have classified the approaches for describing deployment strategies in four main classes: manual, script-, language-, and model-based approaches. Among the others, model-based approaches have gained a lot of interest because they are able to control and evolve an SBA while it is running. Notable is the work of Arnold et al [64] suggesting an approach for Pattern Based deployment. On demand deployment requires the search of application in centralized or distributed repositories, and the installation and the configuration before the operation. A view of the service deployment is shown in the Figure 3.6.

Not only all software components that are part of services have to be installed. Their deployment also requires the associated description to be published on some registries. Thus, deployment is strictly connected to Service Description, Service Publication and Service Operation.

3.5 Operation and Management

In this section the issues related to the phase of Operation and Management will be discussed. More specifically, in the world of Web services, distributed management becomes a clear requirement because the growing complexity of global Web services brings together large numbers of services, suppliers and technologies, all with potentially different performance requirements. However, many existing system management infrastructures do not support service-level agreement reporting or collect specific service information from SBAs for troubleshooting purposes. Furthermore, existing management standards primarily focus on data collection and not on supporting rich management applications for the adaptive infrastructure required by Web services [6].

Web services and SBAs management provides the necessary infrastructure to help enterprises monitor, optimize, and control the Web services infrastructure. A services management system provides

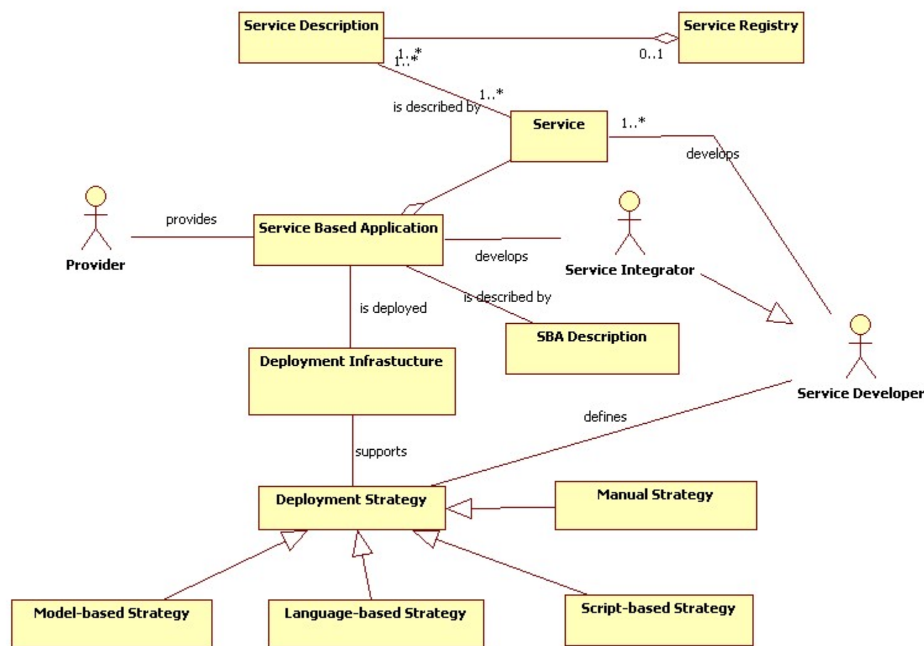

Figure 3.6: The service publication/deployment diagram

visibility into the services runtime environment to enable: monitoring of availability, accessibility, performance of services SLA-compliance tracking and error detection, resolution, and auditing.

OASIS Web Services Distributed Management [65] is a key standard for services management. It allows exposing management functionality in a reusable way through two specifications: one for Management Using Web Services (MUWS) and the other for Management Of Web Services (MOWS). The MUWS specification provides a framework that defines how to represent and access the manageability interfaces of resources as Web services. MOWS builds on MUWS to define how to manage a Web service as a resource. It defines WSDL interfaces, which allows management events and metrics to be exposed, queried, and controlled by a broad range of management tools.

During the operation phase and the execution of its functionalities, the system's behavior must be compliant to the QoS stated in the SLA. An important aspect to guarantee the respect of the SLA is the monitoring of the service state during its execution; more details about monitoring activity can be found in the deliverable CD-JRA-1.2.2. Service operation requires a service governance (see section 3.7.1) ensuring that the architecture is operating as expected maintaining a certain QoS level (Figure 3.7). Of particular interest for the service operation is the service fault, since the identification of service faults permits the triggering of adaptation mechanisms needed to adapt SBAs.

Service Fault

Internet services represent an important class of systems requiring 24x7 availability. Moreover they must guarantee the QoS levels stated in the SLA contract between consumer and provider. Oppenheimer et al. [66] analyzed failure reports from large-scale Internet services in order to identify the major factors contributing to user-visible failures, evaluate the (potential) effectiveness of various techniques for preventing and mitigating service failure, and build a fault model for service-level dependability. Their results indicate that the main contributors to user-visible failures are operator error and network problems, and that online testing and more thoroughly exposing and handling component failures would reduce failure rates in some cases. Referring to the IEEE standard terminology for definitions of failures

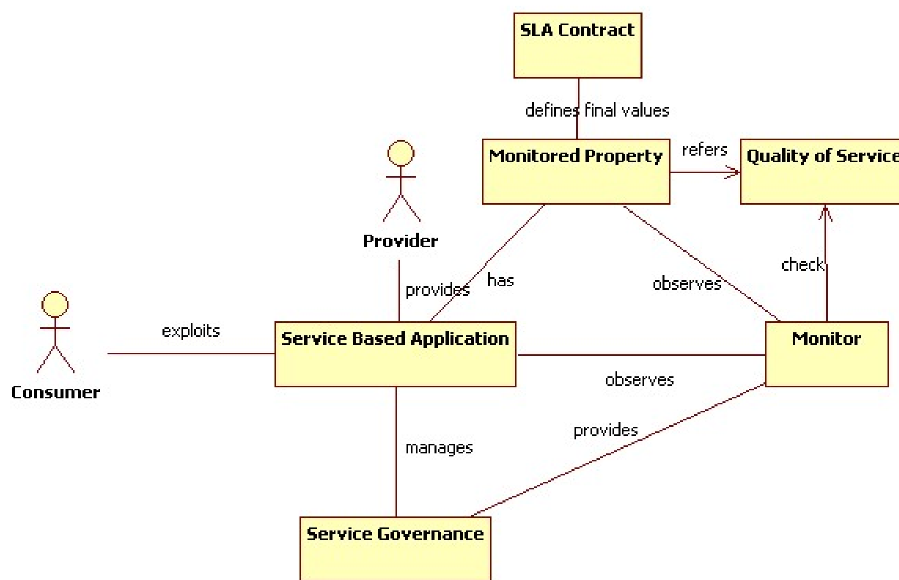

Figure 3.7: The operation and management diagram

and faults [67] we find that *a failure is the inability of a system or component to perform its required functions within specified performance requirements*. Moreover *a fault is (1) a defect in a hardware device or component; (2) an incorrect step, process, or data definition in a computer program*. Figure 3.8 represents the service fault diagram. A service can produce, during execution, a fault. The nature of the fault may be different depending on a wide variety of causes. A fault is an observable event in the service execution that can lead to an erroneous state, and, as consequence, a failure. By observing the system it is possible to discover the occurrence of a fault (Fault Detection activity). The output of this activity are alarms; such events signal the occurrence of a failure, i.e., of a discrepancy between the delivered service and the correct one. Alarms are generally implemented in software using Exceptions. Detecting a fault means only discovering the occurrence of a fault; to know the nature and the cause of the fault its identification is needed; such activity requires a process of diagnosis.

The aim is to achieve the fault tolerance for the architecture: fault tolerance is the ability of an application to provide valid operation after a fault. The application is returned to a consistent state, for example using a checkpointing mechanism. Fault tolerance is considerably more difficult for distributed applications, composed by several process communicating among themselves. Moreover in SBA a single process may be part of multiple applications. Dialani et al. [68] proposed a framework able to offer a method of decoupling the local and global fault recovery mechanisms. In a different way, to achieve fault tolerance in SOA, Santos et al. proposed an approach for deployment of the active replication technique; they presented an engine able to detect and recovery fault and invoke concurrently service replicas [69].

Since the nature of a fault depends on a lot of causes, some authors [70] proposed a classification of the Web service faults distinguishing them in three levels: *infrastructural and middleware, Web service* and *Web application* level faults. Infrastructure middleware level faults are caused by failure in the underlying hardware or network: this type of fault makes it impossible to use the Web service or provide the expected QoS. Among Web Service faults they proposed the classification into Web Service execution faults (raising during invocation or execution of Web service) and coordination faults (resulting by the composition of Web Services). Finally the application level faults are related to the Web applications based on Web services. The same authors proposed some mechanism of recovery action at Web service and Web application level in order to guarantee self-healing properties of Web Services. In particular,

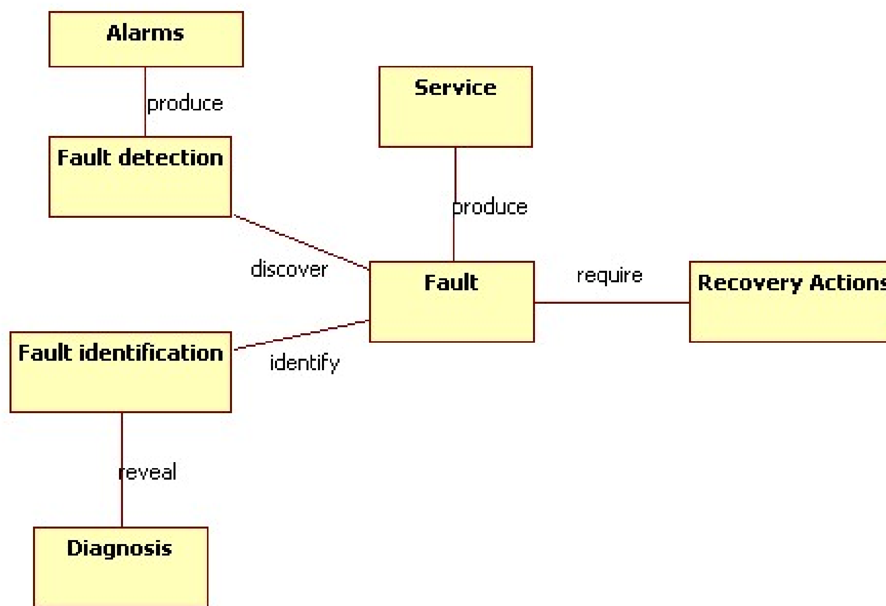

Figure 3.8: The service fault diagram

after diagnosing a fault, adaptable Web Services are able to perform recovery actions and restore the correct state: recovery actions may be reactive (recovery of the running service) and proactive (data mining techniques executed in an off-line mode). Substitution of unavailable services, completion of missing parameters in the input message causing a fault and retry the invocation of an unavailable service until it return available are some of the proposed recovery actions.

Faults can also be related to non-functional behavior of Web services including SLA and QoS agreement [1]. SLAs are used to ensure to the consumer a certain QoS during service execution. Even a violation of the contract raises a SLA disagreement fault.

3.6 Adaptation Life-cycle Phases

Differently from classical SBAs, the distinguishing feature of the adaptable SBAs is the support for accommodating and managing various changes occurring in the application or in its context. This capability extends the traditional view on the Service-Based Application and requires the following two functionalities to become the core elements of the application life-cycle: monitoring and adaptation (Figure 3.9). In more details, the taxonomy of the adaptation of monitoring principles and concepts, as well as the mechanisms from different areas and domains are discussed in deliverable CD-JRA-1.2.2 [34].

In a broad sense, *monitoring* is a process of collecting relevant information in order to evaluate properties of interest over SBA and report corresponding events. As it follows from the diagram, monitoring observes either the application (more precisely, various properties of an SBA instance, the whole class of instances, and/or its evolution) or its context (contextual properties of an instance or of the whole application). When the events reported by the monitoring represent critical deviations from the expected functionality, evolution, or context of SBA, the latter should be adapted and therefore adaptation is triggered.

Adaptation is a process of modifying a Service-Based Application in order to satisfy new requirements and to fit new situations dictated by the environment. It correponds to the adaptive category of the maintenance activity that will be described in detail in Chapter 4. This general definition becomes more

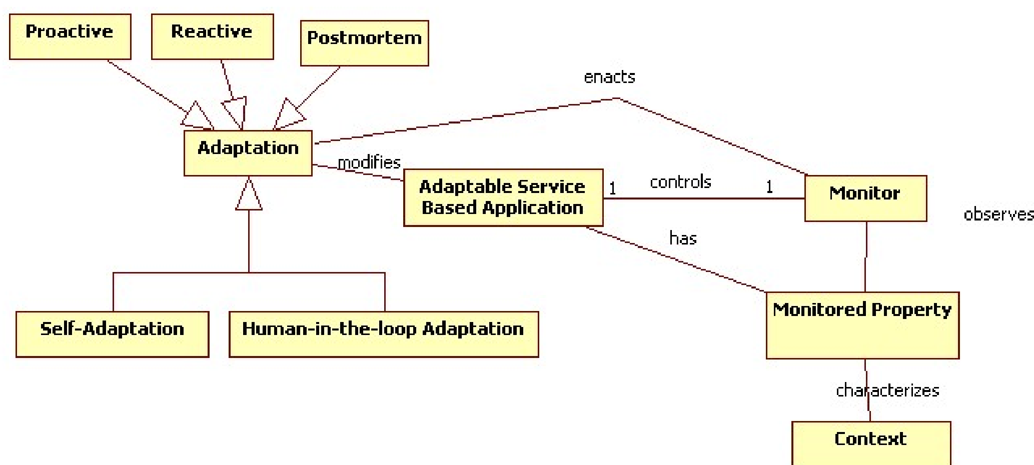

Figure 3.9: The adaptation diagram

concrete when we consider different forms of adaptation (see Figure 3.9): *Proactive* (to prevent future problems proactively identifying and handling their sources), *Reactive*(to handle faults and recover from problems reported during execution of an SBA instance or a set of instances), and *Postmortem* (to modify (or evolve) the system at design time or when it is stopped). With respect to the human involvement, as highlighted in the figure, we distinguish the following two extreme types of adaptation: self-adaptation and human-in-the-loop adaptation. *Self-adaptation* is an adaptation process that is executed without any external human intervention. In this case all adaptation steps, decisions, and actions are performed by the SBA autonomously. This also assumes that all the necessary mechanisms to enact adaptation strategies are built into the application. When the adaptation process assumes any form of human intervention, one deals with *human-in-the-loop adaptation*. This intervention may have different forms and take place at the different phases of the adaptation cycle.

