LNCS 6500

State-of-the-Art Survey

Mike Papazoglou
Klaus Pohl
Michael Parkin
Andreas Metzger (Eds.)

Service Research Challenges and Solutions for the Future Internet

S-Cube – Towards Engineering, Managing and Adapting Service-Based Systems

Springer

S-CUBE

Lecture Notes in Computer Science 6500

Services Science

Subline of Lectures Notes in Computer Science

Mike Papazoglou Klaus Pohl
Michael Parkin Andreas Metzger (Eds.)

Service Research Challenges and Solutions for the Future Internet

S-Cube – Towards Engineering, Managing and Adapting Service-Based Systems

Springer

Volume Editors

Mike Papazoglou
Michael Parkin
Tilburg University
European Research Institute in Service Science (ERISS)
5000 LE Tilburg, The Netherlands
E-mail: {m.p.papazoglou, m.s.parkin}@uvt.nl

Klaus Pohl
Andreas Metzger
University Duisburg-Essen
Paluno (The Ruhr Institute for Software Technology)
45127 Essen, Germany
E-mail: {klaus.pohl, andreas.metzger}@sse.uni-due.de

Library of Congress Control Number: 2010940045

CR Subject Classification (1998): C.2, F.3, D.2, I.2, H.4, H.3

LNCS Sublibrary: SL 5 – Computer Communication Networks
and Telecommunications

ISSN 0302-9743
ISBN-10 3-642-17598-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-17598-5 Springer Berlin Heidelberg New York

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180 5 4 3 2 1 0

Foreword

S-Cube's Foundations for the Internet of Services

Today's Internet is standing at a crossroads. The Internet has evolved from a source of information to a critical infrastructure which underpins our lives and economies. The demand for more multimedia content, more interconnected devices, more users, a richer user experience, services available any time and anywhere increases the pressure on existing networks and service platforms. The Internet needs a fundamental rearrangement to be ready to meet future needs.

One of the areas of research for the Future Internet is the *Internet of Services*, a vision of the Internet where everything (e.g., information, software, platforms and infrastructures) is available as a service. Services available on the Internet of Services can be used by anyone (if they are used according to the policies defined by the provider) and they can be extended with new services by anyone. Advantages of the Internet of Services include the possibility to build upon other people's efforts and the little investment needed upfront to develop an application. The risk involved in pursuing new business ideas is diminished, and might lead to more innovative ideas being tried out in practice. It will lead to the appearance of new companies that are able to operate in niche areas, providing services to other companies that will be able to focus on their core business.

The Internet of Services vision is very attractive, but still needs a lot of fundamental and applied research to become a reality. Some of the questions that need to be answered are:

- How can we deal with the increasing complexity and dynamicity of tomorrow's software and services?
- How can we engineer services that can be reused in situations not foreseen during design time but that will still behave in a reliable and dependable manner?
- How can we deal with the heterogeneity in devices, networks, computers, and service platforms, and ease their management?

As part of the European Union's FP7 – ICT Programme for Research and Development, two calls for projects have been made in the field of the Internet of Services. As a result of these calls, a group of more than 50 projects with a total budget of more than 340 million euro is researching various aspects of the Internet of Services, guided by the questions above. Within this group the project S-Cube is researching fundamental principles, techniques and methods for the service-based systems of the future, combining knowledge from top-level experts in the areas of grid, business process management, software engineering and human–computer interaction. Some of the results of S-Cube are reflected in this book.

Of all the interesting results, I would like to mention the service life cycle research. S-Cube has developed a reference service life cycle for adaptable service-based applications. For each phase of the life cycle an analysis is made of the main concepts, issues, and challenges. In addition, various software engineering and business process management methodologies that can be useful in a certain phase of the life cycle are identified.

This service life cycle research is just one example of how S-Cube is combining fundamental practices in various disciplines to provide a basis for the service-based systems of the future. The research described in this book is important for any further research related to the Internet of Services, and also serves very well as one of the first books of the Springer Service Science subseries.

October 2010

Anne-Marie Sassen
Project Officer
European Commission, DG Information Society and Media

Preface

The field of software services has become increasingly significant as the ubiquitous Internet has provided global connectivity to networked software (i.e., services) and allowed the flexible composition of those services into innovative service-based applications (SBAs). The number and scope of services available globally is predicted to grow exponentially into the "Internet of Services". Therefore, in order for these services to be used in and between SBAs, significant research effort is required into the broader aspects of services and especially the multidisciplinary problems that cut across a number of scientific fields.

Although many research organizations have developed research agendas to study services and SBAs, the multidisciplinary issues and challenges the Internet of Services brings have so far not been investigated in an organized, systematic and coherent manner. S-Cube, the European Software Services and Systems Network, was designed to meet this urgent need in the field of European software services research.

S-Cube uses the combined efforts and the diverse, multidisciplinary knowledge of the network's members to bridge the gap in current research and to define a broader research vision that will help shape the Internet of Services. S-Cube is a four-year project funded by the European Commission under Framework Programme 7. This book presents the foundations, vision, first results and future work of the S-Cube project.[1]

Chapter 1 opens the book with an introduction to the vision of S-Cube and how the anticipated growth in services and service-based systems will have a profound effect on business and society. This vision chapter presents the S-Cube research framework, developed to assist in unifying research com-

[1] The research leading to the results presented in this book has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). The book reflects only the authors' view. The EC is not liable for any use that may be made of the information contained therein.

munities and agendas across Europe. The chapter describes selected, fundamental cross-cutting research challenges and how the cooperation of different research disciplines plays a crucial role in meeting these challenges.

The remainder of the book follows the building blocks of the S-Cube research framework, which are separated into two areas of concern:

- Technologies required for realizing future service-based systems
- Engineering principles, techniques and methods that use those technologies to ensure robust, manageable and adaptable systems

Thus, Chaps. 2–4 focus on the three service technology layers of the framework (business process management, service composition and coordination, and service infrastructure), while Chaps. 5–9 focus on service engineering principles, techniques and methods including service adaptation and service quality assurance.

Chapter 2 surveys recent advances in business process management (BPM). It describes how business transactions between services can be enabled through a novel business transaction model developed by S-Cube researchers. This model and an associated business transaction language allow service networks to behave according to agreed-upon transaction criteria. The objective of this work is to provide the environment to build robust and successful mission-critical SBAs, using a fusion of concepts from application integration, transaction-based and business process management technologies.

Chapter 3 provides a survey of the state of the art in modeling service composition and coordination (SCC). This chapter covers two areas: service composition models and approaches for synthesizing service compositions, including model-driven, automated and QoS-aware strategies. The survey presented in this chapter clearly indicates the need to align and improve service composition approaches, one of S-Cube's research aims. The conclusions of this chapter can be used as a basis for aligning and improving existing approaches and solutions for service composition and to provide future research directions in this area.

Chapter 4 discusses advances in service infrastructure technologies. Those technologies provide the capabilities for defining basic communication patterns and interactions amongst services that can provide, for example, contextual and qualitative information about a service's environment and performance. Providing these capabilities to the other technology layers allows service developers to use contextual information when building service-based applications and facilitates cross-layer and proactive monitoring and adaptation of services. This chapter gives an overview of service infrastructures to provide the adaptation, monitoring and management of services and concludes with a discussion of more detailed research challenges in the context of service infrastructure management.

Chapter 5 addresses the issue of how to handle unexpected events and modified conditions, such as changes in the context in which an application is used or changes in the user's requirements. To this end the chapter presents a survey and review of relevant techniques and methods for monitoring services and SBAs to detect those changes, followed by an analysis of existing techniques for adapting SBAs in response to such changes.

Chapter 6 presents the principles, techniques and methods required to specify and negotiate the nonfunctional properties of a service, or the quality of service (QoS). Establishing and agreeing on QoS properties is essential for building reliable and predictable services and SBAs. This chapter identifies problems inherent in this task and reviews the related work that attempts to solve these problems. The chapter ends with the identification of current research gaps and potential research challenges in modeling and negotiating this aspect of a service's or SBA's behavior.

Chapter 7 continues the theme of service quality and presents a review of techniques and methods to assure the quality of services and service-based applications. This chapter focusses on analytical quality assurance techniques and reviews critically the state of the art in testing, static analysis and formal verification. From the point of view of quality assurance, the chapter also reviews monitoring techniques (introduced in Chap. 5) in order to understand the differences and potential synergies with testing and static analysis. The chapter concludes with a discussion of highly relevant research issues in service quality.

Chapter 8 presents an analysis of existing service engineering and design approaches for building and operating adaptable and evolvable service-based applications. It identifies the main activities and phases in the service life cycle and highlights issues, and challenges that need to be addressed to realize the life cycle. The emphasis in this chapter is on how the current lack of a standard service life cycle incorporating the evolution, adaptation and quality assurance of service-based applications led to the definition of the S-Cube reference life cycle introduced in Chap. 1.

Chapter 9 shows how service engineering and design principles can be used to develop a service by employing two industrial case studies. The work emphasizes the identification of three service engineering views: the automation decision view, degree of service automation view and service automation related data view. These views structure and support the elicitation and documentation of stakeholders' concerns and assists them in their work.

We would like to take this opportunity to thank the people who contributed to the successful publication of this book. We acknowledge the support of Ursula Barth from Springer, who made this book possible. Our particular thanks go the European Commission, in particular to Jorge Gasos for his continuous and valuable support as the project officer of S-Cube and to Anne-Marie Sassen for contributing the foreword to this book. On behalf of the S-Cube consortium, we extend our gratitude to S-Cube's reviewers. Their constructive recommendations have been extremely helpful in improving the project. Last but not least, we would like to thank the many S-Cube researchers that made this book possible through their excellent and comprehensive contributions to the various chapters.

We hope this book will be a helpful reference and inspiring source for your own research and development activities in services and service-based systems.

October 2010

Mike Papazoglou
European Research Institute in Services Science (ERISS)
Tilburg University

Klaus Pohl
Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen

Michael Parkin
European Research Institute in Services Science (ERISS)
Tilburg University

Andreas Metzger
Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen

List of Contributors

Françoise André
INRIA
Campus Universitaire de Beaulieu
35042 Rennes Cedex
France
francoise.andre@irisa.fr

Vasilios Andrikopoulos
European Research Institute in Service
Science (ERISS), Tilburg University
PO Box 90153, 5000 LE Tilburg
The Netherlands
v.andrikopoulos@uvt.nl

Olivier Barais
INRIA
Campus Universitaire de Beaulieu
35042 Rennes Cedex
France
barais@irisa.fr

Luciano Baresi
Politecnico de Milano
Dipartimento di Elettronica
e Informazione, via Golgi, 42
20133 Milano
Italy
baresi@elet.polimi.it

George Baryannis
Department of Computer Science
University of Crete
GR 71409 Heraklion
Greece
gmparg@csd.uoc.gr

Salima Benbernou
Université Paris Descartes
45 rue des Saints Pères
75270 Paris Cedex 06
France
sbenbern@liris.univ-lyon1.fr

Ivona Brandic
Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1
1040 Vienna
Austria
ivona@infosys.tuwien.ac.at

Antonio Bucchiarone
SOA Research Unit, FBK-Irst
via Sommarive 18
38100 Povo (TN)
Italy
bucchiarone@fbk.eu

Cinzia Cappiello
Politecnico di Milano
Dipartimento di Elettronica
e Informazione
Piazza Leonardo da Vinci 32
20133 Milano
Italy
cappiell@elet.polimi.it

Manuel Carro
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo
E-28660 Boadilla del Monte
Spain
mcarro@fi.upm.es

Noel Carroll
Lero - the Irish Software Engineering Research Centre
Department of Computer Science & Information Systems
University of Limerick
Ireland
noel.carroll@lero.ie

Marco Comuzzi
School of Industrial Engineering
Eindhoven University of Technology
Den Dolech 2
5612AZ, Eindhoven
The Netherlands
m.comuzzi@tue.nl

Félix Cuadrado
ETSI Telecomunicación
Universidad Poliécnica de Madrid
Avda Complutense 30
28040 Madrid
Spain
fcuadrado@dit.upm.es

Olha Danylevych
Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany
olha.danylevych@iaas.uni-stuttgart.de

Erwan Daubert
INRIA
Campus Universitaire de Beaulieu
35042 Rennes Cedex
France
erwan.daubert@irisa.fr

Claudia Di Napoli
Istituto di Cibernetica "E. Caianiello" - C.N.R.
Via Campi Flegrei 34
80078 Pozzuoli, Naples
Italy
c.dinapoli@cib.na.cnr.it

Elisabetta Di Nitto
Politecnico di Milano
Dipartimento di Elettronica e Informazione
via Golgi, 42
20133 Milano
Italy
dinitto@elet.polimi.it

Maha Driss
INRIA, IRISA, Equipe TRISKELL, F233
Université de Rennes 1, Campus de Beaulieu
35042 Rennes
France
maha.driss@inria.fr

Juan Carlos Dueñas
ETSI Telecomunicación
Universidad Poliécnica de Madrid
Avda Complutense 30
28040 Madrid
Spain
jcduenas@dit.upm.es

Schahram Dustdar
Technische Universität Wien
Institut für Informationssysteme 184/1
Arbeitsbereich für Verteilte Systeme
Argentinierstrasse 8
A-1040 Wien
Austria
dustdar@infosys.tuwien.ac.at

Guillaume Gauvrit
INRIA
Campus Universitaire de Beaulieu
35042 Rennes Cedex
France
guillaume.gauvrit@irisa.fr

Maurizio Giordano
Istituto di Cibernetica "E. Caianiello" - C.N.R.
Via Campi Flegrei 34
80078 Pozzuoli, Naples
Italy
`m.giordano@cib.na.cnr.it`

Qing Gu
Department of Computer Science
VU University Amsterdam
De Boelelaan 1081a
1081HV Amsterdam
The Netherlands
`qinggu@cs.vu.nl`