As it is shown in Figure 1.1, the adaptation cycle consists of the following principle steps (for information on related work in these areas the reader should refer to the WP-JRA-1.2 deliverable):

- decide whether the SBA adaptation is needed (*Identify Adaptation Requirements*);
- decide how the system should be adapted (*Identify Adaptation Strategies*);
- modify the application (*Enact Adaptation*).

The ability to initiate this process relies, however, on the ability to identify critical discrepancies between the expected (or desired) state, execution, and evolution of SBAs and the actual ones. For this reason, monitoring becomes an essential component of the adaptation process.

Identify Adaptation Requirements phase

The decision on the necessity for SBA to adapt is based on the information about the execution, evolution and context of SBA provided by monitoring. There are two possible ways to make such a decision. In the first case, the monitoring requirements are derived from the adaptation requirements, and the appropriate monitoring properties represent severe problems, contextual changes or other type of discrepancies that are critical from the adaptation perspective. These properties are observed by the monitors, and when the corresponding events are detected, the need for adaptation is automatically triggered. In the second case, the process requires human involvement: based on the monitored information, the user (being end user, system integrator, application manager, etc.) makes a decision on the need for adaptation.

Identify Adaptation Strategies phase

When the adaptation requirements are instantiated, the corresponding adaptation strategies should be identified and selected. In Section 3.2.2 we have already presented a set of strategies applicable to various forms of SBA adaptation, including service substitution or re-negotiation of their SLAs, reconfiguring SBA or recomposing services, execute specific recovery or compensation actions, and even re-planning the underlying business process. Different adaptation strategies may refer to different functional SBA layers, may be predefined or created dynamically, may follow different methodologies and specified in different ways. A more detailed taxonomy of these concepts is presented in deliverable CD-JRA-1.2.2 [34].

An important aspect for the adaptation cycle is how a particular strategy is defined and selected. As in the case of adaptation requirements, this may or may not require human involvement. If it does not require human intervention, the selection is made by the SBA or the execution platform, based on some predefined decision mechanisms and the current information derived from monitors. In the opposite case, the role of the user can be to choose one or another alternative among those proposed by the adaptation framework.

Enact Adaptation phase

After the adaptation strategy is identified and chosen, the corresponding adaptation mechanisms are activated in order to implement the strategy and to execute corresponding adaptation activities. For the strategies mentioned above the following mechanisms are usually considered:

- automated *service discovery* and dynamic *binding* mechanisms are crucial for the realization of such adaptation strategies as service substitution, re-composition and re-configuration; (automated) *SLA negotiation* frameworks and infrastructures are necessary for the realization of re-negotiation strategy,
- *automated service composition* techniques and mechanisms are necessary for the re-composition and re-planning techniques (when the latter is done in autonomous mode),
- *design time adaptation tool* support may be necessary in order to perform manual, design-time adaptation of SBA or its constituent parts when a re-planning strategy is achieved through re-design of SBA. Such tools may include, e.g., various frameworks for designing and generating adapters for constituent services [71, 72], tools supporting customization of the process models [73], etc.

Also in this case the process may involve the users (e.g., to select a particular realization, to provide additional information and decisions, or to perform the adaptation manually through re-designing the application or components) or may be done autonomously.

Depending on the strategy, the adaptation process may involve other phases of the SBA life-cycle such as quality assurance and deployment.

3.7 Cross-cutting Concerns

The previous sections presented methodologies and processes for each the phases of the service life cycle of Figure 1.1. These sections discuss issues that spread beyond one individual phase in the life cycle, affecting in some cases all the life cycle of services like service governance, quality assurance of SBAs, service discovery and service level agreement negotiation.

3.7.1 Service Governance

A significant challenge to widespread SOA adoption is for SOAs to deliver value. To achieve this, there must be control in areas ranging from how a cross-organizational end-to-end business process that is composed out of a variety of service fragments is built and deployed, how QoS is enforced, proven and demonstrated to service consumers, to granular items such as XSD schemas and WSDL creation. This requires efficient *SOA governance*.

Prior to describing SOA governance we shall describe the meaning of *IT governance* as SOA governance stems from and is deeply rooted in IT governance [74]. IT governance is a formalization of the structured relationships, procedures and policies that ensure the IT functions in an organization support and are aligned to business functions. IT governance aligns IT activities with the goals of the organization as whole and includes the decision-making rights associated with IT investment, as well as the policies, practices and processes used to measure and control the way IT decisions are prioritized and executed [75].

The IT Governance Institute (http://www.itgi.org/) has established a value IT framework that consists of a set of guiding principles, and a number of processes conforming to those principles, which are further defined as a suite of key management practices. ITG recommends these guiding principles to be applied in terms of three core processes: value governance, portfolio management and investment management. The goal of value governance is to optimize the value of an organization's IT-enabled investments by establishing the governance, monitoring and control framework, providing strategic direction for the investments and defining the investment portfolio characteristics. The goal of portfolio management is to ensure that an organization's overall portfolio of IT-enabled investments is aligned with, and contributing optimal value to the organization's strategic objectives by establishing and managing resource profiles, defining investment thresholds, evaluating, prioritizing and selecting, managing the overall portfolio, monitoring, and reporting on portfolio performance. Finally, the goal of investment management is to ensure that an organization's individual IT-enabled investment programs deliver optimal value at an affordable cost with a known and acceptable level of risk by identifying business requirements, analyzing the alternatives, assigning clear accountability and ownership, managing the program through its full economic life cycle, and so forth.

SOA governance has to oversee the entire life cycle of an enterprise service portfolio in order to identify, specify, create, and deploy enterprise services, as well as to oversee their proper maintenance and growth [76].

SOA governance is an extension of IT governance and guiding principles, such as the ones described above, which focus is on the life cycle of services and is designed to enable enterprises to maximize business benefits of SOA such as increased process flexibility, improved responsiveness, and reduced IT maintenance costs. SOA governance refers to the organization, process, policies and metrics that are required to manage an SOA successfully [77]. In particular, SOA governance is a formalization of the structured relationships, procedures and policies that ensure that the IT functions in an organization support and are aligned to business functions, with a specific focus on the life cycle of services.

Services that flow between enterprises have defined owners with established ownership and governance responsibilities, including gathering requirements, design, development, deployment, and operations management for any mission critical or revenue generating service.

To achieve its stated objectives and support the enterprise's business objectives on strategic, functional, and operational levels, SOA governance provides a well-defined structure. It defines the rules, processes, metrics, and organizational constructs needed for effective planning, decision-making, steering, and control of the SOA engagement to meet the business requirements of an enterprise and its customers.

SOA governance introduces the notion of business domain ownership, where domains are managed sets of services sharing some business context to guarantee that services fulfil their functional and QoS objectives both within the context of a business unit and the enterprise's within which they operate [6]. Two different governance models are possible [6]:

1. *Central governance:* With central governance, the governing body within an enterprise has representation from each business domain as well as from independent parties that do not have direct responsibility for any of the service domains. There is also representation from the different business units in the organization and subject matter experts who can talk to the developers who implement key technological components of the services solution. The central governance council reviews any additions or deletions to the list of services, along with changes to existing services, before authorizing the implementation of such changes. Central governance suits an entire enterprise.

2. *Federated governance:* With federated governance each business unit has autonomous control over how it provides the services within its own enterprise. This requires a functional service domain approach. A central governance committee can provide guidelines and standards to different teams. This committee has advisory role only in the sense that it makes only recommendations and it does not have to authorize changes to the existing service infrastructure within any business unit. Federated governance suits enterprise chains better.

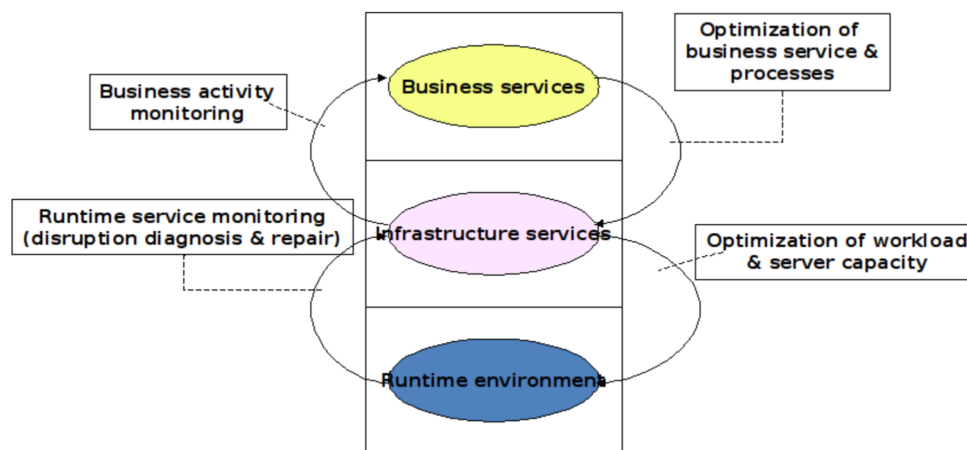

Figure 3.10: Developing and managing SBAs

Figure 3.10 illustrates the usual stratification in runtime environment, infrastructure services and business services and highlights the importance that monitoring facilities play in SOA governance. Resource and business process optimization are also highlighted.

As mentioned above, the concept of SOA governance comprises all the activity needed to exercise control over services in an SOA. The focus is on those resources to be leveraged for SOA to deliver value to the business; it involves many phases of a service architecture lifecycle, including specification, deployment and evolution. SOA governance is about ensuring and validating that assets and artifacts within the architecture are operating as expected and maintaining a certain level of quality. So, it has to offer features to monitor execution, check the policies and handle the exceptions (see the class diagram in figure 3.11).

3.7.2 Quality Assurance of SBAs

The issue of Quality Assurance (QA) in SBAs is covered more extensively in work package JRA-1.3. Here we are summarizing some of the major points of QA for SBAs as identified in deliverables PO-JRA-1.1.1 [4] and PO-JRA-1.3.1 [46] that illustrate the need for QA throughout all the phases of the life-cycle.

More specifically, three major approaches have been identified in the literature for SBA QA:

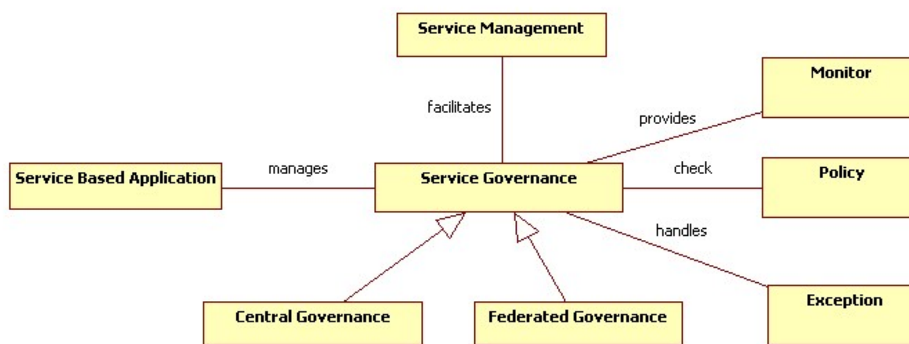

Figure 3.11: The governance of Service Based applications diagram

1. Static Analysis: In a narrower sense, static analysis "...is the systematic examination of program structure for the purpose of showing that certain properties are true, regardless of the execution path the program may take." [78] In a broader sense, static analysis can be extended to documents at all stages of the software life cycle, e.g. it can be used to analyse requirements documents such as goal models and scenarios and design documents such as workflow models and BPEL specifications. Static analysis techniques include for instance model checking approaches, data flow analysis, symbolic execution and type checking. Since static analysis techniques can be applied at any life cycle stage they complement testing and monitoring approaches described below.

2. Testing: "...testing entails executing a program and examining the results produced." [78] Testing a software system or an SBA requires test data, which are fed into the system. The resulting outputs are than compared to the expected outputs. An error (or defect) results if the actual outputs do not fit the expected outputs. In SBAs these defects are due to services or to service compositions, e.g. a wrong sequence of service requests in a BPEL specification.

3. Monitoring: The purpose of monitoring in the software engineering domain is to "...determine whether the current execution [of the software] preserves specified properties; thus, monitoring can be used to provide additional defence against catastrophic failure..." [79] In SOAs monitoring can be used to observe the status of SBAs - as in traditional software engineering - and services. Monitoring of services may lead to the adaptation of the SBA, e.g. when one ore more services are not available. The current state of the art of monitoring is described in detail in the deliverable PO-JRA-1.2.1.

In PO-JRA-1.3.1 [46] the authors distinguish between the design and the operation of an SBA: these phases match the life cycle model depicted in figure 1.1 as the design represents the right and the operation represents the left cycle. There is a rich body of quality assurance knowledge (see also [4] and [46] for a review of these approaches), which was either developed particularly for SBAs or was adapted from traditional software engineering. The challenge is to combine the results of these approaches with engineering principles, techniques and methods, e.g. to achieve an automated adaptation of SBAs due to monitoring results or to closely align requirements with current service provision. Figure 3.12 summarizes the Quality Assurance for SBAs: Monitor is used to check the compliance of the behavior exposed by an SBA during execution and its expected QoS. If some deviation is detected, the monitor could enact some adaptation mechanism to correct the behavior.

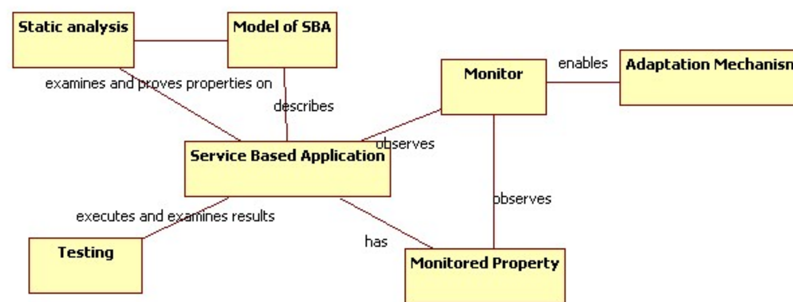

Figure 3.12: The Quality Assurance diagram

3.7.3 Service Discovery

The Service Discovery is an important aspect in Service Oriented Computing. The process of service discovery requires locating the services satisfying user requirements and returning the most relevant ones for the consumer. In other words, service discovery is the matching of the needs of a service requestor with the offerings of service providers.

The continuous growth of the number of services published in the Web makes the process very hard. A key aspect of the service-oriented architecture approach is that the services advertise themselves using directory or lookup services so clients can find them. Consumers need to know only limited information about the service. Like a caller using telephone white pages, a consumer application looks up the desired service in some directory, which returns the associated service provider information. The consumer then uses this information to interact with the provider. Performing a name lookup on an implementation of the Java Naming and Directory Interface, for example, returns a Java object that the caller can use to invoke the named service.

Web services enforce the paradigm of distributed computing enabling enterprise-wide interoperability: integration of the services requires the localization and the purchase of the needed services. Existing Universal Description Discovery Integration (UDDI) [54] technology uses a central server to store information about registered Web services; the centralized approach becomes unsuitable managing large distributed system. WSDL (Web Service Description Language) provides descriptions for Web services [48], containing the service interface, specifying inputs and the outputs of service operations. But these descriptions are purely syntactic: the problem with syntactical information is that the semantics implied by the information provider are not explicit, leading to possible misinterpretation by the users.

Among the most used service discovery approaches, important is the keyword-based (syntactic) discovery mechanism; the limit of this approach is that it doesn't consider the semantic of the requestor goals, service and context, retrieving objects whose descriptions contain keywords from the user's request. This approach can lead to the individuation of services, often not expected by the consumers: for example the query keyword might be syntactically equivalent but semantically different from the terms in the object descriptions; moreover this approach doesn't consider the relations between the keywords. A solution is represented by ontology-based discovery approaches: the retrieval is based on semantic information rather than keywords. Improving Web services discovery requires explicating the semantics of both the service provider and the service requestor. Shoujian et al.[80] proposed an ontology-based approach to capture real world knowledge for a finer granularity annotation of Web services. Moreover [81] proposed a more sophisticated approach using Probabilistic Latent Semantic Analysis (PLSA) to capture semantic concepts hidden behind the term constituting the user query.

A lot of effort is spent to automatize discovery: automatic service discovery requires automated matching of semantic service descriptions or, in worse cases, a composition of them [82]. Figure 3.13 focuses on the *Service Request* and the process of *Service Discovery*. A *Service Consumer* expresses

one or more Service Requests in order to discover *Concrete Services* that can serve its requests and satisfy its needs. Service discovery is usually executed at least in three different moments, related to different phases in the lifecycle of Figure 1.1 : 1) when the requirements for a new system are gathered (*Early Discovery*) (Requirement Engineering activity in Requirement Engineering and Design Phase in figure 1.1), 2) when the system is being designed and new specific needs for services are identified (*Design Time Discovery*) (Design activity in Requirement Engineering and Design Phase in figure 1.1), or 3) when the system is running and new services need to be discovered to replace the ones that the system is currently using (*Run-Time Discovery*) (Operation, management and Quality Assurance Phase in figure 1.1). The latter type of discovery is required during adaptation enactment (see section 3.6). Some researches attempt to optimize the runtime service discovery process [83], using the information gathered during design time service discovery as a sort of cache.

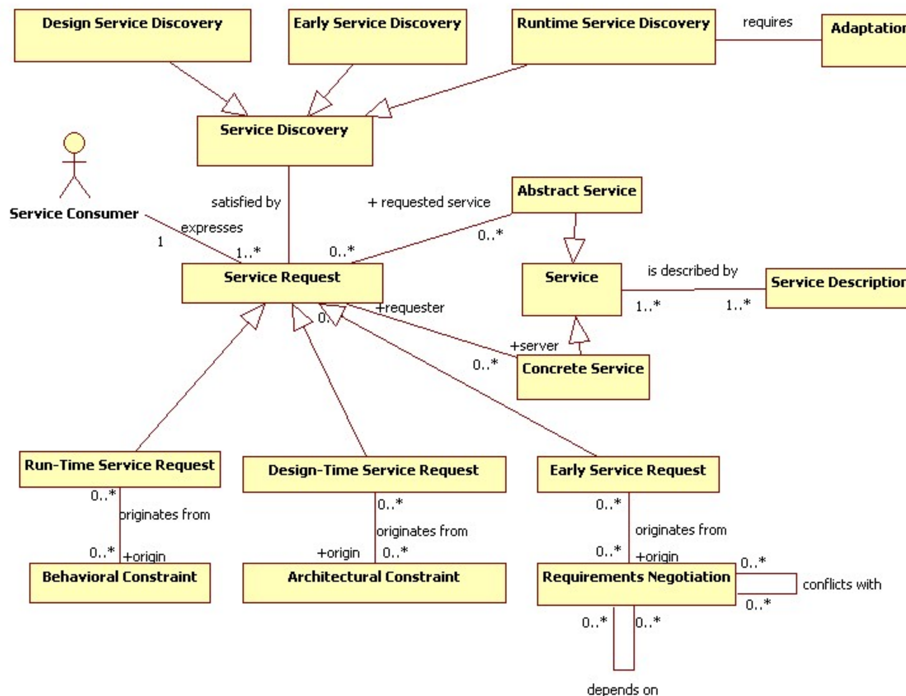

Figure 3.13: The service discovery diagram

3.7.4 Service Level Agreement Negotiation

Service Level Agreements (SLAs) are contracts between a service provider and their customers that describe the service, terms, guarantees, responsibilities and level of the service to be provided. They have been widely used by network operators and their customers. The process that leads to the definition of a SLA between consumer and provider is called an SLA Negotiation. The SLA Negotation cannot be referred to a specific phase of the service lifecycle because an SLA can be negotiated either at design time or at runtime. For example, during service execution an SLA can be negotiated or even re-negotiated if the quality parameters defined in the previous SLA aren't satisfied. If the provider is unable to meet the SLA conditions, instead of stopping the service provisioning, the provider and consumer can decide to re-negotiate the SLA. The end of the process of SLA negotiation consists of the stipulation of a contract in the form of an SLA: this contract contains what user expects from service execution, and what the provider guarantees. Negotiation is a widely studied topic and there are numerous publications addressing different aspects, e.g. [84] is a general purpose negotiation journal, Briquet et al. [85] offer a survey

about negotiation in distributed resource management systems, while Parkin et al. [86] discuss aspects of service negotiation in the Grid. Figure 3.14 focuses on the entities and the activities characterizing the process of SLA Negotiation. The negotiation process consists of two or more *Negotiation Agents*, each acting on behalf of a *Service Provider* or a *Service Consumer*, formulating, exchanging and evaluating a number of *SLA Proposal*s in order to reach an *SLA Contract* for the provision/consumption of a service. A SLA Proposal can be an *SLA Offer* or an *SLA Request* that a Negotiation Agent formulates enacting a certain Strategy. An SLA Proposal specifies negotiation values for a number of *Service Properties*, such as QoS attributes. When the negotiation process leads to an agreement between the involved parties, an SLA Contract enclosing the agreed SLA Proposal is subscribed between these subjects.

More details on SLAs and Negotiation are provided in PO-JRA-1.3.1 [46] and CD-JRA-1.3.2 [47].

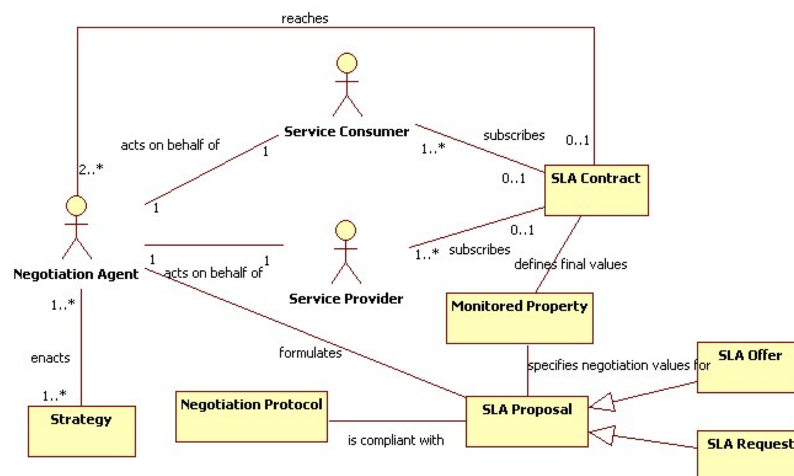

Figure 3.14: The Service Level Agreement negotiation diagram

3.8 Summary

In the previous sections we have presented a reference life-cycle model (figure 1.1). This model can be viewed in two stages: the development stage on the right hand side, and the adaptation stage on the left-hand side. In section 3.1 we presented various SBA life-cycles and compared them with the reference life-cycle model. It is noticeable that each of these can cope with the development of a service, but we question whether the adaptation phase is identified and included in these models. What became evident is that the life-cycle of the reference service model clearly requires an additional adaptation phase and what is needed from our S-Cube research project is the development of processes for that phase. However, we also need to ensure that the requirements engineering, design, construction, deployment, provisioning, operation and management phases take the potential adaptation of services into account. Furthermore, special provision has to be taken for cross-phase issues like service governance, SBA quality assurance, service discovery and SLA negotiation between service providers and consumers.

The following chapter discusses more traditional software engineering techniques, and which lessons can be drawn from them for the purposes of SBA engineering.

Chapter 4

Knowledge Model for Relevant Areas of Software Engineering

This section focuses on building a KM of non SOA-specific software engineering, covering "classical" design and development methodologies and issues, some of which have been established as industry-wide accepted standards in the last decades. In order to scope the discussion, and avoid diverting from the goals of this deliverable, the methodologies that were selected to be added to this KM have been chosen on the basis of their applicability or relation with SBA engineering as evidenced by the existing research in the respective field. In that sense, the presented issues are not explicitly mapped to the phases of the life-cycle of Figure 1.1; the actual identification and analysis of these mappings is part of the following deliverables in this workpackage.

More specifically, Section 4.1.1 presents established methods and theories for software process quality and assurance, continuing the discussion on SBA quality assurance that was initiated in the previous chapter. Furthermore, Section 4.1.2 discusses the predecessor of service orientation, i.e., the component-based paradigm; Section 4.1.3 provides some insight into the issue of legacy system re-engineering and how it affects SBA engineering. Consequently, Section 4.1.4 discusses the issues of software maintenance and evolution and how they are related to SBA adaptation. Finally, Section 4.2 summarizes some of the dominant methodologies for business processes in order to illustrate the challenges and expectations for any SBA methodology that has to be applied in the Business Process Management area.

4.1 Classical Software Engineering

4.1.1 Software Process Quality

Software quality within software engineering is often considered to be only testing. However, the software process community argues that quality should be built into a product, not just 'tested for' at the end of the development process. In this section we discuss the overall concept of software process quality.

Humphrey [87] defines a software process as "the set of tools, methods and practices we use to produce a software product". Paulk et al. [88] expand this definition to "a set of activities, methods, practices and transformations that people use to develop and maintain software and the associated products".

When organisations consider their software process it is usually with a view to improving that process to improve the quality of their product. As an example, improvement of the process can be based on the Plan-Do-Check-Act cycle which is a common technique used in manufacturing quality improvement strategies as shown in Figure 4.1. To be useful, the improvement must be continuous, and the process continually assessed. Specific cycles are stated within some of the process models.

The purpose of implementing software processes within an organisation is to improve the quality of the final product through building in quality throughout the process rather than discovering, either at testing phase or following release, that there are problems with the product.

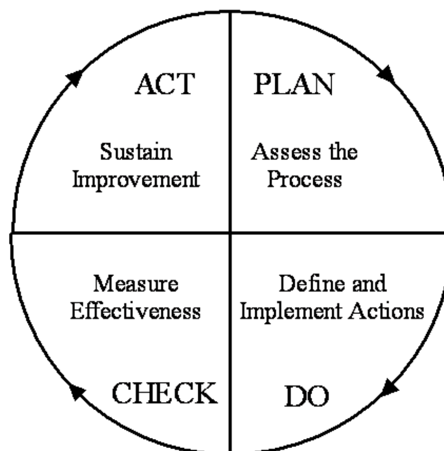

Figure 4.1: Plan-Do-Check-Act for Software Process

There are many proprietary process improvement and assessment frameworks used in industry. Such frameworks normally contain process areas within which specific practices are performed. Two internationally recognised models are ISO/IEC 15504 and the Capability Maturity Model Integrated (CMMI^{TM}) [89, 90]. ISO/IEC 15504 is designed so that other process models can be ratified by the ISO standard. For organisations using CMMI, for example, they can demonstrate a maturity level with respect to both CMMI and within ISO/IEC15504. In the following section, to illustrate the meaning of a process area, we chose to discuss those process areas within the Capability Maturity Model Integrated as within CMMI Version 1.2 [90].