Mohand-Said Hacid
LIRIS-UFR Informatique
Université Claude Bernard Lyon 1
43, boulevard du 11 novembre 1918
69622 Villeurbanne Cedex
`mohand-said.hacid@univ-lyon1.fr`

François Hantry
LIRIS UMR 5205
Université Claude Bernard Bâtiment Nautibus (710)
43, Boulevard du 11 Novembre 1918
69622 Villeurbanne Cedex
France
`francois.hantry@liris.cnrs.fr`

Rafiq Haque
European Research Institute in Service Science (ERISS)
Tilburg University
PO Box 90153
5000 LE Tilburg
The Netherlands
`r.haque@uvt.nl`

Willem-Jan van den Heuvel
European Research Institute in Service Science (ERISS)
Tilburg University
PO Box 90153
5000 LE Tilburg
The Netherlands
`w.j.a.m.vdnheuvel@uvt.nl`

Dimka Karastoyanova
Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany
`karastoyanova@iaas.uni-stuttgart.de`

Raman Kazhamiakin
Raman Kazhamiakin
SOA Research Unit, FBK-Irst
via Sommarive 18
38100, Povo (TN)
Italy
`raman@fbk.eu`

Gábor Kecskeméti
MTA Computer & Automation Research Institute
P.O. Box 63
H-1518 Budapest
Hungary
`gabor.kecskemeti@sztaki.hu`

Attila Kertész
MTA Computer & Automation Research Institute
P.O. Box 63
H-1518 Budapest
Hungary
`attila.kertesz@sztaki.hu`

Kyriakos Kritikos
ICS-FORTH
N. Plastira 100, Vassilika Vouton
GR-700 13 Heraklion, Crete
Greece
`kritikos@ics.forth.gr`

Patricia Lago
Department of Computer Science
VU University Amsterdam
De Boelelaan 1081a
1081HV Amsterdam
The Netherlands
`patricia@cs.vu.nl`

Stephen Lane
Lero - the Irish Software Engineering Research Centre
Department of Computer Science & Information Systems
University of Limerick
Ireland
stephen.lane@lero.ie

Winfried Lamersdorf
Arbeitsbereich Verteilte Systeme und Informationssysteme (VSIS)
FB Informatik der Universität Hamburg
Vogt-Kölln-Straße 30
D-22527 Hamburg
Germany
lamersd@informatik. uni-hamburg.de

Frank Leymann
Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany
leymann@iaas.uni-stuttgart.de

Philipp Leitner
Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1
1040 Vienna
Austria
leitner@infosys.tuwien.ac.at

Valentina Mazza
Politecnico di Milano
Dipartimento di Elettronica
e Informazione
via Golgi, 42
20133 Milano
Italy
vmazza@elet.polimi.it

Andreas Metzger
Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen
45127 Essen
Germany
andreas.metzger@sse.uni-due.de

Andrea Mocci
Politecnico di Milano
Dipartimento di Elettronica
e Informazione
Via Ponzio 34/5
20133 Milano
Italy
mocci@elet.polimi.it

Zsolt Németh
MTA Computer & Automation Research Institute
P.O. Box 63
H-1518 Budapest
Hungary
zsolt.nemeth@sztaki.hu

Christos Nikolau
Transformation Services Lab – CSD
University of Crete
Leoforos Knossou, 71409
Greece
nikolau@tsl.gr

Mike Papazoglou
European Research Institute in Service Science (ERISS)
Tilburg University
PO Box 90153
5000 LE Tilburg
The Netherlands
m.p.papazoglou@uvt.nl

Michael Parkin
European Research Institute in Service Science (ERISS)
Tilburg University
PO Box 90153
5000 LE Tilburg
The Netherlands
m.s.parkin@uvt.nl

Jean-Louis Pazat
INRIA
Campus universitaire de Beaulieu
35042 Rennes Cedex
France
jean-louis.pazat@irisa.fr

Barbara Pernici
Politecnico di Milano
Dipartimento di Elettronica
e Informazione
Piazza Leonardo da Vinci 32
20133 Milano
Italy
barbara.pernici@polimi.it

Pierluigi Plebani
Politecnico di Milano
Dipartimento di Elettronica
e Informazione
Piazza Leonardo da Vinci 32
20133 Milano
Italy
plebani@elet.polimi.it

Klaus Pohl
Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen
45127 Essen
Germany
klaus.pohl@sse.uni-due.de

Harald Psaier
Technische Universität Wien
Institut für Informationssysteme 184/1
Arbeitsbereich für Verteilte Systeme
Argentinierstrasse 8
A-1040 Wien
Austria
h.psaier@infosys.tuwien.ac.at

Wolfgang Renz
Multimedia Systems Laboratory
Faculty of Engineering & Computer Science
Hamburg University of Applied Sciences
Berliner Tor 7
20099 Hamburg
Germany
wolfgang.renz@haw-hamburg.de

Ita Richardson
Lero - the Irish Software Engineering Research Centre
Department of Computer Science & Information Systems
University of Limerick
Ireland
ita.richardson@ul.ie

Florian Rosenberg
CSIRO ICT Centre
GPO Box 664
Canberra ACT 2601
Australia
florian.rosenberg@csiro.au

Fabrizio Silvestri
Istituto di Scienze e Tecnologie
dell'Informazione (ISTI) – A. Faedo
Area della Ricerca di Pisa
Via G. Moruzzi, 1
56124 Pisa
Italy
fabrizio.silvestri@isti.cnr.it

Jan Sudeikat
Multimedia Systems Laboratory
Faculty of Engineering & Computer Science
Hamburg University of Applied Sciences
Berliner Tor 7
20099 Hamburg
Germany
jan.sudeikat@haw-hamburg.de

Maike Uhlig
Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen
45127 Essen
Germany
maike.uhlig@sse.uni-due.de

Branimir Wetzstein
Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany
branimir.wetzstein@iaas.uni-stuttgart.de

Eoin Whelan
Lero - the Irish Software Engineering Research Centre
Department of Computer Science & Information Systems
University of Limerick
Ireland
eoin.whelan@ul.ie

Contents

1

The S-Cube Research Vision

Mike Papazoglou[1], Klaus Pohl[2], Andreas Metzger[2], and
Willem-Jan van den Heuvel[1] together with the S-Cube team

[1] European Research Institute in Service Science (ERISS), Tilburg University, PO Box 90153, 5000 LE Tilburg, The Netherlands m.p.papazoglou@uvt.nl w.j.a.m.vdnheuvel@uvt.nl

[2] Paluno (The Ruhr Institute for Software Technology), University of Duisburg-Essen, 45127 Essen, Germany klaus.pohl@sse.uni-due.de andreas.metzger@sse.uni-due.de

Chapter Overview. This chapter sets the scene and gives the background for S-Cube's research vision and activities described in the remainder of the book. It does this by describing, in Section 1.1, how the anticipated growth in services and service-based systems that together form the Internet of Services will have a profound effect on business and society. Section 1.2 discusses in more detail some selected, fundamental cross-cutting research challenges and how the cooperation of different research disciplines plays an important role. In Section 1.3 we describe the research framework S-Cube has adopted to assist in unifying research communities and agendas across Europe to meet the challenges faced in realizing the Future Internet.

1.1 The Internet of Services

The next decade holds the prospect of remarkable progress in a wide range of pervasive technologies culminating in the introduction of the Future Internet — a global, open platform with emphasis on mobility, massive scale of connected devices, increased bandwidth and digital media. The goal is the development of a converged information, communication and service infrastructure that gradually will replace the current Internet, mobile, fixed, satellite and audiovisual networks. This infrastructure will not only be pervasive, ubiquitous, and highly dynamic, but will also offer almost unlimited capacities to users, by supporting a wide variety of nomadic and mobile interoperable devices and services, a variety of content formats and a multiplicity of delivery modes. Beyond technological aspects, the Future Internet is likely to have a profound effect on our society, from a societal, organizational or business perspective. Future Internet-based systems are set to revolutionize the worlds of healthcare, agriculture, the environment, transport, telecommunications, manufacturing, distribution, recycling, and retailing, to name just a few ap-

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 1–26, 2010.

plication areas, so much that the Future Internet could be of huge benefit to mankind.

Central to this vision is the availability of rich and flexible service capabilities, where the Internet world comprises of cooperating services with application components than can be combined dynamically with little effort into globally value-added services that can be traded outside traditional ownership and provisioning boundaries to yield higher levels of productivity. With these technological changes on the horizon, there is enormous potential for globally-available software services that gravitate towards new kinds of high-speed networks that employ a multiplicity of wired and wireless devices, sensors (e.g., RFID) and other service delivery artifacts. With global availability service-related functions are independent of the underlying devices, platform characteristics, connectivity protocols and transport technologies. This not only widens considerably the scope of systems but also provides the possibility of developing a new range of innovative systems, which can be provisioned by widely distributed network infrastructures.

In this new environment, *software services* (or simply *services*) constitute self-contained computational elements that support rapid and flexible composition of loosely coupled distributed software systems. The functionality provided by a service can range from answering simple requests to executing sophisticated processes requiring peer-to-peer relationships between a multitude of service consumers and providers. Services are described, published, discovered, and can be assembled to create complex service-based systems and service-based systems, which are inherently distributed. For the service consumer, a service represents functionality that can be invoked through the service interface. The actual software or application logic that implements the service composition is owned by the service provider. However, the composed service itself as well as the services which are aggregated by the composed service are often owned, executed and maintained by third parties. Thus, services take the concept of ownership to the extreme: not only is the development, quality assurance, and maintenance of the software under the control of third parties, but the software can also be executed and managed by third parties.

Software services and service-based systems imply fundamental changes to how software is developed, deployed, and maintained. More specifically, three broad classes of challenges need to be addressed:

- Evolution and Adaptation: Service-based systems run in dynamic business environments and address constantly evolving requirements. These systems hence have to be able to adequately identify and react to various changes in the business requirements and application context. Besides run-time mechanisms and strategies to support system adaptation, this requires novel engineering and design approaches that consider these mechanisms and strategies during the construction service-based systems.

- Dynamic Quality Assurance: To provide the desired end-to-end quality of globally distributed service-based systems, the dynamic agreement and assurance of quality becomes a key issue. This requires that not only quality aspects are negotiated and agreed, but also that those are checked during run-time. Further, to address dynamic adaptations of service-based systems, a growing need for automating the negotiation of quality attributes (e.g., stipulated by SLAs) can be observed. Finally, validation and verification techniques that can be applied at run-time become essential.
- Interplay of Technology Layers: Currently, the common practice for developing service-based applications (SBAs) following the Service-Oriented Architecture (SOA) paradigm considers three technology layers: service infrastructure, service composition and coordination, and business process management (BPM). When setting out to build innovative software services and service-based systems of the future, relying on the current layers of the SOA will not suffice. For example, the interplay of the layers must be understood to avoid conflicting adaptations in different layers. Also, each layer impacts on the end-to-end quality of a service-based system and thus needs to be taken into account.

Having set the scene for the Internet of Services, the following section describes in detail selected research challenges arising from it. As we will show, these challenges cut-across many scientific disciplines, techniques and approaches, demand a coordinated interplay between the building blocks of the S-Cube research framework and together represent the focus of research in software systems for the next decade.

1.2 Cross-Cutting Research Challenges

This section identifies typical research challenges arising from the Internet of Services that have helped to form the S-Cube research framework. The purpose of this section is not to be exhaustive but rather to highlight some of the representative research challenges S-Cube will address.

1.2.1 Considering Contextual Information for Service-Based Systems

The information about the context in which service-based systems are executed effects the expected behaviour and quality of the systems. To consider contextual information for service-based systems we face the following challenges:

1. Context modeling approaches are required to facilitate the description of the context, in which the service-based application is embedded. This requires understanding the different context facets, such as the business

context facet (e.g., stakeholders, regulations, business trends and business objects), the user context facet (e.g., end-user preferences and settings, as well as tasks and activities), the application operational context facet (e.g., protocols and networks, devices, and their properties) and so forth. The context description therefore needs to consider context facets ranging from the business (i.e., business process management) layer to the lower level operational context of service-based systems and the service infrastructure.

2. Using and managing context information. Context models are key to support the selection, realisation and enactment of adaptation actions through service engineering and design. As input for extending the existing and for defining novel monitoring and adaptation approaches that are capable of explicitly considering and reasoning upon such information through service adaptation and monitoring techniques, context models need to be exploited, collected, refined, and integrated.
3. Context issues in service discovery, selection, and negotiation. Another relevant impact of context can be observed in service discovery and registries which are devised by the service infrastructure layer and are exploited by composition and coordination techniques and methods. Specifically, feedback-based service discovery deals with finding the impact of human activities and social relationships among service users on the evaluation of the quality of experience in service consumption. Thus, concepts which allow for gathering, storing, exchanging and evaluating quality of experience metrics (which are also relevant for service quality) are needed. To address dynamic adaptation of service-based systems, a growing need for automating the selection, negotiation and agreement of quality attributes (e.g., as stipulated by SLAs) can be observed (also see Section 1.2.4 below). This issue requires considering — in an explicit form — user interaction and experience patterns developed for service engineering and design, as these may impact on the negotiation itself.
4. Context-driven adaptation and monitoring. Service-based systems should be equipped with the mechanisms necessary to quickly adapt to changes in the system's run-time. This requires identifying and codifying the relevant context information such that it can be monitored and exploited to trigger adaptations.

1.2.2 Cross-Layer and Pro-active Monitoring and Adaptation

Existing adaptation and monitoring approaches are not adequate for the purposes of future service-based systems. They are very fragmented and thus only address specific problems, particular application domains or particular types of systems. Often, the monitoring solutions are isolated from the adaptation needs and approaches. What is ultimately needed is a holistic framework for adaptation and monitoring, which leads to the following key research problems:

1. Cross-layer monitoring and adaptation. Integration of the monitoring approaches across the technology layers is crucial for future service-based application provisioning, as it provides a way to properly locate, evaluate and cross-correlate the source of the problem leading to an adaptation and its impact. Together with cross-layer adaptation, this will allow properly identifying and propagating the necessary adaptation activities in different elements of the service-based application, while avoiding conflicting adaptations. Cross-layer monitoring and adaptation will require the integration of currently isolated monitoring and adaptation mechanisms available at different functional layers into the holistic cross layer approach. In addition, the cross-correlation of different quality levels monitored across the different layers will become relevant (also see Section 1.2.3 below).
2. Cross-life-cycle integration of monitoring and adaptation. Cross-life-cycle integration requires monitoring and adaptation techniques that exploit synergies between the knowledge and mechanisms available at different phases of the life-cycle of the service-based application. This allows devising new monitoring approaches (e.g., exploiting post-mortem process analysis for prediction) or adaptation decision mechanisms (e.g., explore previous decisions and adaptation effects to select proper adaptation strategy). This problem needs to be considered by research into service engineering and design and service adaptation and monitoring — the latter providing the reference points of the relevant principles and approaches available at different phases of the service-based application life-cycle.
3. Proactive adaptation. In contrast to reactive adaptation (i.e., an adaptation that is performed after a deviation or critical change has occurred), proactive adaptation offers significant benefits, like not having to compensate for deviation. Therefore, there is a need to perform adaptation not only reactively, as in existing approaches, but also proactively. This will prevent negative and undesired situations by anticipating the decisions and adaptation activities. Realizing techniques for proactive adaptation will require consideration of the following issues:
   - A cornerstone of a proactive adaptation technique is the ability to predict critical changes in service-based application functioning and its delivered quality is needed. As an example, quality prediction can be addressed through novel run-time quality assurance techniques. Local quality assurance mechanisms and techniques of the technology stack — from business process management to service coordination and composition and service infrastructure — as well as techniques from software engineering constitute important inputs for those novel quality assurance techniques.
   - Based on the predicted quality values, the service-based application can be modified in advance. This would mean to adaptation techniques and mechanisms that are based on the predicted knowledge. In contrast to the existing approaches that select and realize adaptation strategies based on current information, novel approaches should provide a way

to make those decisions and realization while relying upon predicted, anticipated situations.

- To support proactive adaptation, proactive negotiation and agreement techniques are needed. Otherwise, effective run-time SLA negotiation will not be feasible, since negotiation implies significant computational costs.

1.2.3 End-to-End Quality Provision

Each functional layer and service provider contributes to the end-to-end quality of a service-based system. Thus, to assure end-to-end quality, the different layer and service quality characteristics (like reliability or performance) and their dependencies must be understood and individual quality contracts (e.g., as part of SLAs) need to be aggregated. However, to guarantee the end-to-end quality of service-based applications, we face the following research problems:

1. End-to-end quality definition. To enable end-to-end quality definition, an end-to-end quality reference model and an according quality definition language is key. This requires understanding and aligning the quality attributes relevant for to the technologies involved at the business process management, service composition and coordination and service infrastructure levels as well as the quality attributes relevant for software engineering and design. Such a model and language will enable the definition of end-to-end quality for service-based systems, which can be used as an requirements engineering techniques developed for service engineering and design and monitoring techniques and mechanisms developed through research into service adaptation and monitoring.
2. Aggregating quality levels across layers. The challenge is to achieve an understanding on how to aggregate quality levels stipulated in individual quality contracts (e.g., as part of SLAs) across layers and across networks of service providers and consumers. This will support assessing the end-to-end quality of a service-based application and will, for example, be relevant for cross-layer monitoring techniques. The starting point can be the quality attributes and their relationships as provided by the quality reference model.

1.2.4 Autonomic Service Infrastructure

To reduce the cost and improve the reliability of making changes to complex service-based systems, new technologies are needed which support automated, dynamic system adaptation via self-configuring architectural service models and performance-oriented run-time gauges. Self-(re)-configuring service-based systems can automatically leverage distributed service infrastructures and resources to create an optimal architectural configuration according to the requirements of the system characteristics. There are various adaptation mechanisms for bootstrapping and initial configuration of service infrastructures but there are few solutions that solve run-time adaptation by reconfiguration.

At the service infrastructure layer one significant challenge is to provide autonomic behavior for services, which would enable them to, for instance, remain healthy, conform to SLAs and make the optimal use of resources. In the general case, an autonomic infrastructure involves dealing with the following open research problems jointly with the other framework building blocks:

1. Autonomic infrastructure support. High-level policies and objectives are needed for establishing methods for decision making, realizing pro-active and reactive adaptation and studying collaboration with middleware and operating system level resource allocation. This involves, for instance, creating optimal infrastructures for service value networks at the business process management level. Also, existing research results from the autonomic/organic computing area need to be leveraged for service-based systems.
2. Automated quality support. Autonomic behavior of services and service-based systems also implies a growing need for automating the selection, negotiation and agreement of quality attributes and SLAs. Thus, novel automated quality negotiation and assurance techniques need to be devised.

1.2.5 Concepts, Languages and Mechanisms for Agile Service Networks

The detailed literature carried out by S-Cube revealed two key classes of broad research challenges in business process management of service-based systems that realize agile service networks (ASNs): design-time and run-time challenges. The survey identified a number of open problems for each of these classes of challenge:

1. Design-time concepts, mechanisms and languages for specifying, analyzing, and simulating end-to-end processes in agile service networks. This challenge requires improving our understanding of service engineering and design principles and methodologies, as well as quality definition and negotiation techniques. In addition, this challenge will be addressed in close alignment with ongoing research in service composition and coordination as well as the enabling service infrastructure. In particular, this issue involves at least overcoming the following three impediments:
   - Exploring, developing and validating effective techniques, concepts, languages and mechanisms for analyzing, modelling and simulating end-to-end business processes in ASNs. In particular, a deeper understanding of existing service engineering methodologies is required.

  - Developing and validating approaches for analysis and formal verification of business protocols involving bi-lateral and multi-lateral agreements between network nodes. Solutions should be grounded on existing approaches and techniques in protocol engineering, as well as devising quality of service schemes for service-based systems and Service Level Agreements.
  - Analysis and design techniques for business-aware transaction concepts and mechanisms to support business protocols in ASNs are typically very traditional in nature addressing traditional, short-running database transactions and thereby ignore important business semantics including multi-party agreements on QoS. This challenge is also related to research into service quality and service engineering and design.

2. Concepts, mechanism and languages for run-time monitoring of business transactions. Meeting this challenge will require a better understanding of existing adaptation and monitoring approaches, techniques and solutions. Some of these approaches are scrutinized in service adaptation and monitoring approaches. This challenge requires addressing the following open problems:
   - Existing transaction monitors typically limit themselves to detecting and aggregating system-level events. An integrated approach including mechanisms and concepts for monitoring and measuring business events raised by business-aware transactions and related protocols and processes is currently lacking. This research challenge will particularly benefit from ongoing research with regard to service adaptation and monitoring and business activity monitors in particular.
   - While existing business transaction monitors may be able to detect and measure system-level errors and anomalies in service-based systems, mechanisms and concepts for adapting business-aware transactions and related protocols and processes in ASNs are not yet effectively supported. In particular, the development of adaptable business-aware transactions can be based on existing adaptation techniques and methods.

1.2.6 Fragmentation of Service Compositions and Their Coordination

Business processes and service compositions realizing those processes can be created faster and at lower cost if process fragments are reused. This approach requires the separation and unique identification of reusable (sub-objective) content and their encapsulation in business process fragments (i.e., building blocks such as service patterns or templates) to rapidly tailor service compositions as users or individual application needs demand. This challenge introduces a number of open research problems:

1. Mechanisms for fragmentation. Reasons and criteria for the fragmentation of service compositions need to be identified (e.g., outsourcing, resource workload distribution and optimization, organizational (re-distribution) and relevant mechanisms for process fragmentation need to be developed. This topic requires service engineering and design principles and methodologies and is used for decomposition of complex service networks in business process management.
2. Reusable process fragments. Mechanisms for creating parameterised fragments from repeatable service compositions (and business sub-processes) are required, which are based on best practices facilitating application and systems delivery and development. Such reusable customized and/or differentiated service patterns can be offered by service providers to their customers. This topic requires service engineering and design principles and methodologies.
3. Coordination of fragments. There is a lack of coordination protocols to maintain the original composition logic of decomposed processes. Depending on the fragmentation reasons and criteria, as well as on the fragmentation mechanisms, the coordination protocols may be different. These protocols can be used for coordination of business transactions at the business process management layer.
4. Cross-layer adaptation support. Fragmentation may lead to conflicts across the technology layers during an adaptation of a service-based application. Thus, a deep understanding of the fragmentation techniques and strategies is needed to support devising the cross-layer adaptation strategies (see Section 1.2.2 above).

1.2.7 Coherent Lifecycle for Adaptable and Evolvable Service-Based Systems

In Section 1.5.1, we will describe in more detail the importance of a life-cycle for service-based systems and demonstrate how the S-Cube life-cycle model focuses on all aspects concerned with the adaptation of such systems. The life-cycle models for SBAs that are currently presented in the literature are mainly focused on the phases that precede the release of software and, even in the cases in which focus on the operation phases, they do not consider the possibility for SBAs to adapt dynamically to new situations, contexts, requirement needs, service faults, and the like. Moreover, they do not seem to pay much attention to the importance of humans both in the application development and operation phases. There are some initiatives that aim at supporting so called Human-Provided Services. However, to this day, there has been little intersection between research in service-centric systems and Human-Computer Interaction (HCI). Thus, many research issues need to be considered and addressed. The ones we consider most relevant are the following:

1. Requirements elicitation and design for adaptation. Adaptation can be addressed both on the fly, while the SBA is being executed, or it can require a new design and development cycle (in this case we talk about evolution). The conception of the Agile Service Network (see Section 1.2.5 above) through to the implementation of the corresponding composition and the infrastructure has to be designed and developed in such a way that it is able to recognize an adaptation requirement and to act accordingly. This means that not only the application logic needs to be analysed, designed, and developed, but also the context in which the system is executed (see Section 1.2.2 above) as well as the rules that will allow the identification of the adaptation and evolution strategy to be chosen (see Section 1.2.4 above). The effects of these issues encompass also the requirements engineering phase. If classical requirements elicitation can be simplified so the system can work even in the presence of missing or misunderstood requirements, new kinds of requirements, i.e., requirements for adaptation, need to be identified and must result in a corresponding implementation. In general, the effects of designing for adaptation on the system life-cycle are only partially understood, if at all.
2. Extended operation phase. The operation phase is not only restricted to the simple execution and monitoring of the application, but also requires adaptations to be identified and the corresponding adaptation strategies to be enacted (see Figure 1.3 and Section 1.2.2 above). The way the operation and the adaptation cycle are managed depends also on the autonomic features offered by the underlying infrastructure (see Section 1.2.4 above) and on their programmability. In principle, this last aspect can have an impact also on the way the application is designed, deployed, and configured.
3. Incorporate end-to-end quality within the life-cycle: Quality assurance has an impact on all aspects of the life-cycle. Therefore, the issues highlighted in Section 1.2.3 above have to be properly incorporated into the various phases of the SBA life-cycle. Quality characteristics to be assessed and ensured should be identified since the requirement analysis phases and should concern not only functional but also non-functional characteristics. Moreover, similar to the application code, these quality characteristics change over time, and their changes have to be re-identified and re-assessed continuously.
4. Control and improve the life-cycle. It is normal practice in software engineering to apply approaches to improve development and operation processes. In the area of service-based systems, little attention has been paid so far to this aspect. However, we feel that, especially to foster the adoption of the service-based approach in companies working in regulatory environments, we need to ensure that all development and operation activities in the life-cycle are somehow measured and kept under control, and process improvement approaches are put in place (which may seem contrasting with the adaptation requirement). Thus, understanding the

interplay between life-cycle control and improvement on the one hand, and application adaptation on the other hand becomes a critical aspect.

5. Develop HCI knowledge for service engineering. Human specificities, diversity, and tasks characteristics need to be taken into account when engineering service-based systems. This requires at least (1) the identification of HCI knowledge that delivers enhanced or new capabilities for service-based systems and (2) the codification of this knowledge for its application to the development and use of service-based systems.

1.3 The S-Cube Research Framework

Having described the context for research into software services this section describes the S-Cube research framework, which is organized around the six building blocks depicted in Figure 1.1. These blocks are organized in two: the *service technology layers* and the *service techniques and methods planes*. The technology layers consist of Service Infrastructure (SI), Service Composition & Coordination (SCC) and Business Process Management (BPM), whilst the service techniques and method planes contains Service Adaptation & Monitoring (SAM), Service Engineering & Design (SED) and Service Quality Definition, Negotiation & Assurance (SQ).

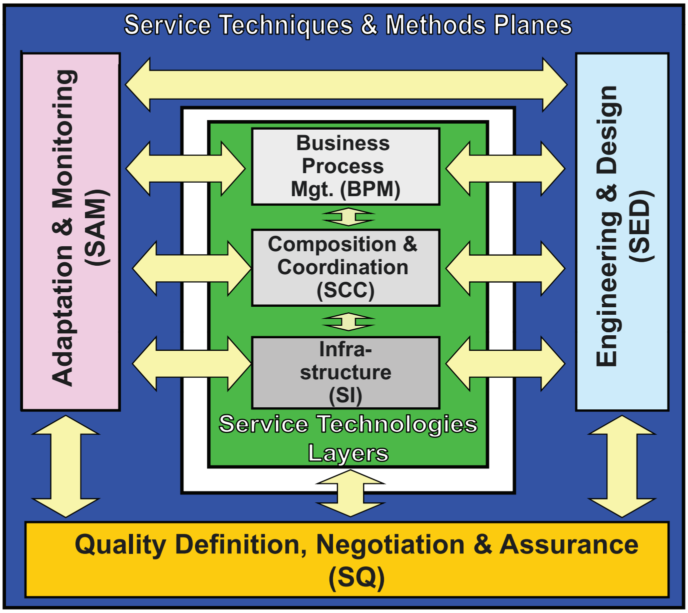

Service Techniques & Methods Planes
Adaptation & Monitoring (SAM)
Engineering & Design (SED)
Business Process Mgt. (BPM)
Composition & Coordination (SCC)
Infra-structure (SI)
Service Technologies Layers
Quality Definition, Negotiation & Assurance (SQ)

Fig. 1.1. Overview of the S-Cube Research Framework

The benefit of adopting such an approach is that the S-Cube research framework provides a clear distinction between technology-focussed approaches of the service technology layers and the cross-cutting principles, techniques and methods provided by the techniques and method planes that together exploit and integrate the capabilities of the technology layers. Each of the elements in the framework is now briefly described with reference to where the full description and treatment of each research area can be found in this book.