Process Management areas contain the cross-project activities related to defining, planning, deploying, implementing, monitoring, controlling, appraising, measuring, and improving processes. More specifically they are:

Organizational Innovation and Deployment is designed to ensure that incremental and innovative improvements improving the organisation's processes and technologies are implemented.

Organizational Process Definition requires that the process is developed and defined within the organization.

Organizational Process Focus ensures that process improvements are planned and implemented based on an understanding of the strengths and weaknesses of the organisation's process and process assets.

Organizational Process Performance establishes and maintains a quantitative understanding of the organisation's process, focusing on quality and process-performance objectives.

Organizational Training is focused on the development of skills and knowledge of people.

Project Management process areas cover the project management activities related to planning, monitoring, and controlling the project. They are:

Integrated Project Management establishes and manages the project and stakeholders according to a defined process.

Project Monitoring and Control is undertaken to ensure that corrective actions can be taken if and when required.

Project Planning establishes and maintains plans that define project activities.

Quantitative Project Management ensures that the project achieves quality and process-performance levels through quantitative measures.

Risk Management allows the organisation to identify potential risks, and to implement a strategy to mitigate these risks where possible.

Supplier Agreement Management manages the acquisition of product from suppliers.

Engineering process areas cover the development and maintenance activities that are shared across engineering disciplines and are:

Product Integration ensures that the product's sub-components are integrated correctly to provide a final working product.

Requirements Development ensures that customer, product and component requirements are produced correctly.

Requirements Management manages the requirements of the product and components, ensuring that they are consistent with the project plans and the work products.

Technical Solution enables the design, development and implementation of solutions to requirements.

Validation allows the organisation to demonstrate that a product or component fulfils its intended use in its intended environment.

Verification ensures that the work products meet the requirements.

Support process areas cover the activities that support product development and maintenance. The Support process areas address processes that are used in the context of performing other processes. They are:

Causal Analysis and Resolution allows the organisation to identify the causes of defects in products and to prevent their re-occurrence.

Configuration Management implements configuration identification, configuration control, configuration status accounting and configuration audits.

Decision Analysis and Resolution enables the analysis of possible decisions against a formal evaluation process.

Measurement and Analysis enables the development of a measurement capability which supports management information needs.

Process and Product Quality Assurance involves evaluating performance of process and process assets against pre-defined standards and ensuring that non-compliance issues are addressed.

While there have been arguments that implementing planned processes decrease rather than increase the efficiency of the software development process [91, 92, 93] there is also evidence that there have been increases in productivity and efficiency due to the implementation of planned processes [94, 95, 96, 97, 98].

Agile Development is a software process which has gained recognition in recent years. Having introduced concepts such as Scrum, Test Driven Development (TDD) and Extreme Programming (XP), it is distinctly different. The agile approach thrives on the lack of stable requirements and uses small self-managed teams to frequently produce reliable software that meets customer requirements.

The reported success of the use of agile development was instantaneous [99]. Key benefits reported include the faster delivery of higher quality products that better matched customer requirements due to their close involvement throughout the project. Leszak et al. [100] have argued that the transition to agile methodologies was initiated as a way of achieving a positive return on investment in quality early in the development life cycle. However, not all reports of these agile development techniques described positive experiences [101]. Despite the fact that they are "simple" and "quick" [102], most are very difficult to get right and require extensive training, discipline and managerial support.

The ever increasing number of agile methods that are available also presents a problem: not every technique is suitable for every type of project. This factor must be given serious consideration before a specific development methodology is chosen for a project.

When researching software process within S-Cube, our interest is in establishing how software quality assurance can be implemented during the development of services. In particular, we are interested in markets which require specific processes to be in place.

We have noted that, in particular industries, such as the Medical Device industry, through governance by the Food and Drugs Administration (FDA) and International Standards Organisation (ISO), and the Automotive industry, who follow Automotive SPICE (derivative of ISO/IEC 15504), documented processes are still required. The Financial sector has also commenced an initiative to implement Banking SPICE as they also have to deal with regulations such as Sarbanes-Oxley.

While there may be an argument for service developers not to consider the implementation of software processes due to their restrictiveness, the community needs to consider that software development within specific industries such as those mentioned above is a growth area. For example, the cycle on the right-hand side of Figure 1.1 does not currently exist within software engineering process models, and, therefore, existing models need to be developed to ensure that service-based software can be used successfully within the regulated industry. In doing this, organisations can work on becoming players within these markets and software developers need to become process aware, seriously considering how quality can be improved through the implementation of software processes for services.

Quality Assurance

Software process models are designed to ensure that the quality of the product is built in from the start of software development, therefore, in this section, we discuss further the service life-cycles already discussed in section 3.1 from the perspective of software engineering quality. It is imperative that, for the future, as services move into regulated industry in particular, good quality assurance systems are implemented. While there are a number of such quality models available we chose to look specifically at the Capability Maturity Model Integrated™(CMMI™[90]) as it is an accepted exemplar containing process areas relevant to the development of systems and software. The CMMI™contains twenty-two process areas which focus on Process Management, Project Management, Engineering and Support Process. While it is imperative that the Engineering process areas are implemented successfully to develop product, it is equally important that, for a quantitatively-managed, defined and repeatable life-cycle, process areas under the other three headings are also implemented.

1. The SLDC appears to cover all the aspects of the CMMI technical solution process areas but seems lacking in some of the other process areas of the model, such as project management and process management.

2. In order to compare the RUP for SOA framework to the CMMI capability model we will first have to look at its components. The RUP framework consists of nine disciplines, six engineering disciplines and three support disciplines. The engineering disciplines are Business Modelling, Requirements, Analysis and Design, Implementation, Testing and Deployment. The remaining three support disciplines are Configuration Management, Project Management and Development Environment. When we compare the RUP for SOA to the CMMI model, the RUP for SOA activities

work flow seem to cover the majority of the CMMI process areas. The RUP model however, does not implement most of the support process areas or some of the important project management processes such as Supplier Agreement management. In a survey carried out by the Software Engineering Institute, there was a comprehensive comparison made between RUP and CMMI, which highlighted some of these weaknesses [103]. [104] discusses the implementation of new process elements that allow RUP to overcome these weaknesses.

3. When we compare SOMA and CMMI it is evident that SOMA puts sufficient emphasis on an organisation's processes and it also covers the software's engineering processes in detail. SOMA does not however put a lot of emphasis on project management or the support processes required to deliver software.

4. The comparison between SOAD and CMMI bears resemblance to the comparison between CMMI and RUP for SOA. SOAD puts a lot of emphasis on process management and SOA engineering processes; however there are still gaps in the support process and project management aspects. In addition to that, the fact that SOAD is not yet fully defined as a process for delivering quality SOA based applications, makes it appear even further from being able to provide compatibility with the CMMI model.

5. When we compare ASTRO and CMMI, we can see that the ASTRO tools are primarily focused on using business process input in the form of BPEL to generate a technical solution. The WS-animator tool, which is an Eclipse™Integrated Development Enjoinment plug-in, can be used in order to visually execute the business process. This can be used to verify and validate the generated business process. The ASTRO methodology makes no attempt to provide support processes or project management techniques.

6. Many aspects of the BEA lifecycle are compatible with the CMMI model. The BEA lifecycle also contains many SOA centric components that cannot me measured using the CMMI. The BEA lifecycle describes in detail the processes around creating, composing and reusing service components. This however seems at the expense of describing a lot of the CMMI key process areas in the process management, project management and support process disciplines.

Summary of Lifecycles vs. CMMI™

All of the lifecycles we have looked at appear to have been developed with varying goals in mind. Some of the lifecycles such as RUP for SOA and the BEA lifecycle make attempts to cover all of the required lifecycles stages to analyse, design, build, test, deploy and monitor service based applications. On the other hand, the ASTRO life cycle focuses on combining and orchestrating third party web services. Figure 4.2 shows the varying levels of CMMI compatibility that exists between each of the life cycles and CMMI.

At a glance it appears that most of the life cycles are more focused on the technical engineering process areas than any of the other process area. In order to make these lifecycles more compatible with the software engineering view of quality they will need to focus more on the areas of support and project management in particular. In addition, direct implementation of a software process model such as the CMMI™into services development does not take into account the left-hand side of the reference lifecycle shown in Figure 1.1. Software process models will need to be adjusted to cope with the adaptation phases of the reference life-cycle. This will be an interesting area of research within the S-Cube project.

4.1.2 Component-Based Software Engineering

In the world of software engineering, software reuse has long been one of the major issues; it is seen as the key for increased productivity, improved reliability, and ease of maintenance. The development

	SOA Lifecycle					
	0	1	2	3	4	5
CMMI Process Area	SLDC	RUP for SOA	SOMA	SOAD	ASTRO	BEA Services Lifecycle
Process Management						
Organizational Innovation and Deployment		x	x	x	x	
Organizational Process Definition		x	x	x	x	x
Organizational Process Focus		x	x	x	x	x
Organizational Process Performance		x		x		
Organizational Training						
Project Management						
Integrated Project Management						
Project Monitoring and Control		x	x	x	x	x
Project Planning	x	x		x		x
Quantitative Project Management						
Risk Management		x		x		
Supplier Agreement Management						
Engineering						
Product Integration	x	x	x	x	x	x
Requirements Development	x	x	x	x		x
Requirements Management	x	x	x	x		x
Technical Solution	x	x	x	x	x	x
Validation	x	x	x	x	x	x
Verification	x	x	x	x	x	x
Support process						
Causal Analysis and Resolution						
Configuration Management	x	x		x		
Decision Analysis and Resolution						x
Measurement and Analysis	x		x			x
Process and Product Quality Assurance	x					x

Figure 4.2: SBA Lifecycles vs. CMMI

of software starting from existing components draws on analogy with the way that hardware is designed and built, using "off-the-shelf" modules. In fact, Component-Based Software Development (CBSD) approach is based on the idea to develop software systems by selecting appropriate off-the-shelf components and then to assemble them with a well-defined software architecture. The process leading to component based systems is integration-centric rather than development-centric. The idea behind the engineering concept is that components can easily be reused in other systems since they are autonomous units, free of the context in which they are deployed. Components are black box, providing an external interface to their functionality hiding all internal details. CBSD aims to reduce development cost and time to market since ready-made and self-made components can be used and re-used. These Commercial Off-The-Shelf (COTS) components can be made by different developers using different languages and different platforms.

The idea behind CBSD makes the life cycle and software engineering model of CBSD much different from that of the traditional ones. Component-based software systems are developed by selecting various components and assembling them together rather than programming an overall system from scratch; thus the life cycle of component-based software systems is different from that of the traditional software systems. Boehm et al. [20] retain that both the waterfall model and the evolutionary development are unsuitable for COTS-based development. Since in the waterfall model requirements are identified at an earlier stage and the COTS components chosen at a later stage, it's likely to choose COTS components not offering the required features. The evolutionary development on the other hands assumes that additional features can be added if required. However, COTS components cannot be upgraded by one particular development team. The frequent lack of code availability hinders developers to adapt them to their needs. Therefore, Boehm et al. proposed that development models which explicitly take risk into account are more suitable for COTS-based development than the traditional waterfall or evolutionary approaches. A

possible life cycle of component-based software systems consists of the following activities:

- requirements analysis;
- architecture selection, creation, analysis, and evaluation;
- component evaluation, selection, and customization;
- integration;
- system testing;
- and software maintenance.

The focus of CBSD is on composing and assembling components often developed separately, and even independently. Component identification, customization and integration is a crucial activity in the life cycle of component-based systems; component selection addresses the issue of browsing and individuating the component to use satisfying the desired functionality. The selection of COTS products is a challenging process that utilizes and generates a lot of information, aiming to find software components among the components that are previously built. When the number of component grows, the complexity of the choice becomes greater. Hence, management of the existing components is required. For COTS selection activity, information repositories play a crucial role; repositories contain the object code of the components, and they should have features that allows for convenient access to reusable components and provide reuse functionality such as selection, analysis, adaptation, test, and deployment. Lee et al. [105] proposed a component repository for facilitating EJB (Enterprise JavaBeans) component reuse. An EJB component is available as class files packaged in a Java ARchive (JAR) file. The class files contained in the JAR are separated into interfaces and beans. The beans are designed to execute their business logic through their interfaces. Among the component infrastructure technologies that have been developed, three have become somewhat standardized: OMGs CORBA, Microsoft's Component Object Model (COM) and Distributed COM (DCOM), and Sun's JavaBeans and Enterprise JavaBeans.

Issues in the use of Components Software

Component users develop component-based systems by integrating their applications with independently-developed components. Typically, the source code of the components is not available to the component users. Consequently, traditional program analysis and techniques requiring access to the source code, such as alias analysis, static analysis, control dependence computation, and testing techniques, such as data flow, cannot be applied. One way to perform the analysis without the source code is to analyze relations that hold in the components and the relations caused in the application by the code in the components, but unfortunately these analyses are often too imprecise, and therefore useless. It's interesting to notice that the use of high quality components doesn't guarantee the quality of the resulting component based system, but its quality depends on the quality of its components and the framework and integration process used. Hence, techniques and methods for quality assurance and assessment of a component-based system would be different from those of the traditional software engineering methodology [106], requiring adaptation to this context. The Quality Assurance (QA) of the overall system is a critical issue: it is important to certificate the quality of a component and the quality of software systems based on components. To this aim Cai et al. [107] proposed a QA model for component-based software development, which covers both the component QA and the system QA as well as their interactions. One problem that CBSE currently faces is the lack of a universally accepted terminology. Even the most basic entity, a software component, is defined in many different ways; it would be useful to have a clarified and unified terminology. To this aim Lau et al. [108] proposed a taxonomy of component models (JavaBeans, EJB, COM.).

CBSE and SBAs

SBAs are developed composing available functionalities exposed by the services; in this context services can be considered very similar to a reusable components, and approaches developed in CBSE could be adapted to services. However, service-oriented architectures introduce some important issues that need to be considered: in a service-oriented scenario, users acquire just the use of a service without integrating physically it in their applications . Each service of a service-based system ideally represents a component executing a business task and provides an interface that is invoked through a data format and protocol that is understood by all the potential clients of that service. Services can be distributed across organizations and can be reconfigured into new business processes as organizations evolve. Users can discover a Web service by querying a service registry and retrieving the service description of the service they want. The service description contains enough information for the service requestor to bind to the service he wants to use. While in the component repository the physical component is contained, in the service registry only the description of the service is contained: using a service means invoking it and not owning it. Another important difference is about the composition. While component composition is made assembling component using connectors or glue code, service compositions are obtained composing the service descriptions. Consequently, since services are bound only at runtime, the realization of service composition is known only at execution time [109].

4.1.3 Legacy Systems Re-Engineering

Legacy systems constitute the enterprise's heritage of software and hardware systems. Often, legacy systems are relatively old, mainframe-based systems that were optimized to accommodate the memory, disk, and other operational constraints of archaic software and hardware platforms. A vast majority of them is older than twenty years and written in COBOL, PL/I or Assembly/370 using transactions management systems like Customer Information Control System (CICS), although this certainly does not have to be the case.

Legacy systems pose an Amphitryon dilemma for enterprises. On the one hand, enterprises perceive them as bottlenecks to implement new or reinvented business processes as they are notably hard to adapt to new business requirements. Disruptions to these systems, even those as short as a couple of seconds, may cause catastrophic losses. On the other hand, legacy systems typically lock valuable, and in many cases indispensable, business knowledge. This business knowledge contains not only explicit knowledge about business processes, policies and data that is codified in a shared formal language, but also tacit knowledge that is employed implicitly to smoothen daily business operations (e.g., know-how).

Devising a balanced strategy for handling legacy systems and (re-)aligning them with new process requirements has proven a particularly challenging issue. Over the past decades, a number of strategies, methodologies and tools have been touted by the industry as the next silver bullet to overcome the legacy dilemma, ranging from non-intrusive approaches such as screen scraping and legacy wrapping, to more invasive ones like grafting new designs into the outdated parts of the architecture of legacy systems.

Approaches for dealing with Legacy Applications

The following evolution strategies have been proposed during the past decades ([110], [111], [112], [113]): maintenance, modernization, replacement and phase-out. The impact of these strategies on the enterprise applications ranges from minimal to large: maintenance activities entail a contained type of evolution implying marginal changes and extensions, whilst phasing-out is the most disruptive approach involving retirement of (parts of) the legacy systems.

These strategies can be classified as follows:

Continued Maintenance This evolution strategy is applicable in case a legacy system is still relatively well-functioning. As no intrusive changes are accompanied with this strategy, it is by far the most optimal

category of legacy evolution strategies from a cost and risk perspective.

Continued maintenance involves nurturing the application without making fundamental changes to the code and breaking its underlying architecture. The strategy basically comes in three variants ([112], [111]): adaptive maintenance, corrective maintenance, and perfective maintenance. Adaptive maintenance pertains to making minor changes in the system's functionality to ensure that it stays in flux with new business requirements. Besides these activities, maintenance activities can be directed towards eliminating fixed errors in the code (corrective maintenance), and optimizing the code for both the functional and the non-functional requirements (perfective maintenance).

Modernization Modernization through service-enablement of legacy applications and/or repository systems usually becomes desirable after several years of continued maintenance, weakening the technical quality, e.g., flexibility, of the legacy systems.

Basically, legacy system modernization can be achieved in two orthogonal manners. Firstly, legacy system may be renovated by firstly *packaging* them as services (encapsulation), and subsequently *integrating* it with new applications. Some authors refer to this approach to as access/integration in place [114], or black-box modernization [111]. The second, fundamentally different, way of modernizing the legacy system is to *transform* it into a new service-enabled application. Transformation requires a detailed understanding of the legacy system to allow legacy code and data to be converted, whereas integration merely demands abstract knowledge about the external services of a legacy system to integrate them with modern system components. Hence, transformation is considered to be an invasive, and integration a non-invasive strategy.

In particular, transformation of legacy systems constitutes moving a source (the legacy system) to a new, target application. As such, transformation involves the examination and reconstitution of an enterprise information system according to state-of-the-art engineering techniques. Transformation may be realized with (a combination of) several techniques, including: source code translation, program and data restructuring, reverse engineering, and re-targeting. Source code translation involves transforming old code into a new version that is written in another, contemporary programming language, or a newer version of the same language. For example, systems may be converted from COBOL-II into Object-Oriented COBOL [115]. Program restructuring refers to correcting structural flaws to the code, e.g., infinite loops in code whilst data restructuring involves refreshing the data-structure of legacy data files and/or databases. Reverse engineering entails the recovery and analysis of a legacy system to extract an abstract description of the system components and their relationships. Lastly, re-targeting of legacy systems constitutes the transformation of the systems to another platform. An in-depth treatment of these transformation techniques falls outside the scope of this report, but may be found in [116].

To implement the encapsulation and integration strategy it suffices to re-create shallow understanding of the abstract services that are offered by legacy systems, databases or user interfaces. In particular, legacy applications and database repositories may be encapsulated and accessed using adapters, allowing new application components to co-exist with encapsulated legacy systems. Screen scrapers constitute an encapsulation technique to reface archaic, mostly textual, user interfaces.

Replacement Replacement implies rebuilding an application from scratch. Assembling third party components, customizing standard packages (e.g., ERP solutions), in-house development or a mixture of these development practices may be employed to realize this strategy.

Despite the fact that this strategy may at first sight seem very attractive to management as it holds the promise of one shared corporate data model using the newest technologies and leads to a fast discontinuation of redundant applications and repository systems, practice has taught that the replacement strategy bears large risks and many unpredictable pitfalls. Firstly, costly and complex data and code conversions have to be made in order to save past investments in legacy systems. Avoiding expensive downtime of the existing enterprise application is often a difficult hurdle. Secondly, upfront it is usually not possible to guarantee that the new system will outperform the existing application in terms of both functionality

and extra-functional properties such as security and robustness (transactions). Nascent technologies may at first seem to offer tantalizing possibilities, but may not yet be ready for prime-time implementations.

Phase Out The most rigorous enterprise application approach possible is to discontinue the enterprise application. This imposes the supporting business process also to cease to exist.

Service-enabling Legacy Applications

A key challenge of service design is to be able to resurrect and rehabilitate preexisting enterprise assets into modern services that can smoothly operate with novel business processes. In that sense, service-enabling legacy application falls under the category of modernization, as discussed in the previous section. Nevertheless, the challenges and opportunities created by the introduction of SOA into the enterprise level require further examination of the interaction between legacy systems and services. In particular, service enablement of these systems can be achieved through two key techniques:

Firstly, *redevelopment* requires re-engineering the existing asset from scratch, which is unfortunately in many cases a too expensive and risky endeavour, if not unfeasible. This is especially the case for legacy systems that should be modernized into service-oriented systems, having critical characteristics such as continuous availability. Wrapping is a technique to surround existing data, programs and interfaces with new interfaces. Wrapping entails a rather popular approach towards modernization since it is conceptually simple, requiring limited development costs and preserving past investments in pre-existing assets. On the downside, it unfortunately comes with some serious drawbacks such as decreased performance and architectural erosion. Therefore, wrapping as a legacy modernization should be applied carefully, preserving the architecture and maintaining the overall quality of the migrated system. Still, wrapping techniques can be successfully applied, e.g. to export the functionalities of interactive systems towards SOAs [117].

The second modernization technique involves *migration* of the legacy system into an updated and/or extended target software (application) system designed, architected and coded in a modern development and deployment environment. Migration of legacy software has caught a lot of attention in the research and industrial community. E.g., an approach tailored for the migration of supervisory machine control architectures has been presented in [118]. Model-driven architecture migration is defined by transformation rules in terms of patterns associated with the source and target meta-models. Further work at the architecture level, but aimed at the migration towards web-based systems is provided in [119]. Further examples are [120, 121]. These approaches are mainly based on implementation-level architecture reconstruction and/or on the documentation of the technical solution. These techniques assume that the architectural decisions, drivers and rationale are directly accessible by asking people. Unfortunately, and especially for legacy systems that live for decades, have deteriorated, and lack any documentation, such invaluable know-how is either forgotten or has left the company [122]. The necessary know-how must be rebuilt, and existing legacy components must be analyzed in a disciplined way to assess if their functionality can be successfully exposed as services [123].

In current business practices, modernization of pre-existing enterprise assets is leveraged with SOA by placing a thin SOA/WSDL veneer around them, while leaving the underlying code and data untouched. Though this may work for simple and small enterprise applications, this is by no means sufficient for developing large-scale, industrial applications. Unless the existing enterprise assets are naturally suitable for use as a Web service – and most are not – it takes serious thought and redesign to properly deliver an enterprise assets functionality through a Web service.

In ([124], [125]), a meet-in-the-middle legacy modernization methodology is introduced that allows to selectively integrate those parts of legacy applications that are still in line with the modern business policies and objectives, while constructing new services that are not sufficiently supported by existing enterprise assets in general, and legacy applications in particular. The suggested SOA-enabled methodology combines forward engineering of service-enabled business processes with reverse engineering of

legacy applications, for the purpose of *selective* encapsulation/integration. This methodology, named BALES, has been validated and explored by a comprehensive case study that has been drawn from a real-world project at the Dutch Department of Defense that integrated fragments of an existing proprietary material resource planning package into a modern service-enabled application.

4.1.4 Evolution and Maintenance

In the lifecycle of software the development of the first version is only a minor part: evolution and maintenance cover the majority of the software lifecycle. System maintenance is the process of providing service and maintenance activities needed to use the software effectively after it has been delivered. The objectives of system maintenance are to provide an effective product or service to the end-users while correcting faults, improving software performance or other attributes, and adapting the system to a changed environment. All changes for the delivered system should be reflected in the related documents. Lientz and Swanson [126] categorized maintenance activities into four classes (the classification is in the Standard IEEE 610.12[67]):

Adaptive adapting software to changes in the software environment

Perfective managing new or changed user requirements

Corrective fixing errors

Preventive preventing problems in the future

Only corrective is 'traditional' maintenance, the others can be considered software 'evolution'. Often, new technologies are proposed and introduced without consideration of what happens when the software has to be changed. If such innovations are to be exploited successfully, the full lifecycle needs to be addressed, not just the initial development. For example, object oriented (OO) technology was considered to be 'the solution to software maintenance' [127], but empirical evidence shows that OO created its own new maintenance problems, and has to be used with care (e.g. by keeping inheritance under control) to ensure that maintenance is not even more difficult than for traditional systems.

Evolution and Maintenance in CBSE

Development of a software system from commercial components involves new issues in maintenance, evolution, and management system. Component-based systems must deal evolving user requirements, react to failures in the system or to changes in the operation environment, and managers must be able to monitor and control the deployed system. Traditional maintenance involves observing and modifying lines of source code. However, in component-based systems, the primary unit of construction is often a black-box component; the custom developed source code is typically used to tailor the components and integrate them together: maintenance of these systems is restricted to reconfiguring and reintegrating components. Wu and Offut [128] proposed the use of UML diagram, for corrective maintenance of component based software, to represent the changes on a component. The research of Casanova et al. [129] illustrates the use of multi-dimensional libraries to manage the versions; moreover to track the dependecies among the components in a system is proposed to use a configuration model; the use of metrics on the models and the documentation for the component is a support for maintenance and evolution of the components. One of the advantages of using components is that their cost is amortized over many users. Although this provides many advantages, it also means that the system builder is just one of many voices requesting changes or modification to the underlying components. When building a component-based system, system builders must consider maintainability and evolvability during two important phases of construction. The first is during evaluation and selection because the components used to build the system directly impact the maintainability of the system. The second phase is the design of the component infrastructure. The approach used to integrate components determines the flexibility of the system, which directly impacts its evolvability.

Evolution and Maintenance in SOA

Service oriented systems differ from traditional systems so new issues have to be addressed in maintenance and evolution activities. Service oriented systems are applications satisfying the needs of a wide variety of customers and businesses. Examples of their use may be found in B2B and B2C applications, e-learning, and so on. Web services are highly vulnerable and subject to constant change. Hence, they offer a novel challenge to software engineering. From the evolution and maintenance perspective, there are many things that must be examined. The diversity of service provider and consumer often using different programming languages in their applications, the presence of third party services, the high dynamicity of the environment and the shorter cycle releases needed to react to changing business needs open new challenges in the process of maintenance and evolution. In particular, since a service may be shared by different consumers, it must have been identified the responsible for the maintenance, moreover could be happen that different requirements are desired from different business unit. In a service-oriented scenario, users just invoke a service, without having control on it. So, the service provider can decide to maintain the service, and the user could not be aware of that. For example if inputs and outputs are not affected, the service provider could add new features without advertising the changes. However, the change made could alter the service behavior. Moreover, the service provider could optimize the source code of the service causing a variation in the service's non-functional properties. An optimization could improve a non-functional property while worsening another; even an improvement of some Quality of Service (QoS) attributes (e.g., the response time) may not be desirable since it may cause unwanted effects in the whole system behavior. Moreover, any optimization could introduce faults, thus varying the service functional behavior as well.