Service Technologies

- Business Process Management (BPM): The BPM layer provides mechanisms for expressing, understanding, representing and managing enterprises that are organized in service networks, which may be loosely defined as a large, geographically dispersed and complex networks of collaborating and transacting value-adding services. Service networks furnish a collection of business processes built on top of composed services by the SCC layer, and are responsive to the business environment of internal or external events. Service networks use BPM facilities to coordinate work between people and systems, with the ultimate goal of improving organizational efficiency, responsiveness and reliability, strategic measures (business performance management), and their correlation as the basis for continuous process improvement and innovation. The state of the art and future research directions for this aspect of service technologies are described in Chapter 2.
- Service Composition & Coordination (SCC): The SCC layer encompasses the functions required for the aggregation of multiple services into a single composite service offering. The execution of the constituent services in a composition is controlled through the SI layer. In addition to managing the control flow, this layer also manages data flow between the services in a composition, for example by specifying workflow models and using a workflow engine for runtime control of service execution. SCC technologies and research are set out in Chapter 3.
- Service Infrastructure (SI): The SI layer represents the most basic layer of the S-Cube framework and supports services communication primitives and utilizes service middleware and architectural constructs that connect heterogeneous systems, provide multiple-channel access to services, and introduces a runtime environment for the execution of services. The SI layer provides infrastructure capabilities for defining basic communication patterns and interactions involving the description, publishing, finding and binding of services. It also provides facilities for analyzing and aggregating historical data to support, for example, predictive adaptation. A full description of this research area and S-Cube's activities in this domain is given in Chapter 4.

Service Principles, Techniques & Methods

- Service quality definition, negotiation and assurance (SQ): This plane involves principles, techniques and methods for defining, negotiating and assuring end-to-end quality and conformance to SLAs. It provides novel facilities to detect problems and deviations in systems before quality is impacted and triggers the adaptations coordinated by the SAM plane to pro-actively respond to these situations. To detect and resolve problems in service-based systems, different quality attributes (i.e., quality characteristics) need to be measured for a service, including throughput, utilization, service availability, response time, transaction rate, service throughput, service idle time, service usability and so on. End-to-end quality provision implies that those different quality attributes must be understood and correlated across the technology stack layers: service infrastructure, service composition, and business process management. The modeling, specification and analytical analysis of qualities of service in S-Cube are described in Chapters 5 and 6.
- Service Engineering and Design (SED): The SED plane provides the principles, techniques and methods that interweave and exploit the mechanisms provided by the technology stack with the aim of developing high-quality service-based systems. The SED plane provides requirements engineering and design principles and techniques, which – in conjunction with context, HCI and quality knowledge – help to create high-quality service-based systems. For example, the SED plane provides specifications that guide the service composition and coordination layer in composing services in a manner that guarantees that the composition does not produce spurious results and that the overall system behaves in a correct and unambiguous manner. Similarly, the SED plane provides specification to the BPM and SAM layers. Moreover, the SED plane aims at establishing, understanding and managing the entire service lifecycle, including identifying, finding, designing, developing, deploying, evolving, quality assuring, and maintaining services. Further details about how this area supports the technology layers can be found in Chapter 7.
- Service Adaptation and Monitoring (SAM): Service-based systems should possess the ability to continuously adapt themselves in reaction to contextual changes, such as evolving user or customer requirements or the appearance of new services. In addition, service-based systems should also possess the ability to predict problems, such as potential degradation scenarios and erroneous behavior (e.g., exceptions/deviations from expected behavior) and move toward resolving them. The SAM plane supports monitoring, predicting and governing the activities of a distributed services-system and performing control actions to adapt the entire services technology stack, for example in cases where individual services need to evolve and adapt their functionality and characteristics to be able to interoperate with other services. S-Cube's approach to the evolution and adaptation of service-based systems is described in full in Chapter 8.

1.4 The Interaction View

The SED, SQ and SAM planes operate in close concert with the BPM, SCC and SI layers to handle evolving and adaptable services and service-based systems. Figure 1.2 depicts the interactions between the three planes and the three technology layers. In the following sections we briefly characterize these interactions between the building blocks of the S-Cube framework.

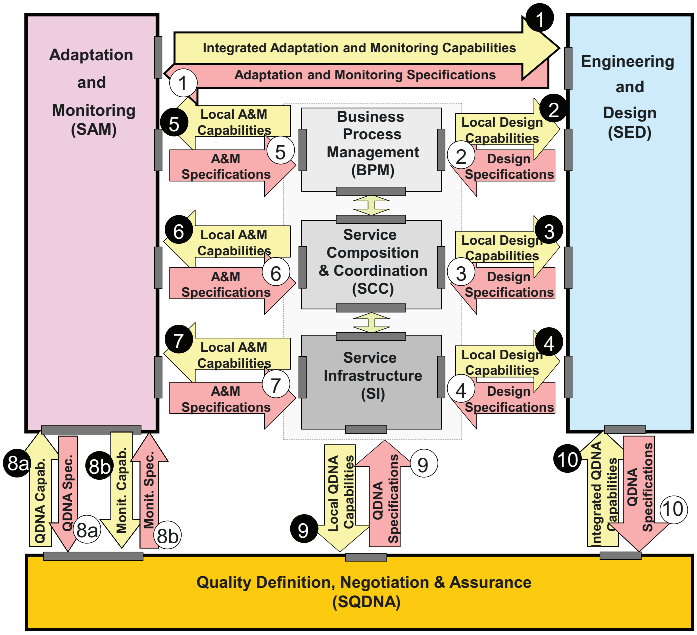

Adaptation and Monitoring (SAM)
Integrated Adaptation and Monitoring Capabilities
Adaptation and Monitoring Specifications
1
Engineering and Design (SED)
Local A&M Capabilities
A&M Specifications
5
Business Process Management (BPM)
Local Design Capabilities
Design Specifications
2
6
Service Composition & Coordination (SCC)
3
7
Service Infrastructure (SI)
4
8a
QDNA Capab.
QDNA Spec.
8b
Monit. Capab.
Monit. Spec.
9
Local QDNA Capabilities
QDNA Specifications
10
Integrated QDNA Capabilities
QDNA Specifications
Quality Definition, Negotiation & Assurance (SQDNA)

Fig. 1.2. Interactions between the Elements of the S-Cube Research Framework

1.4.1 SED Interactions with Technology Layers

The service engineering and design plane provides methodological guidance to the three technology layers in order to help develop sound, complex, and continuously evolving service-based systems. Taking the capabilities and features offered by the three technology layers into account, the integrated SED techniques and methods specify a service-based application. Resulting restrictions and constraints for the mechanism and capabilities provided by the technology layers are passed to the corresponding layers which adjust their behavior accordingly. An example for such a restriction is the disabling of a mechanism provided by the SI plane (e.g., virtualization of the infrastructure) for a particular service-based application. In addition, if adaptations of the SAM plane do not suffice, the SED plane also supports the evolution of a service-based system (e.g., by means of re-design) based on evolution requests received, e.g., from the three technology layers.

SED & Business Process Management (BPM)

Approaches and mechanisms for Agile Service Networks (ASN) and business process definitions developed in the BPM layer are incorporated and used in requirements engineering and high-level design techniques of the SED plane. In addition the BPM layer provides the techniques and methods of the SED plane with knowledge on mechanisms for detecting the need for adaptation at the business level (❷). Further, the SED plane aims at aligning ASN and SCC and provides guidelines as how to construct end-to-end service networks supporting the business processes (②). Such information describes all the resources used to generate the process outcome, including people, technology, procedures and any other resources linked together for the capability's specific purpose. For example, it provides knowledge about transaction volumes, the number and value of transactions per business process step, how long it takes to deal with transactions between process steps, process cycle times, wait-time between events, and so on.

SED & Service Composition and Coordination (SCC)

To allow the creation of meaningful service compositions that realise business processes, the SCC layer provides a service composition meta-model and service composition techniques to the SED plane (❸). To assist in composing services in a manner that guarantees consistency (i.e., so composite services do not lead to spurious results and that the overall process behaves in a correct and unambiguous manner) the SED plane specifies the use of the mechanisms in the SCC layer (③). During the operation an evolution of the service-based application might be required, for example, to adjust the application to contextual changes. Thereupon, (re-)design activities, which produce updated or even new specifications for the SCC layer are initiated. (Also see the lifecycle description in Section 1.5.1).

SED & Service Infrastructure (SI)

To operate a service-based application in the most appropriate run-time environment, the main input from the SI layer to the SED plane is knowledge about infrastructural services, e.g., service discovery or search facilities, which are in turn used to guide the lifecycle's design and runtime phases (❹). The SED techniques and methods provide specifications to the SI layer which restrict the SI mechanisms and capabilities for describing, publishing and discovering services and on how to run that service-based application (④).

Thereby, the SI layer, under consideration of the specifications from the SED plane, chooses the most appropriate architecture and run-time environment for the service application.

1.4.2 SED Interactions with Service Techniques & Methods Planes

To drive the continuous adaptation of a service-based application across all three technology layers and to guarantee a constant quality of the application, the interactions between the SED plane and the SAM and SQ planes are as follows:

SED & Adaptation and Monitoring (SAM)

The Adaptation and Monitoring plane provides monitoring and adaptation principles, techniques and methods. It communicates their capabilities to the SED plane. These capabilities are taken into account when designing the service-based system (❶). In turn, the SED plane specifies which monitoring and adaptation principles can be used for the service-based system at hand (①). Also, the SED plane provides the SAM plane with codified knowledge about the application context and about user types, which will be exploited for monitoring the context of the service-based application as well as for pro-actively adapting the application. This means that the SED plane is responsible for "design for adaptation", while the SAM plane enacts adaptation.

SED & Service Quality Definition, Negotiation and Assurance (SQ)

The SQ plane provides techniques to ensure the quality of a service-based application during its lifecycle. It communicates the capabilities of those techniques to the SED plane (❿). The SED plane takes this knowledge into account when designing the service-based application. It specifies which quality definition, negotiation and assurance techniques are used during the design of the service-based application (⑩). In addition, the SQ plane will offer a quality reference model and an end-to-end quality modeling language together with quality negotiation techniques, which are envisioned to be integrated in the lifecycle of service-based systems (addressed in the SED plane). Further, the SED plane provides HCI knowledge in the form of user and task models as well as knowledge about potential systems contexts to the SQ plane which exploits this knowledge, e.g., when devising automated negotiation techniques.

1.4.3 SAM Interactions with Technology Layers

The adaptation and monitoring plane describes interactions with the three technology stack layers and what type of knowledge needs to be provided to these three layers in order to control, to steer and to successfully adapt complex service-based systems. The current convention is that each technology layer focuses separately on providing local adaptation mechanisms. However, in S-Cube the adaptation mechanisms employed by SAM adopt a holistic view of all local perspectives and cross-correlate events and data to provide a unified and more complete solution to service adaptation problems that span more than one technology layer.

The SAM plane receives monitoring events and data gathered and produced by each of the three technology layers. The SAM plane analyses those events and data and decides whether the service-based systems needs to be adapted and if so, how the system should be adapted. In the case of an adaptation, the SAM plane steers the three technology layers by invoking the adaptation mechanisms across the three layers needed to achieve the required adaptations. In essence, the SAM plane takes a logically centralised decision (involving human actors if needed) about the adaptation based on analysing and cross-correlating technology layer-specific monitoring events and data. It thereby rectifies partially overlapping and even conflicting adaptations introduced by each of the local layers.

SAM & Business Process Management (BPM)

The efficient management of services requires new techniques and methods in monitoring large networks. The SAM plane uses knowledge of business events (❺) that cause changes to processes in service networks to recognize and respond to these events and manage process changes (⑤). The SAM plane detects the event patterns that indicate potential process changes and correlates the relationships between possibly independent or isolated events and event-driven processes, such as causality, membership and timing.

SAM & Service Composition and Coordination (SCC)

Based on the monitoring events and data provided by the SCC layer (❻) as well as other monitoring events and data obtained from the BPM and SI layers, the SAM plane derives specifications to ensure that service compositions are able to function in spite of changes of constituent services in the composition and in spite of context changes (⑥). The intention is to catch and repair faults and predict behavior (to the extent this is possible) so as to reduce as much as possible the need of human intervention in adapting services to subsequent changes, while guaranteeing QoS and SLA requirements. During the operation of the service-based application, concrete monitoring events are provided by the SCC layer. These are analyzed and cross-correlated by the inter-layer adaptation techniques in the SAM plane taking into account monitoring events from the other technology layers. If required, this leads to adaptation triggers which are enacted by the SCC layer.

SAM & Service Infrastructure

The SAM plane typically gathers information about the service platform, managed resource status and performance (e.g., root cause analysis, SLA monitoring and reporting, service deployment, and lifecycle management and capacity planning) and communicates this information to the SAM layer (❼).Vice versa, the SAM plane communicates adaptation request to the SI layer (⑦). The SAM plane has the knowledge to detect system malfunctions and initiate policy-based corrective actions without disrupting the service infrastructure. Corrective actions could, for example, involve a server altering its state or effecting changes in other components in the service infrastructure as a result of a changing service composition that may create uncharacteristically high processing loads. In this way, service-based solutions — as a whole — become more resilient because day-to-day operations are less likely to fail. Further, triggered by the SAM plane, the SI layer supports the automatic tuning of service resources. For example, a tuning action could lead to the reallocation of resources — such as in response to dynamically changing workloads — to improve overall utilization, or ensuring that particular business processes or transactions can be completed in a timely fashion.