Obviously the maintenance process has to be slightly adapted to manage investigation of problems and impact analysis which have made across several collaborating applications belonging to different organizations. For example corrective maintenance has in SOA different implications: when an error occurs in a service based application, a maintenance activity could be the selection of a different service in the composition, but this could not be desirable by all the users. Moreover, while roles that are derived from the standard maintenance offer a starting point, a number of tasks in SOA environments are different from those of traditional maintenance tasks and therefore require a different set of roles. Kajko et al. [130] proposed to create a new role of a service owner responsible for evolving and maintaining high level Web services. Finally, it must be considered that the failure of a web service may affect the productivity of other organizations. To this aim Kaijko et al. [130] proposed a general framework (SERVIAM Maintenance Framework) for evolving and maintaining Web services.

Adaptation and Evolution

Evolution is related to the adaptation aspects: adaptation is a process of modifying Service-Based Application in order to satisfy new requirements and to fit new situations dictated by the environment on the basis of Adaptation Strategies designed by the system integrator. If an application is designed to be adaptable, adaptation can be fired by user requirement changes or environment changes without requiring change in the source code. Evolution on the other hand in the context of SBAs [7], refers to the continuous process of development of a service through a series of consistent and unambiguous changes (created by adaptation or the environment of the SBA), expressed through the creation and decomission of different versions of the SBA (see also deliverable CD-JRA-1.2.2 [34] for further discussion on the relationship between adaptation and evolution).

4.2 Business Process Methodologies

A business process methodology is a formal and structured description of a comprehensive approach to organizing companies around processes that can be applied to the incremental design and improvement of business processes. An important characteristic of a business process methodology is that it focuses

only on the design or improvement of a business process, and on measuring processes and redefining processes and not on the development of a software system. A business process methodology is used for business process centric projects ranging from incremental process improvement to full functional transformation.

There are several established business process methodologies, which include the Rummler-Brache-PDL Methodology [131], the Define, Measure, Analyze, Improve, and Control methodology http://www.isixsigma.com/me/dmaic/ and the various methodologies of the various vendors, e.g. ARIS which is heavily focused on software development, but it is also widely used by business process analysts, especially when they are working on company ERP-centric projects.

4.2.1 DMAIC Methodology

Probably the best known and most widely used methodology is Six Sigma's DMAIC (Define, Measure, Analyze, Improve, and Control) which is widely used by Six Sigma practitioners today. The five steps of the DMAIC methodology are briefly described below:

Define During this first step in the DMAIC methodology, it is important to define specific goals in achieving outcomes that are consistent with both an organizations customer's demands and with its own business' strategy. In essence, you are laying down a road map for accomplishment.

Measure In order to determine whether or not defects have been reduced, base measurement is needed. In this step, accurate measurements must be made and relevant KPIs must be collected so that future comparisons can be measured to determine whether or not defects have been reduced.

Analyze Analysis determines the relationships and the factors of causality. When trying to understand how to fix a problem related to a business process, cause and effect is extremely necessary and must be considered.

Improve Improvement relies on upgrading or optimizing an organization's business processes, based on measurements and analysis that can ensure that defects are lowered and processes are streamlined.

Control This is the last step in the DMAIC methodology. Control ensures that any variances stand out and are corrected before they can influence a process negatively by causing defects. Controls can be in the form of pilot runs to determine if the processes are capable and then, once data are collected, a process can transition into standard production. However, continued measurement and analysis must ensue to keep processes on track and free of defects below the Six Sigma limit.

All steps rely on analysing each new business process as if it were unique. One begins by defining the scope of the process to be analysed, and then proceeds to decompose the process, identifying its major sub-processes, and then the sub-processes of those, identifying their major activities, and so on down to whatever level of granularity the designer chooses. Once the process is laid out in detail, the business analyst usually considers how to change it.

4.2.2 Supply Chain Operations Reference Methodology

A second-generation approach to business process redesign began to emerge a few years ago. This approach was proposed by the Supply Chain Council www.supply-chain.org who combined the expertise of supply-chain professionals across a broad cross-section of industries to develop best-in-class business practices and design a specific methodology tailored to the analysis of supply chain processes. Second generation software is usually tailored for specific industries or niche markets. The SCC named this second generation methodology the Supply Chain Operations Reference (SCOR) Framework [132]. SCOR is a business process methodology built by, and for, supply chain analysis and design. SCOR is a cross-industry, standardized, supply-chain reference model that enables companies to analyze and

improve their supply chain operations by helping them to communicate supply chain information across the enterprise and measure performance objectively. SCOR also assists enterprises with identifying supply chain performance gaps and improvement objectives and influences the development of future supply chain management software. SCOR provides standard definitions of measures and procedure for calculating the metrics. SCOR as a business process reference model contains [133]:

- Standard descriptions of management practices.
- A framework of relationships among the standard processes.
- Standard metrics to measure process performance.
- Management practices that produce best in class performance.
- Standard alignment to features and functionality.

The SCOR model depicts the supply-chain from a strategic perspective. It profiles the enterprise-wide business scope, it establishes the process boundaries, and it portrays the interrelationship of activities within the SCOR structure. This end-to-end business process model includes the primary activities by which business partners provide exceptional service to their customers, and it serves as a navigational tool and starting point to access all lower-level workflow models. The SCOR model consists of five basic processes: Plan, Source, Make, Deliver and Return [132]. In addition to these basic processes, there are three process types or categories: Enable, Planning and Execute. The SCOR modelling approach starts with the assumption that any supply chain process can be represented as a combination of the five basic processes. The Plan process balances demand and supply to best meet the sourcing, manufacturing and delivery requirements. The Source process procures goods and services to meet planned or actual demand. The Make process transforms product to a finished state to meet planned or actual demand. The Deliver process provides finished goods and services to meet planned or actual demand, typically including order management, transportation management and distribution management. The Return process is associated with returning or receiving any returned products.

At the core of the SCOR model comprises four levels of processes that guide supply chain members on the road to integrative process improvement [132]. These are shown in Figure 4.3. Level 1 describes supply chain processes at the most general level. It consists of the four key supply chain process types Plan, Source, Make, and Deliver, and assumes that all supply chains are composed out of these four basic processes. In other words, complex supply chains are made up of multiple combinations of these basic processes.

Level 2 defines 26 core supply chain process categories that were established by the SCC with which supply chain partners can jointly present their ideal or actual operational structure. Level 2 provides for variations in the Level 1 processes. These are not in fact sub-processes, but variations in the way the processes can be implemented. Each of the Level 1 processes currently has three variations. In analysing a process, an analyst first decides that there is a sourcing process (Level 1 process), and then decides which of three (Level 2) variations of sourcing process it is. For example, in the case of Level 1 Source process, the Level 2 variations are S1: Source Stocked Products, S2: Source Made-to-Order Products, or S3: Source Engineered-to-Order Product. Figure 4.3 shows all of the four basic SCOR Level 1 processes with current Level 2 variations inside their respective Level 1 process. Each Level 2 process is further defined by a set of sub-processes or activities that define the basic sequence of steps involved in implementing the process. In fact, in SCOR, the Level 3 processes are sub-processes of the Level 1 processes, and are the same, no matter the variation. Level 3 provides partners with information useful in planning and setting goals for supply chain improvement. Processes in the first three levels of the SCOR framework serve as the foundation for the development of Level 4 processes. Level 4 processes focus on implementation of supply chain process improvement efforts and are company specific practices designed to achieve and maintain competitive advantage in the industry. Level 4 processes are beyond the scope of the SCOR framework.

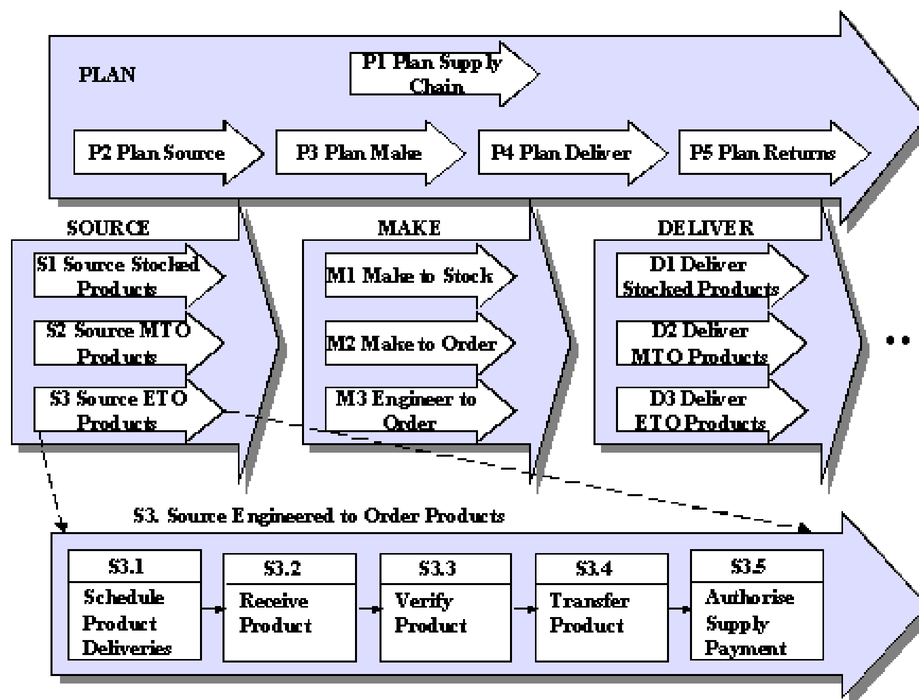

Figure 4.3: Levels of SCOR processes (source: [132])

The SCOR methodology is divided in six broad phases:

1. *Define the Supply Chain Process:* During the first phase existing processes are analyzed. This effort includes decisions about the number and scope of the supply chain processes to be examined.

2. *Determine the Performance of the Existing Supply Chain:* Once the existing supply chain process is scoped, it can use historical data to define how the existing supply chain is performing. In addition, the performance of a supply chain can be compared with benchmarks to determine how its processes measure up against similar processes in similar industries.

3. *Establish Supply Chain Strategy, Goals and Priorities:* Once the performance of an existing supply chain is determined, business analysts are in a position to consider if the supply chain strategy is reasonable, and how it might be improved.

4. *Redesign the Supply Chain as Needed:* SCOR provides tools for identifying problems and gaps and suggests the best practices used by enterprises within superior supply chains.

5. *Enable the Redesign* and *Implement:* Once the design is complete, the redesign must be implemented using software and human performance improvement techniques. Then the new supply chain must be implemented and data must be gathered to determine if the new targets are met.

The use of a framework-based business process methodology such as the SCOR model is only possible in cases where a high-level analysis of the processes to be analysed already exists, and where measures of process success have already been standardized.

4.2.3 Discussion

In general it is possible to divide business process methodologies, such as the ones described in the previous, into two broad categories:

1. bottom-up approaches: where analysts focus narrowly on redesigning and improving business processes, and
2. top-down approaches: where analysts focus more broadly on reorganizing an entire end-to-end process chain (network) and establishing a context for business process management.

Consider for example the difference between Six Sigma's DMAIC and the Supply Chain Council's SCOR methodology: DMAIC focuses on a single, narrowly defined process - usually a sub-process or sub-sub-process. The analyst measures the process and proceeds to focus on improving the quality of the output of the process and there is little focus on how the process fits within the larger context of an end-to-end process chain, or how the process is managed, or measured and monitored. On the other hand, the SCOR methodology begins by defining an enterprise's entire supply chain, comprising an end-to-end processes. Once the supply chain is defined, measures and benchmarks are applied to determine which specific business processes within the supply chain would yield the greatest performance improvement for the supply-chain, as a whole.

4.3 Summary

In the previous sections we presented in brief some of the major approaches in "classical" SE that have been applied with various degrees of success to SBA engineering. More specifically, we discussed component-based software engineering since approaches developed for components can be easily adapted to services. However, SOAs introduce some important issues that need to be considered: the ownership, physical location, description, discovery, and usage models of a service are drastically different than those of a component. Consequently, quality assurance was elaborated on as an important issue for both SBA and Software Engineering. Essentially, the purpose of implementing planned software processes is to ensure the quality of the final product by building in quality throughout the process - rather than discovering, either at testing phase or after its release, that there are problems with the product. The need for standardized processes in developing products has also been prevalent in business process methodologies. The challenges addressed and the solutions proposed in the business process domain have to be taken seriously into account for the service-based computing domain if the synergy between them is to be exploited. In addition, due to the wide adoption of SOAs in the enterprise domain, and the intrinsic relation between business processes and SBAs, SBA engineering should also consider the existing assets of each organization while providing solutions for leveraging its business processes. Finally, the service development has a set of implications for system development and maintenance processes, in particular with respect to the necessity of constant change driven by the shifting business needs they have to fullfil. There are a number of issues pertaining to the effect of the evolution and/or the maintainance of SBAs, sometimes with unforseen consequences, to their clients. Despite the existing efforts towards this direction there are still many questions and challenges to be met in that domain.

Chapter 5

Discussion on Knowledge Models

In the previous chapters we present two separate knowledge models, one for Software Engineering and one for Service Oriented Computing. The models presented are not complete in terms of depth and content, something that is very difficult to achieve within the scope of one deliverable, but they are nevertheless representative of the key research areas in both fields.

In summary, chapter 3 classified Service Oriented Computing methods, techniques and tools according to a proposed service lifecycle model (Figure 1.1). This model can be viewed in two stages: the development stage on the right hand side, and the adaptation stage on the left-hand side. We also discussed (in summary) the various SBA life-cycles. It has to be noted that each of these can cope with the full-blown development of a service, but due to the fact that the life-cycle of the service model clearly requires an adaptation phase we questioned whether the adaptation phase is identified and included in these models. In addition to that, we also need to ensure that the requirements, design, construction, deployment, provisioning, operation and management phases take the potential adaptation of services into account when they are being undertaken in the first place. Furthermore, there are governance, quality assurance, discovery and SLA negotiation issues that need to be considered, and these also need to be included in the life-cycle model.