1.4.4 SAM Interactions with Service Techniques & Methods Planes

In addition, we envision interactions of the SAM plane with the SED and SQ planes in order to control, steer and successfully adapt complex service-based systems.

SAM & Service Engineering and Design (SED)

The SED and SAM planes work closely together to jointly handle adaptive services and service-based systems by exchanging relevant information. To achieve this synergy, the SAM plane contains knowledge required to predict, sense and respond as required for service and process adaptability. In a sense, SED designs for adaptation, while SAM provides techniques to enact adaptation. These interactions are described in more detail from the perspective of the SED plane in Section 1.4.2.

SAM & Service Quality (SQ)

These interactions are described in Section 1.4.6 from the perspective of the SQ plane.

1.4.5 SQ Interactions with Technology Layers

The SQ plane focuses on achieving end-to-end service quality and SLA conformance and thus complements the SED and SAM planes. It provides assurances with respect to relevant quality guarantees (e.g., KPIs) that drive an end-to-end service composition. In general, the SQ plane integrates and correlates the quality attributes, also known as quality characteristics, and the local quality negotiation and assurance techniques provided by each of the three technology layers (SI, SCC, BPM) (❾). This will result in the S-Cube quality reference model and end-to-end quality definition language. Based on the reference model and language, novel quality assurance techniques and methods of the SQ plane will provide guidance to all three technology layers to measure, negotiate and assure, among others, the quality of business processes, the quality of service (QoS) and the quality of information (⑨).

SQ & Business Process Management (BPM)

The BPM layer provides knowledge of how quality characteristics of running processes could be expressed in the form of real-time and historical reports (❾). The techniques of the SQ plane, together with the mechanisms of the BPM layer (⑨) enable the detection of deviations from KPI target values, such as the percent of requests fulfilled within the limits specified by a SLA, and may trigger an alert and an escalation procedure, or even propose changes to affected process models (i.e., trigger an adaptation in cooperation with the SAM plane) to allow them to achieve their goals. The BPM layer will provide mechanisms for keeping metrics aligned to corporate objectives in order to help understand how to continually improve processes and underlying IT resources to most effectively contribute to the organization's overall goals.

SQ & Service Composition and Coordination (SCC)

By exploiting the quality reference model, the SQ plane provides information to help achieve QoS-aware service compositions (⑨). This requires understanding and respecting composed service policies, performance levels, security requirements, SLA stipulations, and so forth. An illustrative example can be provided when considering security as a quality attribute. Here, knowing a newly composed service adopts a Web services security standard, such as one of the WS-Security specifications, would not be enough information to enable successful composition. The client also needs to know if the services in the business process actually require WS-Security, what kind of security tokens they are capable of processing, and which ones they prefer. Finally, the client must decide on when to encrypt the messages, which algorithm to use, and how to exchange a shared key with the service. For example, a purchase order service in an order management process may indicate that it only accepts username tokens that are based singed messaged using X.509 certificate that is cryptographically endorsed by a third party (❾).

SQ & Service Infrastructure (SI)

We anticipate the SQ plane will exploit service infrastructure data (e.g., performance statistics) (❾) to support the assessment of platform effectiveness, permit complete visibility into individual service compositions, guarantee consistency of service compositions, and ultimately assure end-to-end SLAs. This will require a better visibility into individual service compositions, which in turn will require new infrastructure capabilities, like dedicated test interfaces for services (⑨). To allow quality prediction techniques from the SQ plane, it is envisioned that novel infrastructure technology, e.g., reputation systems, will be exploited.

1.4.6 SQ Interactions with Technology Planes

SQ & Service Engineering and Design (SED)

The SQ plane complements the SED plane in developing complex and continuously evolving service-based systems. These interactions are described in more detail from the perspective of the SED plane in Section 1.4.2.

Interactions with SAM

The SQ plane provides to the SAM plane quality prediction techniques capabilities that facilitate the development of integrated, proactive adaptation techniques (❽a). To support pro-active adaptation, the SQ plane further provides pro-active negotiation techniques and offers an end-to-end quality definition language used, among others, by the cross-layer monitoring techniques devised by the SAM plane. Vice versa, the SAM plane stipulates specifications for the quality definition, prediction and negotiation techniques of the SQ plane (⑧a). In the opposite direction, SAM provides to SQ integrated monitoring capabilities for the purpose of checking at run-time that expected quality is met (⑧b). Analogously, monitoring specifications are passed from the SQ to SAM (❽b). Synergies between monitoring and other quality assurance techniques (like testing or model analysis) are envisioned to be jointly explored between the SAM and the SQ planes.

1.5 The Lifecycle & Runtime Views

To elaborate on the research vision some more, this section describes two further views of the research framework and activities of the S-Cube network:

- In Section 1.5.1 we elaborate on the lifecycle view on our S-Cube research framework which focuses on analyzing, identifying, designing, developing, deploying, finding, provisioning, evolving, and maintaining service-based systems.

- In Section 1.5.2 we sketch the run-time view of our research framework which structures the implementation, deployment, and management of distributed service-based systems with a focus on assembling, deploying, and managing those systems.

1.5.1 The Lifecycle View

The S-Cube research framework places emphasis on avoiding the pitfalls of deploying an uncontrolled maze of services and therefore provides a holistic and solid approach for orderly service development so services can be efficiently combined into service-based systems. The S-Cube framework views service-based systems as an orchestrated set of service interactions and it adopts a broader view of its impact on how the service-based solutions are designed, what it means to assemble them from disparate services, and how deployed services-oriented systems can evolve and be managed. This requires addressing common concerns such as the identification, specification and realization of services, their flows and composition, as well as ensuring the required quality levels.

In their early use of SOA, many enterprises assumed that they could port existing components to act as Web services by merely creating wrappers and leaving the underlying component untouched. Since component methodologies focus on the interface, many developers assume that these methodologies apply equally well to service-oriented environments. As a consequence, implementing a thin SOAP/WSDL/UDDI layer on top of existing systems or components that realize the services is by now widely practiced by the software industry. Yet, this is in no way sufficient to construct commercial strength service-based systems. Unless the nature of the component makes it suitable for use as a service, and most components are not suited to this, for instance, because they are tightly coupled to other components, it takes serious thought and redesign effort to properly deliver a component's functionality through a service. While relatively simple Web services may be effectively built that way, a service-based development methodology is required to specify, construct, refine and customize highly flexible service-based systems from internally and externally available components and services. More importantly, older software development paradigms for object-oriented and component-based development cannot be blindly applied to SOA and services.

The service lifecycle model envisioned by the S-Cube framework relies on a twin development and adaptation cycle. The development cycle addresses the classical development and deployment lifecycle phases, while the second cycle extends the classical lifecycle by explicitly defining phases for addressing changes and adaptations (see Figure 1.3).

The S-Cube service lifecycle model builds on established practices from software engineering and captures a highly iterative and continuous method for developing, implementing, and maintaining services in which feedback is continuously cycled to and from phases in iterative steps of refinement and

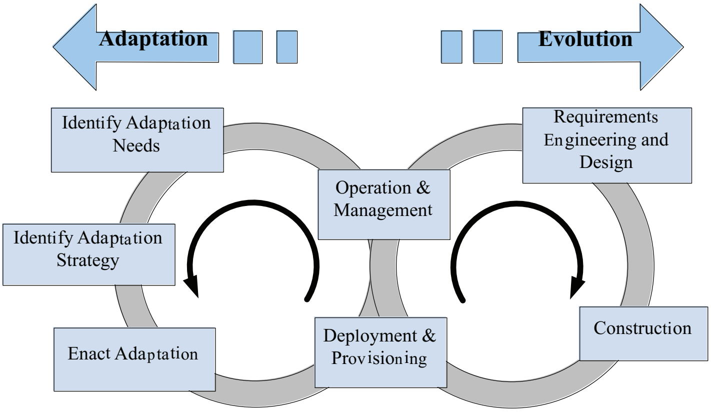

Adaptation
Evolution
Identify Adaptation Needs
Identify Adaptation Strategy
Enact Adaptation
Operation & Management
Deployment & Provisioning
Requirements Engineering and Design
Construction

Fig. 1.3. The S-Cube Lifecycle View

adaptations of all three layers of the technology stack. To that effect the method facilitates designing solutions as assemblies of services in which the assembly description is a managed, first-class aspect of the solution, and hence, amenable to analysis, change, and evolution. The method accommodates continuous modifications of service-based systems and its quality (e.g., QoS and KPIs) at all layers. Continuous modifications (evolutions and adaptations) are based on monitoring and measurement of service execution against SLAs and quality goals. In this way, the S-Cube reference lifecycle is enabled to continuously a) detect new problems, changes, and needs for adaptation, b) identify possible adaptation strategies, and c) enact them. These three steps are depicted on the left hand side of Figure 1.3. Once service-based systems (or parts thereof) have been adapted, they will be re-deployed, re-provisoned and put into operation.

More details on the S-Cube lifecycle can be found in S-Cube deliverable CD-JRA-1.1.2 “Separate design knowledge models for software engineering and service-based computing” [1].

1.5.2 The Runtime View

S-Cube's run-time view depicts a conceptual run-time architecture which structures the implementation, deployment, and management of distributed service-based systems with a focus on assembling, deploying, and managing those systems. The run-time architecture supports service invocations, message, and event-based interactions with appropriate service levels and manageability (see Figure 1.4). In essence, the S-Cube run-time architecture envisions providing containers and engines for hosting and providing services that can be assembled and orchestrated and are available for use by any other service on the communication backbone. Once a service is deployed into a service container it becomes an integral part of the S-Cube run-time architecture and thus can be used by any application or service. The service container hosts, manages, and dynamically deploys services and binds them to external resources, e.g., data sources, enterprise and multi-platform systems. The different engines (e.g., composition, monitoring, adaptation, etc) work in tandem to execute a service-based application. These engines rely heavily on techniques and methods provided by the SED, SAM and SQ planes.

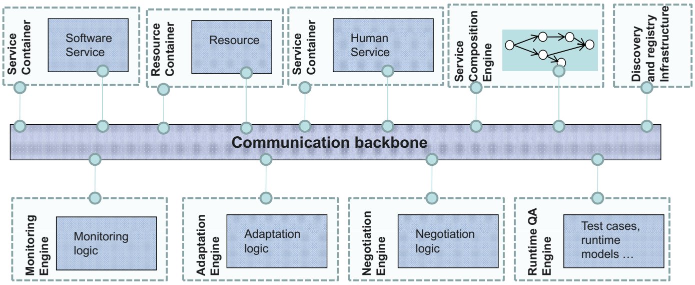


Fig. 1.4. The S-Cube Runtime View

The communication backbone facilitates the communication among any kind of service, regardless of whether the service is a core service or an application specific service. In particular, the communication backbone allows access to both core services and application specific services which are deployed within a container. An application specific service exposes a dedicated interface to access the application specific functionalities provided by the service. In contrast, a core service offers a management interface for controlling the behavior of the container or, in particular, for the deployment and operation of an application specific service. Examples for core services are an engine for executing service compositions, a discovery service, or an engine for monitoring the behavior of a service-based application. Exposing core services and application services on the same communication backbone enables interactions between the two logical levels and is one of the key elements of the conceptual run-time architecture proposed.

More details on the S-Cube run-time view can be found in S-Cube deliverable CD-IA-3.1.1 “Integration Framework Baseline” [2].

1.6 Adaptive Services in Context

The software services studied by S-Cube pose research challenges in contexts of applications characterized by high variability, dynamic service invocation, quality of service requirements, and flexibility and evolution. One of the goals of S-Cube is to analyze the research results within cases studies characterized by the use of services in a flexible way, derived from an analysis of real case studies n which the above mentioned needs are particularly relevant.

The case studies in S-Cube are derived from industrial experiences within the NESSI platform[3] and from industrial experiences presented by the S-Cube partners. However, within S-Cube the need is not only to apply research results within a real or realistic case study, but also to analyze the characteristics of these case studies in order to compare the results obtained from validation. Therefore, in order to make the research results comparable, a methodology suited for the description of case studies in service-based applications has been developed, together with strategies to link the validation of research results to the research challenges and research questions defined within the S-Cube Integrated Research framework. This section briefly describes the case-studies developed for service-based applications in S-Cube, with an introduction to the validation process.

1.6.1 Case Studies for SBA and Their Documentation Process

Five case studies have been studied, each focusing of different flexibility and adaptation needs:

1. Vineyard management and wine production.
2. A complex and geographically distributed supply chain from the automotive sector.
3. E-Health and, in particular, the management of Complex Diagnostic Workflows.
4. E-Government.

In the first case study, the focus is on two aspects of wine production and distribution: harvesting of the grapes and the logistics to deliver the product to retailers. Harvesting of grapes, observing the vineyard parameters and reacting to critical conditions that may happen during the cultivation phase are crucial, and supported by a service-based adaptive sensor network. Critical conditions may be represented by overcoming the threshold for some particular environmental parameter. In the product delivery phase, a flexible distribution network is envisioned, supported by agile service networks, and with the need of supporting business transactions.

In the automotive case study, the focus a flexible approach which allows different distribution logistics providers to participate in the SBA to provide the transportation of finished products from the manufacturing factory to the warehouses, and from the warehouses to the retail customers. The providers are dynamically selected according to the transportation routes and rules. The evaluation of KPIs in processes and linking them to quality of service in service compositions, when some tasks are outsourced, poses new research challenges in the management of these SBAs.