Two major conclusions can be drawn after the presentation of the various life cycle methodologies:

1. Almost all methodologies have phases that correspond to right hand cycle of Figure 1.1. In that sense, our proposed life cycle is a fit model for representing the various stages of the service life cycle.

2. Most of the existing methodologies lack either partially or completely in providing for the left hand part of our life cycle, i.e., the adaptation phases of the SBA. This creates a number of opportunities in research towards that direction.

Chapter 4 on the other hand presented in brief some of the major approaches in classical software engineering that have been applied with various degrees of success to SBA engineering. We discussed some of the key contributions towards establishing the latter domain, building on the knowledge established by the former (domain). One of the most obvious candidates towards this direction is CBSE (component-based software engineering) due to the fact that services are the evolution of (software) components, and approaches developed in CBSE can be easily adapted to services. However, SOAs introduce some important issues that need to be considered: the ownership, physical location, description, discovery, and usage models of a service are drastically different than those of a component.

Quality assurance is an important issue for both SBA and Software Engineering. Essentially, the purpose of implementing planned software processes, following a standard like CMMI or ISO/IEC 15504, is to ensure the quality of the final product through building in quality throughout the process - rather than discovering either at testing phase or after its release that there are problems with the product. While we recognise that there are many valid reasons for not implementing the process models prescribed by

these standards, there are also efficiencies and increases in quality to be gained in doing so, and, in particular, there are markets who require planned processes to be in place. For example, the financial sector has commenced an initiative to implement Banking SPICE as they have to deal with regulations such as Sorbonnes-Oxley. In any case, SBAs as software artifacts can definitely benefit from lessons learned in software process quality approaches.

The need for standardized processes in developing products has also been prevalent in business process methodologies like the Rummler-Brache-PDL Methodology, DMAIC, SCOR, and vendor specific methodogies. An important difference though is that business process methodogies focus on the design or improvement of a business process, and on measuring and redefining processes, and not on the development of a software system. Nevertheless, the challenges addressed and the solutions proposed in the business process domain have to be taken seriously into account for the service-based computing domain if the synergy between them is to be exploited. For a further discussion on the beneficial effects of supporting Business Process Management with SOA solutions, and the usage of SBAs in implementing business processes, refer to PO-JRA-2.1.1.

In addition, due to the wide adoption of SOAs in the enterprise domain, and the intrinsic relation between business processes and SBAs, SBA engineering should also consider the existing assets of each organization while providing solutions for leveraging its business processes. One of the major assets that have to be taken into account is the legacy systems of the organization, that is the relatively old, mainframe-based systems that were optimized to accommodate the memory, disk, and other operational constraints of archaic software and hardware platforms. Devising a balanced strategy for handling legacy systems and (re-)aligning them with new process requirements is a very challenging issue. A number of strategies, methodologies and tools have been proposed by the industry, ranging from non-intrusive approaches such as screen scraping and legacy wrapping, to more invasive ones like grafting new designs into the outdated parts of the architecture of legacy systems. None of them though proved to be a silver bullet that could be applied in each situation, and that is a lesson also for SBA engineering: different approaches in developing and managing SBAs have to be examined, and the criteria based on which the decision to apply one or more of them have to be investigated. Finally, the service development has a set of implications for system development and maintenance processes, in particular with respect to the necessity of constant change driven by the shifting business needs they have to fullfil. There are a number of issues pertaining to the effect of the evolution and/or the maintainance of SBAs, sometimes with unforseen consequences, to their clients.

The classical SE methodologies such as the ones that we presented do not directly address three key elements of an SOA: services, service assemblies (composition), and components realizing services. These methodologies can only address part of the requirements of service-oriented computing applications. These practices fail when they attempt to develop service-oriented solutions while being applied independently of each other. Service-oriented design and development requires an inter-disciplinary approach fusing elements of techniques like object-oriented and component-based design with elements of business modeling. The challenge of SOA [6], and of SBA engineering by extension, is to elevate service enablement beyond just technology functions. The reality is that an SOA has limited value unless it encompasses disparate applications and platforms, and most importantly, it moves beyond technology and is orchestrated and controlled in the context of business processes. Developers need to be offered a variety of different services and functions that they can combine at will to create the right set of automated one-of-a-kind processes that can distinctly differentiate themselves from those of competitors. New processes and alliances need to be routinely mapped to services that can be used, modified, built or syndicated. In addition, there is also a clear need for SOA design methods that allow an organization to avoid the pitfalls of deploying an uncontrolled maze of services and provide a solid foundation for service enablement in an orderly fashion so that Web services can be efficiently used in SBAs.

The following deliverable in this work package will attempt to bridge this gap between the SE methodologies and the SBA engineering needs by focusing on integrating knowledge from both fields. Using this deliverable as a starting point, it will compare and coordinate the processes, methodologies

and techniques presented in isolation here towards identifying the principles and methodologies required for SBA engineering.

Bibliography

[1] L. Baresi, E. Di Nitto, and C. Ghezzi, “Toward open-world software: Issue and challenges,” *Computer*, vol. 39, no. 10, pp. 36–43, 2006.

[2] D. Parnas, “On the criteria to be used in decomposing systems into modules,” *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, December 1972.

[3] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, “A journey to highly dynamic, self-adaptive service-based applications,” *Automated Software Engineering*, vol. 15, no. 3-4, pp. 313–341, 2008.

[4] S-Cube Consortium, “State of the art report on software engineering design knowledge and survey of hci and contextual knowledge,” S-Cube project, Tech. Rep., 2008.

[5] “SeCSE Project,” http://www.secse-project.eu/.

[6] M. P. Papazoglou, *Web Services: Principles and Technology*. Addison-Wesley, 2007.

[7] M. Papazoglou, “The Challenges of Service Evolution,” in *CAiSE*, ser. Lecture Notes in Computer Science, vol. 5074. Springer, 2008, pp. 1–15.

[8] O. Zimmermann, P. Krogdahl, and C. Gee, “Elements of service-oriented analysis and design,” 2005, published: IBM developerWorks White Paper. [Online]. Available: http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/

[9] A. W. Brown, S. K. Johnston, G. Larsen, and J. Palistrant, “SOA Development Using the IBM Rational Software Development Platform: A Practical Guide,” 2005. [Online]. Available: ftp://ftp.software.ibm.com/software/rational/web/whitepapers/G507-0956-00.pdf

[10] J. Ganci, A. Acharya, J. Adams, P. D. de Eusebio, G. Rahi, D. Strachan, K. Utsumi, and N. Washio, *Patterns: SOA Foundation Service Creation Scenario*. IBM Redbooks, 2006.

[11] A. Arsanjani, “Service-oriented modeling and architecture,” IBM, November 2004. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/

[12] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, “SOMA: A method for developing service-oriented solutions,” *IBM Systems Journal*, vol. 47, no. 3, 2008. [Online]. Available: http://www.research.ibm.com/journal/sj/473/arsanjani.pdf

[13] A. Marconi, M. Pistore, P. Poccianti, and P. Traverso, “Automated Web Service Composition at Work: the Amazon/MPS Case Study,” in *ICWS, 2007 IEEE International Conference on Web Services*. IEEE, 2007, pp. 767–774.

[14] M. Trainotti, M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, F. Barbon, P. Bertoli, and P. Traverso, “Astro: Supporting composition and execution of web services,” in *ICSOC*, ser. Lecture Notes in Computer Science, vol. 3826. Springer, 2005, pp. 495–501.

[15] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-time monitoring of instances and classes of web service compositions," in *ICWS '06: Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 63–71.

[16] R. Kazhamiakin and M. Pistore, "Static verification of control and data in web service compositions," in *ICWS '06: Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 83–90.

[17] S. Durvasula and et al., "SOA Practitioners Guide," 2007, published: BEA Systems.

[18] I. Sommerville and G. Kotonya, *Requirements Engineering: Processes and Techniques*. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[19] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos, "Identifying and measuring quality in a software requirements specification," *Software Metrics Symposium, 1993. Proceedings., First International*, pp. 141–152, May 1993.

[20] B. Boehm and C. Abts, "Cots integration: Plug and pray?" *Computer*, vol. 32, no. 1, pp. 135–138, 1999. [Online]. Available: http://dx.doi.org/10.1109/2.738311

[21] W. T. Tsai, Z. Jin, P. Wang, and B. Wu, "Requirement engineering in service-oriented system engineering," in *ICEBE '07: Proceedings of the IEEE International Conference on e-Business Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 661–668.

[22] N. Maiden, "Servicing your requirements," *Software, IEEE*, vol. 23, no. 5, pp. 14–16, Sept.-Oct. 2006.

[23] T. Bohmann, M. Junginger, and H. Krcmar, "Modular service architectures: a concept and method for engineering it services," *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pp. 10 pp.–, Jan. 2003.

[24] L. A. Macaulay, *Requirements engineering*. London, UK: Springer-Verlag, 1996.

[25] S. Lichtenstein, L. Nguyen, and A. Hunter, "Issues in it service-oriented requirements engineering," *Australasian Journal of Information Systems*, vol. 13, no. 1, 2007. [Online]. Available: http://journals.sfu.ca/acs/index.php/ajis/article/view/70

[26] H. Goldsby, P. Sawyer, N. Bencomo, B. Cheng, and D. Hughes, "Goal-based modeling of dynamically adaptive system requirements," *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, pp. 36–45, 31 2008-April 4 2008.

[27] J. Hielscher, R. Kazhamiakin, A. Metzger, and M. Pistore, "A Framework for Proactive Self-Adaptation of Service-based Applications Based on Online Testing," in *ServiceWave 2008*. to be published, 10-13 December 2008.

[28] M. Pistore and P. Traverso, "Assumption-Based Composition and Monitoring of Web Services," in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds. Springer, 2007, pp. 307–335.

[29] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan, "Fault Tolerant Web Service Orchestration by Means of Diagnosis," in *Software Architecture, Third European Workshop, EWSA 2006*, 2006, pp. 2–16.

[30] J. Harney and P. Doshi, “Speeding up adaptation of web service compositions using expiration times,” in *WWW ’07: Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 1023–1032.

[31] L. Baresi and S. Guinea, “Towards Dynamic Monitoring of WS-BPEL Processes,” in *Service-Oriented Computing - ICSOC 2005, Third International Conference*, 2005, pp. 269–282.

[32] WS-Diamond, “Characterization of diagnosability and repairability for self-healing web services,” WS-DIAMOND Project IST-516933, Tech. Rep., 2007, deliverable D5.1.

[33] K. Mahbub and G. Spanoudakis, “Monitoring WS-Agreements: An Event Calculus-Based Approach,” in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds. Springer, 2007, pp. 265–306.

[34] S-Cube Consortium, “Taxonomy of adaptation principles and mechanisms,” S-Cube project, Tech. Rep., 2009.

[35] A. Lazovik, M. Aiello, and M. P. Papazoglou, “Associating assertions with business processes and monitoring their execution,” in *Service-Oriented Computing - ICSOC 2004, Second International Conference*, 2004, pp. 94–104.

[36] L. Baresi, S. Guinea, and L. Pasquale, “Self-healing BPEL processes with Dynamo and the JBoss rule engine,” in *ESSPE ’07: International workshop on Engineering of software services for pervasive environments*, 2007, pp. 11–20.

[37] M. Reichert and P. Dadam, “Adeptflex: Supporting dynamic changes of workflow without loosing control,” *Journal of Intelligent Information Systems*, vol. 10, pp. 93–129, 1998.

[38] S-Cube Consortium, “Overview of the state of the art in composition and coordination of services,” S-Cube project, Tech. Rep., 2008.

[39] T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana, “Business Process Execution Language for Web Services (version 1.1),” 2003.

[40] “JOpera Project,” http://www.iks.ethz.ch/jopera.

[41] “jBPM Process Definition Language (JPDL),” http://docs.jboss.org/jbpm.

[42] *Business Process Modeling Notation (BPMN) version 1.1.*, Object Management Group (OMG), January 2008. [Online]. Available: http://www.omg.org/spec/BPMN/1.1/

[43] W3C, *Web Services Choreography Description Language Version 1.0.*, 2005, [http://www.w3.org/TR/ws-cdl-10/].

[44] M. Pistore, P. Traverso, and P. Bertoli, “Automated Composition of Web Services by Planning in Asynchronous Domains,” in *Proc. ICAPS’05*, 2005.

[45] M. Pistore, A. Marconi, P. Traverso, and P. Bertoli, “Automated Composition of Web Services by Planning at the Knowledge Level,” in *Proc. IJCAI’05*, 2005.

[46] S-Cube Consortium, “Survey of quality related aspects relevant for sbas,” S-Cube project, Tech. Rep., 2008.

[47] ——, “Quality reference model for sba,” S-Cube project, Tech. Rep., 2009.

[48] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (wsdl) 1.1," http://www.w3.org/TR/wsdl, 2001.

[49] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. Mcllraith, S. Narayanan, M. Paolucci, B. Parsia, B. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "Owl-s: Semantic markup for web services," http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/, 2004.

[50] D. Fensel and C. Bussler, "The web service modeling framework wsmf," 2002. [Online]. Available: http://www1-c703.uibk.ac.at/users/c70385/wese/wsmf.paper.pdf

[51] "Business process execution language for web services," ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf,, 2003.

[52] "Business process modeling language," http://www.bpmi.org/, 2003.

[53] "Web service choreography interface (wsci) 1.0." www.w3.org/TR/wsci, 2003.