[3] http://www.nessi-europe.com/

The e-health case study seeks to provide new kinds of services and to integrate new and existing services, to support the work of healthcare professionals. In particular, this case study takes the viewpoint of the medical staff and the patient during a diagnostic workflow. The focus is on integration issues arising in dynamically evolving SBAs and in developing efficient processes in a distributed users environment. In such a context critical issues concerning security aspects are also considered.

In the e-government case study the goal is to improve the efficiency of public sector, avoiding the time spent traveling to different offices or waiting in queues, resulting in an improvement of the offered services and better accessibility and transparency of the public services. The focus here is on a seamless integration of services provided within a given geographical area, with guaranteed service and security levels.

1.6.2 Validation Methodology

A validation methodology for the results of S-Cube has been developed based on the case studies introduced above.

For each case study, a description is provided in terms of *business goals*, expressing the main purposes of some system in the terms of the business domain in which the system will live or currently lives; *domain assumptions and constraints*, reporting properties of the domain or restrictions on the design of the system architecture; *domain description*, phenomena occurring in the world together with the laws that regulate such a world; *abstract scenario description* as a way to describe world phenomena, which correspond to concrete detailed scenarios in the case study.

The definition of detailed scenarios helps to validate the research of S-Cube by relating each scenario to the research challenges introduced earlier in this chapter. Each scenario is described in terms of involved actors, a detailed operational description, problems and challenges posed by the scenario, non-functional requirements and constraints.

1.7 Chapter Conclusions

S-Cube sets out to address cross-cutting research challenges faced when engineering, designing, adapting, operating and evolving the next generation of services and service-based systems for the Internet of Services. The research in S-Cube is guided by the S-Cube research framework, which clearly distinguishes between principles and methods for engineering and adapting service-based systems and the technology and mechanisms which are used to realize those systems, while taking into account cross-cutting issues. By synthesizing and integrating diversified knowledge across different research disciplines, S-Cube aims at delivering the novel principles, techniques and methods for the service-based systems of the future.

The remainder of this book is structured as follows:

- Service Technologies
  - Chapter 2: Business Process Management,
  - Chapter 3: Service Composition & Coordination,
  - Chapter 4: Service Architectures & Infrastructures.
- Service Engineering Techniques & Methods
  - Chapter 5: Modeling & Specification of Quality of Service,
  - Chapter 6: Analytical Quality Assurance,
  - Chapter 7: Service Engineering,
  - Chapter 8: Adaptation of Service-Based Systems,
  - Chapter 9: Service Automation Aspects.

References

1. Andrikopoulos, V. (ed.): Separate Design Knowledge Models for Software Engineering & Service-Based Computing. Contractual Deliverable CD-JRA-1.1.2, S-Cube Network of Excellence (March 2009)
2. Pistore, M. (ed.): Integration Framework Baseline (including initial definition of interfaces among layers). Contractual Deliverable CD-IA-3.1.1, S-Cube Network of Excellence (March 2009)

2

Business Process Management

Francois Hantry[1], Mike Papazoglou[2], Willem-Jan van den Heuvel[2], Rafique Haque[2], Eoin Whelan[3], Noel Carroll[3], Dimka Karastoyanova[4], Frank Leymann[4], Christos Nikolaou[5], Winfried Lammersdorf[6], and Mohand-Said Hacid[1]

[1] Université Claude Bernard, Lyon 1, France
[2] Tilburg University, The Netherlands
[3] Lero – the Irish Software Engineering Research Centre, Ireland
[4] University of Stuttgart, Germany
[5] University of Crete, Greece
[6] University of Hamburg, Germany

Chapter Overview. Business process management is one of the core drivers of business innovation and is based on strategic technology and capable of creating and successfully executing end-to-end business processes. The trend will be to move from relatively stable, organization-specific applications to more dynamic, high-value ones where business process interactions and trends are examined closely to understand more accurately an application's requirements. Such collaborative, complex end-to-end service interactions give rise to the concept of Service Networks (SNs).

This book chapter surveys business process management, concentrating on business transactions, and introduces a business transaction language to realizes a novel business transaction model that enables end-to-end service constellations to behave according to agreed-upon transaction criteria. The objective of the BTL is to provide the environment to build robust and successful mission-critical SBAs, using a fusion of concepts from application integration, transaction-based and business process management technologies.

2.1 Introduction: Towards Business Transaction Management

Over the past decade, Business Process Management (BPM) emerged as both a management principle and a suite of software technologies focusing on bridging diverse systems, organizations, people, and business processes [7]. BPM is an information technology-enabled management discipline that treats business processes as assets to be valued, designed and enhanced in their own right. BPM technologies support both human-centric processes (e.g., claims processing, accounts payable or customer servicing) and system-intensive pro-

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 27–54, 2010.

cesses (straight-through processing or trade settlement), as well as a mixture of both (the granting of loans).

BPM is the capability to discover, design, deploy, execute, interact with, operate, optimize and analyze end-to-end business processes, and to achieve this at the level of business design (modeling, designing, simulating and redesigning business processes) and not purely technical implementation. We may therefore define BPM as: a strategy and associated technology for managing and improving the performance of an enterprise or value-chain through continuous monitoring and optimization of business processes in a closed-loop cycle of modeling, execution, and measurement.

BPM tools are far less effective without a method and strategy for defining, measuring, and improving processes [8]. BPM is a structured approach that aims to improve agility and operational performance and it represents a fundamental change in how organizations manage and run their operational business processes. It treats business processes as organizational building blocks with a focus in managing the efficiency and effectiveness of business processes throughout the organization or integrated supply chain.

The ideas behind modern BPM are not new, though the term itself was only introduced in the early 2000s. BPM follows initiatives established throughout the 1980s and 1990s such as Total Quality Management (TQM), Business Process Reengineering (BPR), Enterprise Resource Planning (ERP) and Enterprise Application Integration (EAI). These methodologies attempt to improve the performance of enterprises through measurement, restructuring, automation, and business process integration techniques.

The trend in BPM will be to move from relatively stable, organization-specific applications to more dynamic, high-value ones where business process interactions and trends are examined closely to understand more accurately application needs, most notably requirements with regard to business transactions, and dynamics. Such collaborative, complex end-to-end service interactions give rise to the concept of Service Networks (SNs)in which management of business processes will concentrate on analyzing, designing, implementing, simulating and continuously improving business transactions between network parties.

BPM is a natural complement to SOA [19], and a mechanism through which an organization can apply SOA to high-value business challenges. The objective is to effectively align technical initiatives with the strategic goals of the business user at every level within service networks to achieve a comprehensive approach to real business transformation.

Currently, SOA-centric BPM solutions concentrate on service-enabled processes and cannot explicitly correlate critical business activities and events, QoS requirements, and application (business) data, such as delivery dates, shipment deadlines and pricing, in one process with related activities, events, QoS and business data in other processes in an end-to-end process constellation. This implies that application management information and procedures are deeply buried in service-based application (SBA) code, which severely hinders maintenance and adaptation, both of which are essential for SNs. Such hardwiring means that any change or update to the application management logic already fabricated within an application requires programmatic changes to the SBA itself. This renders impossible the potential reuse, customization, and monitoring of application management capabilities. This also introduces intrinsic discontinuities between end-to-end business processes as information flows may be disrupted. For instance, a possible decoupling of payment information in payment and invoicing business processes from the ordering and delivery information of goods and services in order management and shipment business processes increases risks, could violate data integrity and contractual agreements, and may introduce discrepancies between the various information sources, which underlie these processes. Fixing this problem requires expensive and time-consuming manual reconciliation. The principal activities required to sustain SBAs that collectively enact end-to-end processes, as outlined by [24], include the "collection, management, analysis, and interpretation of the various business activities and data to make more intelligent and effective transaction-related decisions".

SBAs that support end-to-end processes in SNs typically involve well-defined processes such as payment processing, shipping and tracking, determining new product offerings, granting/extending credit, managing market risk and so on. These reflect standard processes or process fragments that apply to a variety of application scenarios. Although such standard processes or process fragments may drive transactional applications between SN partners, they are completely external to current Web service transaction mechanisms and are only expressed as part of application logic. Indeed, there is a need for explicitly managing fine-grained properties of SBAs such as business data, events, operations, process fragments, local and aggregated QoSs and associated Key Performance Indicators (KPIs), and so on, to guarantee a continuous and cohesive information flow, correlation of end-to-end process properties, and termination and accuracy of interacting business processes that is driven by application control (integration) logic.

The above considerations give rise to a multi-modal transaction processing scheme by enabling reliable business transactions that span from front-end SBAs to back-end system-level transaction support and beyond to organizations that are part of a SN. Thus, the need for application-level management technologies that can ensure highly reliable application (not only system)-level transactions and end-user performance via rapid problem diagnosis and resolution — while at the same time support change and capacity planning — is paramount. In particular, we argue in favour of the need to provide rich features and QoS similar to that offered by transaction processing monitors but at the level of application management and application control logic.

Clearly, Business Transaction Management (BTM) is the heart-and-soul of the Business Process Management workpackage in S-Cube, viewing everything from an application perspective. In the world of BTM, an application is considered as a collection of business transactions and events, each triggering actions on the application and corresponding on the infrastructure-level, which is handled by transaction monitors using WS-standards such as WS-Transaction and WS-Coordination. The goal is to track every business transaction in an end-to-end process and correlate it to the information collected from the infrastructure so that, solving problems and planning is done efficiently and holistically. It should be possible to have the ability to "stitch together" the individual business transaction data points into a map of the transaction topology and to monitor the metrics of each transaction for Service Level Agreement (SLA) compliance. Service analytics, e.g., simulation scenarios, and monitoring can then be applied to the transaction data to proactively manage services and accurately pinpoint problems. Such an end-to-end view enables to quickly isolate and troubleshoot the root cause of application bottlenecks, e.g., failure of an order due to the unavailability of just-in time production alternatives, potential performance problems, and tune proactively.

With the above backdrop in mind, there is a need for explicitly introducing fine grained application management techniques that can be applied to SNs ranging from business data, e.g., delivery times, quantities, prices, discounts, events, operations, local and aggregated QoSs and associated KPIs, to business process of transactional nature, e.g., payment and delivery, to guarantee a continuous and cohesive information flow and correlation of end-to-end process properties. This facilitates the potential reuse, customization, of applications expressed in terms of processes and process fragments, as well as the monitoring of applications. The philosophy of this work is that this information and integration logic should be carved out, isolated and made visible to facilitate the design of transactional processes or process fragments that could be used to compose end-to-end processes in SNs (see Figure 2.1).

Loosely speaking, a transactional process fragment is a transactional subprocess that is realized within an end-to-end process, while meeting granular process properties (a list of granular process properties is shown in Figure 2.1). Figure 2.1 shows that SBAs need to cross-correlate granular process properties that span across processes in an end-to-end process constellation. Clearly, granular process properties go far beyond conventional application properties that are considered in traditional system transaction models, and include: operational level agreements (e.g., SLAs), underpinning contracts, policies, rules and QoS thresholds for services at the application-level. Some of these process properties may be designated as being transactional in nature as they can be used to drive a service composition, e.g., end-to-end SLAs.

In this way, granular process criteria can be used to drive and manage the composition of end-to-end processes at the application level. In particular, end-to-end processes exhibit transactional characteristics to deliver on undertakings or commitments that govern their formation and execution at the SBA-level. In other words, an entire end-to-end process or parts of it may fail if some transactional process properties, e.g., non-conformance to SLAs or aggregate mean-value KPIs, are violated. End-to-end processes exhibit transactional characteristics that can be supported by an appropriate transaction infrastructure that employs Web service standards, such as Web services Atomic Transaction [4], Web services-Business Activity [5], Web Services-Coordination [3], and the Business Process Execution Language (BPEL) [17]. Currently, this quartet of Web service standards is used to implement Web service transactions.

The transaction management infrastructure (see bottom layer in Figure 2.1) could for example be based on an open source implementation framework provided by JBoss Transactions (http://www.jboss.org) which supports the latest Web service transactions standards, providing all of the components necessary to build interoperable, reliable, multi-party, Web service-based applications. In such environments there is a clear need for advanced SBAs to coordinate multiple services and processes into a multi-step business transaction.

The suggested approach requires that several service operations or processes attain transactional process properties reflecting SBA semantics, which are to be treated as a single logical (atomic) unit of work that can be performed as part of a business transaction. For example, consider a SN populated by a manufacturer and various suppliers; the manufacturer develops SBAs to automate the order and delivery business functions with its suppliers as part of a business transaction (see Figure 2.1). The transaction between the manufac-

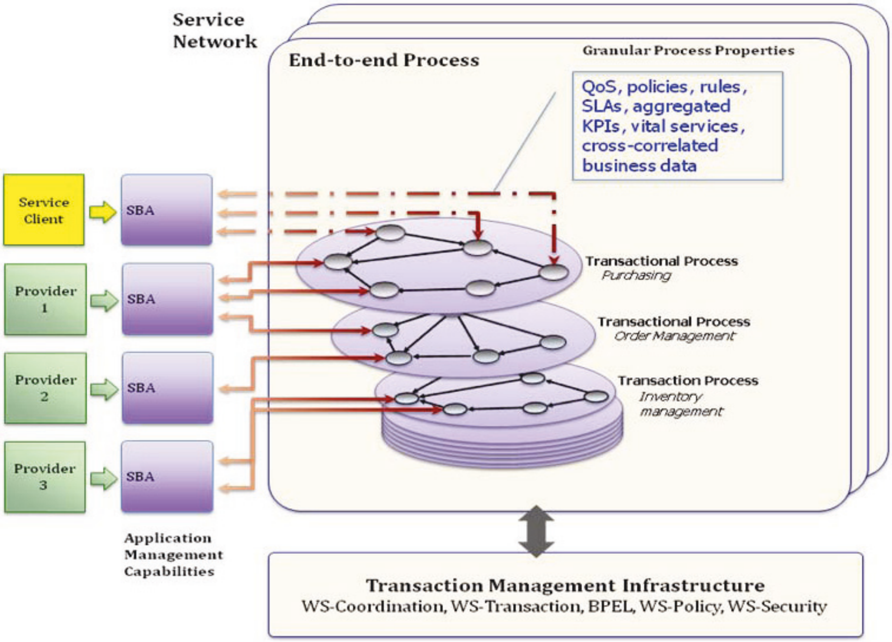

Service Network
End-to-end Process
Granular Process Properties
QoS, policies, rules, SLAs, aggregated KPIs, vital services, cross-correlated business data
Service Client
SBA
Provider 1
SBA
Provider 2
SBA
Provider 3
SBA
Transactional Process
Purchasing
Transactional Process
Order Management
Transaction Process
Inventory management
Application Management Capabilities
Transaction Management Infrastructure
WS-Coordination, WS-Transaction, BPEL, WS-Policy, WS-Security

Fig. 2.1. Business Transactions in Service Networks

turer and its suppliers may only be considered as successful once all products are available in stock, delivered to their final destination, which could take days or even weeks after the placement of the order, and payment has ensued.