[54] "Uddi white papers," http://www.uddi.org/whitepapers.html.

[55] "UNSPSC," http://www.unspsc.org/.

[56] L. W., *New Directions in Software Engineering*. Leuven: Leuven University Press, 2001, ch. Web Service decription, advertising and discovery: WSDL and beyond, pp. 135–152.

[57] "ebXML," http://www.ebxml.org/.

[58] S. Dustdar and M. Treiber, "A view based analysis on web service registries," *Distributed and Parallel Databases*, vol. 18, no. 2, pp. 147–171, 2005.

[59] R. Lara, M. Corella, and P. Castells, "A flexible model for web service discovery," in *1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives*, Seoul, Korea, September 2006. [Online]. Available: http://sisinflab.poliba.it/smr06/

[60] K. Verna, K. Sivashanmugam, A. Shet, A. Patil, S. Oundhakar, and J. Miller, "Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services," *Information Technology and Management*, vol. Volume 6, pp. 17–39, 01 2005.

[61] T. Pilioura, G.-D. Kapos, and A. Tsalgatidou, "Seamless federation of heterogeneous service registries," in *EC-Web*, 2004, pp. 86–95.

[62] K. Sivashanmugam, K. Verma, and A. Sheth, "Discovery of web services in a federated registry environment," in *ICWS '04: Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2004, p. 270.

[63] V. Talwar, Q. Wu, C. Pu, W. Yan, G. Jung, and D. Milojicic, "Comparison of approaches to service deployment," *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pp. 543–552, June 2005.

[64] W. Arnold, T. Eilam, M. H. Kalantar, A. V. Konstantinou, and A. Totok, "Pattern based soa deployment," in *ICSOC*, 2007, pp. 1–12.

[65] "Web Services Distributed Management (WSDM)." [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm

[66] D. Oppenheimer and D. A. Patterson, "Studying and using failure data from large-scale internet services," in *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop*. New York, NY, USA: ACM, 2002, pp. 255–258.

[67] I. O. Electrical and E. E. (ieee), *IEEE 90: IEEE Standard Glossary of Software Engineering Terminology*. IEEE Computer Society, 1990.

[68] V. Dialani, S. Miles, L. Moreau, D. D. Roure, and M. Luck, "Transparent fault tolerance for web services based architectures," in *In Eighth International Europar Conference (EUROPAR02), Lecture Notes in Computer Science, Padeborn*. Springer-Verlag, 2002, pp. 889–898.

[69] G. T. Santos, L. C. Lung, and C. Montez, "Ftweb: A fault tolerant infrastructure for web services," in *EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 95–105.

[70] D. Ardagna, C. Cappiello, M. Fugini, E.Mussi, B.Pernici, and P.Plebani, "Faults and recovery actions for self-healing web services," 2006, www 2006, Edinburg, UK.

[71] A. Brogi and R. Popescu, "Automated Generation of BPEL Adapters," in *International Conference on Service Oriented Computing*, 2006.

[72] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani, "Developing Adapters for Web Services Integration," in *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*, 2005, pp. 415–429.

[73] L. Lambers, H. Ehrig, L. Mariani, and M. Pezze, "Iterative Model-Driven Development of Adaptable Service-Based Applications," in *ASE07*, 2007, pp. 453–456.

[74] N. Bieberstein and et al., *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap*. IBM Press, 2006.

[75] S. G. Kerrie Holley, Jim Palistrant, "Effective SOA Governance," 2006. [Online]. Available: http://www-306.ibm.com/software/solutions/soa/gov/lifecycle/

[76] T. Mitra, "A case for SOA governance," 2005. [Online]. Available: AcaseforSOAgovernance

[77] E. A. Marks and M. Bell, *Service Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology*. Wiley, 2006.

[78] L. Osterweil, "Strategic directions in software quality," *ACM Comput. Surv.*, vol. 28, no. 4, pp. 738–750, 1996.

[79] N. Delgado, A. Q. Gates, and S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *IEEE Trans. Software Eng.*, vol. 30, no. 12, pp. 859–872, 2004.

[80] J. L. Shoujian Yu and J. Le, "Intelligent Web Service Discovery in Large Distributed System," in *Intelligent Data Engineering and Automated Learning IDEAL 2004*, R. E. Zhen Rong Yang and H. Yin, Eds. Springer, 2004, pp. 166–172.

[81] J. Ma, J. Cao, and Y. Zhang, "A probabilistic semantic approach for discovering web services," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 1221–1222.

[82] U. Küster, B. König-Ries, M. Stern, and M. Klein, "Diane: an integrated approach to automated service discovery, matchmaking and composition," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 1033–1042.

[83] M. Stollberg, M. Hepp, and J. Hoffmann, "A caching mechanism for semantic web service discovery," in *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, ser.

LNCS, K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, G. Schreiber, and P. Cudr-Mauroux, Eds., vol. 4825. Berlin, Heidelberg: Springer Verlag, November 2007, pp. 477–490. [Online]. Available: http://iswc2007.semanticweb.org/papers/477.pdf

[84] M. Shakun, Ed., *Group Decision and Negotiation*. Springer Netherlands, 2002. [Online]. Available: http://www.citeulike.org/journal/springerlink-100270

[85] C. Briquet and P.-A. de Marneffe, "Grid resource negotiation: survey with a machine learning perspective," in *PDCN'06: Proceedings of the 24th IASTED international conference on Parallel and distributed computing and networks*. Anaheim, CA, USA: ACTA Press, 2006, pp. 17–22.

[86] M. Parkin, D. Kuo, and J. Brooke, "A framework & negotiation protocol for service contracts," in *SCC '06: Proceedings of the IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 253–256.

[87] W. S. Humphrey, *Managing the Software Process*. Reading, M.A., U.S.A.,: Addison-Wesley, 1989.

[88] M.Paulk, B. Curtis, M. Chrissis, and C. Weber, "The capability maturity model for software," SE Institute Carnegie Mellon, Tech. Rep., 1993.

[89] I. S. Organisation, "Information technology - software process assessment," International Standards Organisation," 2, 1998, parts 1-9.

[90] C. P. Team, "Capability maturity model™integration for development," S. E. Institute, Tech. Rep., 2006.

[91] Y. L. Norman Fenton, Robin Whitty, *Software Quality Assurance Measurement Perspective*. UK: International Thomson Computer Press, 1995.

[92] C. Jones, *Patterns of Software Systems Failure and Success*. International Thompson Computer Press, 1996.

[93] J. Kolind and D. G. Wastell, "The sei's capability maturity model: a critical survey of adoption experiences in a cross-section of typical uk companies," in *IFIP TC8 WG8.6 International Working Conference on Diffusion, Adoption and Implementation of Information Technology*, T. McMaster, E. Mumford, E. B. Swanson, B. Warboys, and D. Wastell, Eds., Ambleside, Cumbria, U.K, 1997, pp. 305–319.

[94] B. Bergman and B. Klefsjo, *Quality From Customer Needs to Customer Satisfaction*, Lund, Ed. Studentlitteratur, 1994.

[95] L. Strader, M. Beim, and J. Rodgers, "The motivation and development of the space shuttle onboard software (obs) project," *Software Process Improvement and Practice*, vol. 1, pp. 107–113, 1995.

[96] J. G. Brodman and D. L. Johnson, "A software process improvement approach tailored for small organisations and small projects," in *19th International Conference on Software Engineering*, Boston, Massachusetts, USA, 1997.

[97] W. Humphrey, "Three dimensions of process improvement, part i: Process maturity," *CROSSTALK The Journal of Defense Software Engineering*, 1998.

[98] D. Galin and M. Avrahami, "Are cmm program investments beneficial? analyzing past studies," *IEEE Software*, pp. 81–87, 2006.

[99] B. Schatz and I. Abdelshafi, "Primavera gets agile: a successful transition to agile development," *Software, IEEE*, vol. 22, no. 3, pp. 36–42, 2005.

[100] M. Leszak, D. Perry, and D. Stoll, "A Case Study in Root Cause Defect Analysis," in *International Conference on Software Engineering: Proceedings of the 22 nd international conference on Software engineering*, vol. 4, no. 11, 2000, pp. 428–437.

[101] A. Law and S. Learn, "Waltzing with Changes," in *Proceedings of the Agile Development Conference*. IEEE Computer Society Washington, DC, USA, 2005, pp. 279–288.

[102] B. Kent, "Extreme Programming Explained: Embrace Change," *Reading, Mass.: Adison-Wesley*, 2000.

[103] B. Gallagher and L. Brownsword, "The rational unified process and the capability maturity model – integrated systems/software engineering," in *RUP/CMMI Tutorial – ESEPG*, 2001.

[104] W. Melo, "Rup for cmmi compliance: A methodological approach," July 2008. [Online]. Available: http://www.ibm.com/developerworks/rational/library/5318.html

[105] J. Lee, J. Kim, and G.-S. Shin, "Facilitating reuse of software components using repository technology," in *APSEC '03: Proceedings of the Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2003, p. 136.

[106] S. Mahmood, R. Lai, Y.-S. Kim, J. H. Kim, S. C. Park, and H. S. Oh, "A survey of component based system quality assurance and assessment," *Information & Software Technology*, vol. 47, no. 10, pp. 693–707, 2005.

[107] X. Cai, M. R. Lyu, K. fai Wong, and R. Ko, "Component-based software engineering: Technologies, development frameworks, and quality assurance schemes," in *Lecture Notes*. IEEE Computer Society, 2000, pp. 372–379.

[108] K.-K. Lau and Z. Wang, "A taxonomy of software component models," in *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 88–95.

[109] H. Pei-Breivold and M. Larsson, "Component-based and service-oriented software engineering: Key concepts and principles," in *33rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Component Based Software Engineering (CBSE) Track, IEEE*, August 2007. [Online]. Available: http://www.mrtc.mdh.se/index.php?choice=publications&id=1291

[110] M. L. Brodie and M. Stonebraker, *Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach*. Morgan Kaufman Publishing Company, 1995.

[111] N. Weiderman, L. Northrop, D. Smith, S. Tilley, and K. Wallnau, "Implications of distributed object technology for reengineering," Technical Report CMU/SEI-97-TR-005 / ESC-TR-97-005, 1997.

[112] I. Warren, *The Renaissance of Legacy Systems: Method Support for Software-System Evolution*. Springer, Practitioner Series, London, 1999.

[113] W. Ulrich, *Legacy Systems Transformation Strategies*. Upper Saddle River, NJ.: Prentice Hall PTR, 2002.

[114] A. Umar, *Application (Re) Engineering: Building Web-Based Applications and Dealing with Legacies*. Prentice Hall, 1997.

[115] R. Levey, *Reengineering COBOL with Objects: Step by Step to Sustainable Legacy Systems*. McGraw-Hill, New York, 1996.

[116] R. Seacord, D. Plakosh, and G. Lewis, *Modernizing Legacy Systems*. Carnegie Mellon, SEI, Addison-Wesley, 2003.

[117] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures," *J. Syst. Softw.*, vol. 81, no. 4, pp. 463–480, 2008.

[118] B. Graaf, S. Weber, and A. van Deursen, "Model-driven migration of supervisory machine control architectures," *J. Syst. Softw.*, vol. 81, no. 4, pp. 517–535, 2008.

[119] U. Zdun, "Reengineering to the web: A reference architecture," in *CSMR '02: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2002, p. 164.

[120] A. Erradi, S. Anand, and N. Kulkarni, "Evaluation of strategies for integrating legacy applications as services in a service oriented architecture," in *SCC '06: Proceedings of the IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 257–260.

[121] J. Hutchinson, G. Kotonya, J. Walkerdine, P. Sawyer, G. Dobson, and V. Onditi, "Evolving existing systems to service-oriented architectures: Perspective and challenges," in *ICWS*. IEEE Computer Society, 2007, pp. 896–903.

[122] I. Rus and M. Lindvall, "Knowledge management in software engineering," *IEEE Software*, vol. 19, no. 3, May 2002.

[123] G. Lewis, E. Morris, D. Smith, and L. O'Brien, "Service-oriented migration and reuse technique (smart)," in *STEP '05: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 222–229.

[124] W.-J. van den Heuvel, J. van Hillegersberg, and M. P. Papazoglou, "A methodology to support web-services development using legacy systems," in *Proceedings of the IFIP TC8 / WG8.1 Working Conference on Engineering Information Systems in the Internet Context*. Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 2002, pp. 81–103.

[125] W.-J. van den Heuvel, *Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems)*. The MIT Press, 2006.

[126] B. P. Lientz and E. B. Swanson, *Software Maintenance Management*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1980.

[127] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Conference on The Future of Software Engineering*. New York, NY, USA: ACM Press, 2000, pp. 73–87. [Online]. Available: http://dx.doi.org/10.1145/336512.336534

[128] Y. Wu, D. Pan, and M.-H. Chen, "Techniques of maintaining evolving component-based software," *Software Maintenance, IEEE International Conference on*, vol. 0, p. 236, 2000.

[129] M. Casanova, R. Van Der Straeten, and V. Jonckers, "Supporting evolution in component-based development using component libraries," in *CSMR '03: Proceedings of the Seventh European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2003, p. 123.

[130] M. Kajko-Mattsson and M. Tepczynski, "A framework for the evolution and maintenance of web services," in *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 665–668.

[131] G. A. Rummler and A. P. Brache, *Improving Performance: How to Manage the White Space on the Organization Chart*, 2nd ed. San Francisco: Jossey-Bass, 1995.

[132] P. Harmon, "Second generation business process methodologies," *Business Process Trends: Newsletter*, vol. 1, no. 5, 2003.

[133] V. Kasi, "Systemic assessment of scor for modeling supply chains," in *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 3*. Washington, DC, USA: IEEE Computer Society, 2005, p. 87.2.