Some participating services in a SBA may be vital to a successful outcome of an end-to-end process. For example, a successful order for goods is a prerequisite, i.e., a strong requirement, for the entire end-to-end process to succeed followed by shipping and payment of goods. Given this vital precondition, there are several viable (successful and useful) outcomes. One viable option could be shipping and insurance. Within each viable combination, there may be a further subdivision: competitive (alternative) selection. There might be several prices and types of shipping insurance available from which a selection must be made. These correspond to different instances of insurance services and insurance prices offered by diverse service providers.

The above rationale directs us towards a business-transaction driven service composition and eventual service selection. Such business transactions differ from conventional atomic (database-like) transactions through their ability to interact with a pool of potential composable services and participants (providers) at run-time and to ponder different outcomes, before making a process-specific decision on a subset of those services and associated providers. Such process-specific decisions are based on granular process constructs (e.g., SLA mandates, cross-correlated business operations and data, policies, mean-time KPIs, and so on).

In a SN environment, transactions are complex involving multiple parties, spanning many organisations, and can have a long duration. More specifically, they are automated long-running propositions involving negotiations, commitments, contracts, shipping and logistics, tracking, varied payment instruments, and exception handling.

Business transactions are dynamic in nature. Parts of a business transaction may be retracted, alternatives may be tried out and optional activities may fail without influencing the transaction. Vital activities of a transaction behave as conventional short-lived ACID transactions and need to commit in order for the overall transaction to successfully commit its results. In addition, data — for example, a customer account number or invoice — must be passed between the individual actions that make up the business transaction and some application control logic is required, to “glue” the actions together and make them succeed or fail (individually or collectively) depending on the requirements of the SBA. For example, there must be logic to deal with the conditional invocation of actions and with failures. Performance of all these tasks requires the infusion of advanced and unconventional transactional properties onto the services paradigm.

To achieve some of the above stated objectives, business transactions rely on and extend transactional workflow technology. Like a general workflow, a transactional workflow consists of tasks that satisfy a set of coordination constraints but unlike a general workflow, a transactional workflow emphasizes the transaction aspects that are poorly supported by general workflow, for instance, to guarantee the correctness and reliability of an application in the presence of concurrency and failure [14]. The term transactional workflow is used to emphasize the relevance of the transactional properties for collaborative workflows implementing public processes that transcend functional units within an enterprise or across the organizational boundaries.

As a result of these considerations, this chapter sets out to review and redefine the concept of "business transaction" and their requirements as well as the high-level design principles for a business transaction language or BTL whilst outlining the fundamental properties involved. As such it targets the first research challenge of S-Cube's Global Research Vision on Concepts, Languages and Mechanisms for Agile Service Networks (see Chapter 1). In particular, through integrative efforts, bringing knowledge from various disciplines including SOC, CSCW, BPM and Software Engineering, this book chapter defines how we should design a business transaction, which incorporates the process-level approach with the more conventional applications-level view and sketches the constructs for an initial transactional language.

The book chapter is organised as follows. We will firstly review the essential characteristics of business transactions, leading to our own definition of the concept. Based on the essential characteristics of business transactions we then define a list of requirements for defining a BTL. Subsequently, we will introduce a detailed scenario that serves for further exploring and illustrating the various concepts. Following this we will then introduce the business transaction model, which defines the essential business transaction properties and constructs. The business transaction model is used as a basis to develop and illustrate some initial core BTL elements. Lastly, we summarize the key conclusions of this book chapter and outline directions for future work.

2.2 Essential Characteristics of Business Transactions

As organisations and technology continue to evolve, our understanding of the concept of a "business transaction" also changes. Therefore, it is critical that we formulate a deeper understanding of what a business transaction actually is. Today's organisational structure is heavily influenced by advanced SBAs, which execute well-defined business processes. [21] explains that although service applications execute well-defined business functions, which may drive transactional applications, they are normally external to current Web service (i.e., WS-*) mechanisms. The unprecedented growth in SBAs in a short period of time has emphasized the need to understand the mechanisms and underlying theorize related to the business transaction concept. Understanding the logic of business processes requires us to re-examine what is meant by a business transaction. In this vein, the goal of this section of the book chapter is to help achieve an understanding of what a business transaction is, what do business transactions achieve, and how do they compare with conventional transaction procedures. As research works and technological developments broaden the scope of transaction management, we will align transactional developments with the developments to improve functionality and performance within a SN environment.

2.2.1 Business Transaction Overview

Transactions are mainly associated with the business domain as they represent both tangible and intangible items (goods, money, information, and service). In recent years, the focus within computer science was on the automation of business transactions (i.e., process, execute and coordinate). In addition, the transaction model has undergone some significant changes through the introduction of business and information technological influences.

Nowadays, most business-to-business (B2B) collaborative applications require transactional support, while presenting many difficulties incorporating transactional properties such as Atomicity, Consistency, Isolation, and Durability (ACID). The ACID model is comprised of these four fundamental properties, which are considered the "building blocks" for transaction models. Although extremely reliable, classic ACID transactions are not suitable for loosely coupled environments such as Web service-based systems and transactions, which rely on long running processes. Strict atomicity and isolation is not appropriate to a loosely coupled world of autonomous trading partners, where security and inventory control may foster issues and prevent the 'hard locking' of local databases. The major issue is the isolation of a database transaction. This property requires resource locking that is impractical in the services world. It would also preclude clients from participating in a business process, which is a strong requirement for SBAs. Sometimes, in a loosely-coupled or long-running activity, it may be desirable to cancel a work unit without affecting the remainder. In such cases, the strict atomicity property of transactions needs to be flexible. This is very similar to the property of open nested transactions where the work performed within the scope of a nested transaction is provisional and sub-transaction failure does not affect the enclosing transaction.

The concept of the business transaction is heavily documented throughout various bodies of literature, where many authors share similar meanings and others argue its meaning within various contexts. The increase in organizations adopting a service-networked approach challenges our traditional understandings of the business transaction paradigm. In the following we summarize most of the important works and definitions for reasons of completeness.

The emergence of e-markets has created opportunities for organizations to combine capabilities and configure business transactions to integrate roles and relationships across partnering networks. The relationships of these networks play a fundamental role in the architecture of transactions. [2], captures the essence of the change that business transactions have brought about in recent years due to "the unprecedented reach, connectivity, and low-cost information processing power, open entirely new possibilities for value creation through the structuring of transactions in novel ways".

[13] also makes reference to the 'relationship' factor within a transaction. According to [13], a business relationship is "any distributed state maintained by two or more parties, which is subject to some contractual constraints previously agreed to by those parties". [13], then describes a business transaction as "a consistent change in the state of a business relationship. Each party in a business transaction holds its own application state corresponding to the business relationship with other parties in that transaction".

[11] also adopts an operational view and states that all business transactions should present "significant information processing and communication to reduce uncertainties for buyers and sellers", i.e., quality, commitment, and protocols in plane for resolution over conflicts. Reducing uncertainties within a transaction heavily influences its outcomes.

[9] simply defines a transaction as a "sequence of messages" which suggests that a transaction is triggered through the exchange of messages within a business management system (i.e., the initiator).

[21] reports that a business transaction is defined as "a trading interaction between possibly multiple parties that strives to accomplish an explicitly shared business objective, which extends over a possibly long period of time and which is terminated successfully only upon recognition of the agreed conclusions between the interacting parties". This implies that there is an atomic or 'all or nothing' approach to meet defined objectives, and upon failure the transaction is rolled back.

According to [6], a business transaction is a "set of business information and business signal exchanges amongst two commercial partners that must occur in an agreed format, sequence and time period. If any of the agreements are violated then the transaction is terminated and all business information and business signal exchanges must be discarded". Thus, the document flow structure (time, format, and sequence) which exists between parties is important. [12], states that a business transaction consists of "one or two predefined business document flows and additional business signals". [1] suggests that document flow is important and defines a business transaction as "an atomic unit of work between trading partners. Each business transaction has one requesting (incoming) document and an optional responding (outgoing) document".

However, the flow of these documents often only indicates the pattern in which a transaction relationship exists, for example [12] and [1] do not propose that a transaction provides any business gain. [21], includes the business value factors and states that business transactions are driven by "economic needs and their objective is accomplished only when the agreed upon conclusion among trading parties is reached, e.g., payment in exchange for goods or services".

One of the most important evolutionary factors of the traditional transaction model has been the transition from the single level transaction structure to the multi-level structures. Business processes now interact across and between organizations to create a SN. Therefore, within a typical business transaction there must be at least two parties involved, i.e., a supplier who has a product or service to sell, and a customer who buys this product or service in exchange at a cost. A business model should therefore explicitly describe the collaborative interoperable business processes that are required to fulfill a business transaction.

The formation of a business transaction evolves to encapsulate a more networked and collective effort to reach predefined agreements, practices, procedures and outcomes. To add to this effort, [20] introduces a business transaction model encompassing business principles and models transactions with QoS characteristics, which highlights the need to describe the collaboration aspects' of business processes. Nonetheless, managing the complex transactions is extremely difficult, and these services are managed through the 'negotiation and enforcement of service level agreements' [21].

As these definitions outline a number of key principles, they have been summarized in Table 2.1.

From the above it is evident that the concept of business transactions has adopted several interpretations. It is therefore important to attempt to tie in these meanings to develop a more holistic vision of what constitutes a business transaction.

Within end-to-end processes in a SBA, complex information is exchanged for example, expected service, financial and contractual. [15] draws our attention to the concept of scalability in transaction-based systems, which need to grow to support the relationships within these transactions, especially in the case where organizations are increasing the level of negotiating and interaction in transactions with other organizations to provide some form of business solution. This places a greater emphasis on the choreography of business transitions (i.e., specific business states and the transitions between those states). [12] explains that "the purpose of choreography is to order and sequence business transaction activity and/or collaboration activity within a binary collaboration, or across binary collaborations within a multiparty collaboration". In that respect, a business transaction describes the mission, behavior, action, sequence, correlations of collaborative interactions with an objective of securing a business relationship to request or supply a product or service under predefined conditions.

Having critically analyzed and assessed the previous business transaction definitions, we can now define a business transaction as follows: "A series of collaborative activities that explicitly enforces the achievement of an agreed-upon business objective in end-to-end processes. This objective is subject to service-level agreements that govern the choreographed/orchestrated behavior, non-functional and timing requirements, correlated exchange of information, and control flow of composed services". Business transaction management [16] has evolved from its early roots in business process management to a more comprehensive approach that offers concepts, mechanisms and tools

Table 2.1. Summary of Business Transaction Definitions

Author	Definition	Key Words
Hofman (1994) [9]	"sequence of messages"	Message, sequence
Kambil (1997) [11]	"significant information processing and communication to reduce uncertainties for buyers and sellers"	Information processing, reduce uncertainties
Clark (2001) [6]	"set of business information and business signal exchanges amongst two commercial partners that must occur in an agreed format, sequence and time period. If any of the agreements are violated then the transaction is terminated and all business information and business signal exchanges must be discarded"	Information and business signals, exchange, format, sequence, violation, termination
Aissi (2002) [1]	"an atomic unit of work between trading partners. Each business transaction has one requesting (incoming) document and an optional responding (outgoing) document"	Atomic unit, request, respond
Kim (2002) [12]	"one or two predefined business document flows and additional business signals"	Predefines flows
Kratz (2004) [13]	"a consistent change in the state of a business relationship. Each party in a business transaction holds its own application state corresponding to the business relationship with other parties in that transaction"	Distributed relationship, change, transaction state
Papazoglou (2006) [21]	"a trading interaction between possibly multiple parties that strives to accomplish an explicitly shared business objective, which extends over a possibly long period of time and which is terminated successfully only upon recognition of the agreed conclusions between the interacting parties"	Interaction, parties, accomplish, objectives, long periods, negotiation, conclusion

to manage the lifecycle of a business transaction starting from business goals over transactions definition, through deployment, execution, measurement, analysis, change, and redeployment.

A shared business objective extends over a possibly long period of time and is terminated successfully only upon recognition of the agreed conclusions, e.g., stipulated QoS, compliance to business and regulations, etc, between the interacting parties. A transaction usually outlines the liabilities of each party in the event where the intended actions are not carried out (e.g., promised services not rendered, services rendered but payment not issued). If a business transaction completes successfully then each participant will have made consistent state changes, which, in aggregate, reflect the desired outcome of the multi-party business interaction.

2.3 Requirements of a Business Transaction Language

Business Transaction Management (BTM) views everything from an application perspective. In the world of business transaction management, an application is considered as a collection of business transactions and events, each triggering actions on the application and corresponding on the infrastructure-level, which is handled by transaction monitors using WS-standards such as WS-Transaction and WS-Coordination. The goal is to track every business transaction in an end-to-end process and correlate to the information collected from the infrastructure so that, solving problems and planning is done efficiently and holistically. It should be possible to have the ability to "stitch together" the individual business transaction data points into a map of the transaction topology and to monitor the metrics of each transaction for SLA compliance.

A BTL plays a pivotal role in BTM. The core requirement for a BTL is the ability to describe the granular transactional process properties of end-to-end processes, such as business commitments, mutual obligations and agreed upon KPIs, in a standard form that can be consumed by tools for business transaction implementation and monitoring. In this section, we will refine this all-encompassing BTL requirement.

Table 2.2 summarizes the behavioral characteristics and key factors, which differ within a transaction and need to be expressed as part of a BTL.

As summarised in Table 2.2 above, there are several characteristics and key factors, which distinguish certain stages within a business transaction. These stages, including transaction activities, business service dimensions, and their implications for SNs, are captured in Table 2.3.

As Table 2.3 (partly based on Open Electronic Data Interchange, or Open EDI [10]) shows, in order to deliver a business transaction, there are business processes, transaction activities, dimensions, and behaviour which need to be agreed upon, at various phases upon entering a transaction agreement. The last column (far right) summarises the key tasks required at each stage

Table 2.2. Behavioural Characteristics and Key Factors of Transactions (Adapted from [20]

Characteristics	Key Factors
Generic	Who is involved What is being transacted Destination of payment and delivery Transaction time frame Permissible operations
Distinguishing	Links to other transactions Receipt and acknowledgment
Advanced	Ability to support reversibility (compensatible) and repaired (contingency) transactions Ability to reconcile transactions with other transactions Ability to specify contractual agreements, liabilities, and dispute solution policies

within a transaction occurring in a SN. While the information gathering and negotiation stages fall outside the scope of this chapter, we will concentrate in this book chapter on language requirements emerging from stages 3 and 4.

Based on an extensive literature survey and associated comparative analysis of existing business transaction models — notably Open EDI, the UN/CEFACT Modelling Methodology (UMM) [23] and electronic business using eXtensible Markup Language (ebXML) [6] — in connection with the S-Cube reference architecture, we have distilled the following key transaction language requirements for stages 3 and 4:

Requirement 1: Expressing Collaborative Activities That Explicitly Enforce the Achievement of an Agreed-Upon Business Objective in End-to-End Processes

The BTL is pervasive in that it will be defined over end-to-end processes involving choreographed and/or orchestrated services that exhibit transactional properties. Transactional properties may be expressed by combining existing transactional services or process fragments and associating them with application-level characteristics as well as contractual agreements in order to develop SBAs for SNs.

Requirement 2: Expressing an On-Demand Delivery Model for SBAs

Business transactions are required to furnish an "on-demand" delivery model in which end-users may specify their preferences, e.g., desirable QoS, mandatory regulations, etc, as regards an end-to-end process. This implies that services are tentatively (re-) selected from a pool of service providers on the fly. Services and transactional process fragments can then be tailored, composed and then deployed over a variety of platforms.

Table 2.3. Business Transaction Stages & Dimensions [25]

Business Transaction	Transaction Activities	Main Business Service Dimensions		Applications in Service networks
Stage 1: Information Gathering	Source and product identification & potential supplier listed	Marketing Service	Information Service	Product information, response to inquiries
Stage 2: Negotiation	Evaluate supplier & analyze supplier quote		Communications Service	
Stage 3: Contract Fulfilment	Order, Payment, Delivery, Operations management	Logistics Service	Order and Payment Service	Order process, invoice, lead-time transportation, tracking shipment, advance ship notice, inventory transparency, response to inventory variety
			Delivery Service	
			Inventory Service	
		Operations Service	Product Service	Product quality, feature offering, cost, production scheduling, cycle time, capacity
			Production Service	
Stage 4: Collaboration	Collaboration planning, collaboration product development, co-location	Collaboration Service	Product Collaboration	Planning coordinator, geographical location, alternate delivery channel
			Information Collaboration	Updated information, joined planning and forecasting

Requirement 3: Facilitating Reusability and Extensibility

The BTL will impart constructs that define reusable and extensible transactional process fragments.

Requirement 4: Expressing Conventional Atomic Actions

The transaction language needs to cater for conventional atomicity as, in some circumstances, service operations or transactional process fragments in an end-to-end process have to be strictly atomic. Assuming, for instance, that a client application decides to invoke one or more operations from a particular process fragment such as order confirmation, or inventory check, it is highly likely for the client application to expect these operations to succeed or fail as a unit. We can thus view the set of operations used by the client in each process fragment as constituting an atomic unit of work (viz. atomic action).

Requirement 5: Expressing Application-Level Atomicity Criteria

In addition to the previous requirement, the language should be able to express and associate application-level atomicity (described in Section 2.5.1) criteria. For instance, we may be able to express that a transaction is a payment-aware. This means that if payment is not made within a pre-specified period then the transaction fails. Similarly, transactions could be made QoS, or SLA-aware and succeed or fail depending whether QoS criteria or SLA terms are met.

Requirement 6: Expressing Long Duration Nested-Activities

Long-duration (business) activities could be expressed as aggregations of several atomic actions and may exhibit the characteristics and behaviour of open nested transactions and transactional workflows. The atomic actions forming a particular long-duration business activity do not necessarily need to have a common outcome. Under application control (business logic), some of these may be performed (confirmed), while others may fail or raise exceptions such as time-outs or failure. To exemplify a long-duration business activity, consider a slight variation of the order processing scenario where a manufacturer asks one of its suppliers to provide it with valuable and fragile piece of equipment.

Now, consider that one atomic action arranges for the purchase of this product, while a second arranges for its insurance, and a third one for its transportation. If the client application is not risk-averse (due to excessive costs), then even if the insurance operation (atomic action) votes to cancel, the client might still confirm the transaction and get the item shipped uninsured. Most likely, however, the client application would probably retry to obtain insurance for the item. Once the client discovers a new insurer, it can try again to complete the long-duration business activity with all the necessary atomic actions voting to confirm on the basis of the particular coordination protocol used.

Requirement 7: Expressing and Enforcing Policies

This helps service networks achieve the global control of end-to-end processes by enforcing policy consistently across the runtime environment, without requiring applications to be recoded and deployed. This involves constructs to define policies and SLA thresholds based for example on transaction averages. It also involves constructs to define and enforce policies.

Requirement 8: Expressing and Enforcing QoS and Compliance Criteria

Activities in business transactions will be able to express compliance with regulations, SLA terms and QoS characteristics in an end-to-end fashion. The BTL will therefore be equipped with constructs to define controls and counter measures.

Requirement 10: Incident Management

The language needs to be endowed with constructs to feed the transaction management infrastructure with information on business transaction events and SLA violations (in addition to current component based events) that is useful for repairing a process anomaly or problem, reducing the mean time to repair. This includes constructs to define stalled transactions, missing steps, faults, and application exceptions, as well as other issues such as incorrect data values, boundary conditions, and so on.

Requirement 11: Business Transaction Monitoring

Traditional transaction monitors are able to monitor only system-related activities and performance. The capability of the BTM system (that is built around the BTL) is the ability to compute and monitor KPIs using rules that trigger automated alerts or process actions when they move outside their target range. Process owners can then respond instantly to events that affect the bottom line. This thus implies that the BTL will contain constructs and operators that will define exactly how processes and services will be monitored (e.g., through logging or actively checking), and how process-level KPIs will be mapped down to SBA-level SLAs and QoS.

Requirement 12: End-to-End Visibility and Optimization

The purpose is to provide visibility into the service interactions within the scope (i.e., context) of the business transaction and make process performance visible to process owners and also to provide a platform for problem escalation and remediation. BTM system should provide the ability to measure performance across organizational and system boundaries and detect exceptions in service interactions. As processes run, the BTM system should be able to continuously capture the snapshots of process instance data and aggregate them in a meaningful manner.

The above requirements above are essential in determining the characteristics and functionality of the BTL, which is presented in Section 2.6 of this chapter.

2.4 Illustrating Scenario

The following scenario deals with an integrated logistics process involving a customer, suppliers and a logistics service provider. This logistics model consists of forecast notification, forecast acceptance, inventory reporting, shipment receipt, request and fulfil demand, consumption and invoice notification activities. In particular, this scenario is part of the automotive supply chain case study proposed in the vision chapter, illustrating the use of simple business transactions and associated event monitoring.

Figure 2.2 depicts the flow of information between the interacting nodes (business partners) in a very simple SN involving three parties (car manufacturers, part suppliers and logistics providers). Service interactions are governed by a simplified SLA. This figure shows the sequential ordering of the interaction events between the business partners in terms of message exchanges. The message "Notify of Forecast", contains car part demand information (planning demand), is sent by a forecast owner (the car manufacturer) to a forecast recipient (the supplier).

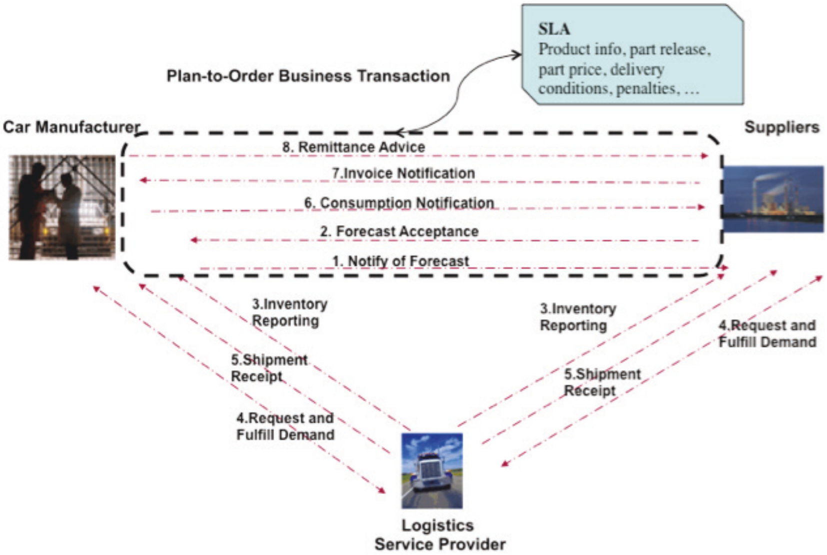

SLA
Product info, part release, part price, delivery conditions, penalties, ...
Plan-to-Order Business Transaction
Car Manufacturer
Suppliers
8. Remittance Advice
7.Invoice Notification
6. Consumption Notification
2. Forecast Acceptance
1. Notify of Forecast
3.Inventory Reporting
5.Shipment Receipt
4.Request and Fulfill Demand
3.Inventory Reporting
5.Shipment Receipt
4.Request and Fulfill Demand
Logistics Service Provider

Fig. 2.2. The Integrated Logistics Scenario

The message "Forecast Acceptance" sent back from the supplier acknowledges that the demand forecast has been accepted. Next, the message "Distribute Inventory Report" is performed by an inventory information provider to report the status of the inventory to an inventory user.

The inventory report can include any car parts held in inventory. The message "Advance Shipment Notification" allows a shipper to notify a receiver that a shipment has been assigned. This notification is often a part of the shipment process. Message "Shipment Receipt" is performed by a consignee to report the status of a received shipment to another interested party, such as another consignee, a transport service provider, a third-party logistics firm, or a shipper.

Receipt of a shipment is reported after it has been delivered by a carrier and inspected by receiving personnel. The customer then issues an "Invoice

Notification" to communicate car parts consumption to the supplier, allowing the supplier to trigger invoicing for the consumed material. The message "Invoice Notification" enables a provider to invoice another party, such as a buyer, for goods or services performed.

Finally, the message "Notify of Remittance Advice" enables a payer to send remittance advice to a payee (in this case the supplier), which indicates which payables are scheduled for payment.

Message exchanges are bundled together in three separate message exchanges (A, B and C). As shown in Figure 2.3, the message exchanges 1,2,6,7 and 8 (named A) can be bundled together in the form of a business-aware transaction, which is governed by a simplified SLA between the two trading partners, i.e., the car manufacturer and the supplier.

As a whole, our global business transaction scenario consists of three binary business transactions, i.e., transaction A between the car manufacturer and the supplier, B between the car manufacturer and the logistics service provider, and C between the logistics service provider and the supplier. The SLA for each transaction contains a set of policy constraints, such as temporal and spatial constraints, penalties, some business regulatory rules, etc, that must be fulfilled during the transaction execution. It also prescribes the recovery strategies for the business transaction in case any of the message exchanges fail, for instance, either this business transaction must be compensated by some means, e.g., issuing another forecast or invoice, or the entire transaction fails. Besides, the SLA also drives the business-aware transaction with other agreements on the KPIs of the transaction, which enables the performance monitoring and measurement of the execution. To sum up, a business transaction between two trading partners is driven by the conditions specified in the agreed-upon SLA.

Some sample conditions in the SLA that should be monitored and enforced by the transaction management system during the transaction execution are:

- Has the supplier acknowledged the order?
- Has the supplier and logistics service provider committed to a ship date?
- Will the supplier start manufacturing on-time?
- Will the order be shipped on-time?
- Does this order meet our on-time delivery goals and other KPIs?
- If the order shipped by the logistics provider does not arrive on-time, how should we proceed?
- Does it affect other partners if the logistics service provider cannot deliver the order? How to compensate this problem?

A process activity can be vital or non-vital for its business transaction. If the vital one fails, its transaction will fail, e.g., the activity "Prepare Invoice" in transaction A is vital for its transaction, since without the invoice, the payment between the trading partners in A cannot be conducted. Process activities in the same or different business transactions can have weakly or strongly consistent atomicity relationships. "Weakly" means that either both the two activities succeed or if one of them fails and is compensated/recovered then the other one will also fail or be compensated/recovered in the near future. “Strongly consistent-atomic” enhances the “weakly consistent-atomic” relationship by requiring that if both activities fail, they must result into a consistent (predefined) state (see Section 2.5.1 which explains formalization of consistent-atomicity). For instance, the activities “Send Invoice” in transaction B and “Receive Invoice” in transaction C are strongly consistent-atomic.

Traditional transaction monitoring mechanisms are able to monitor only system-related activities and performance. However, it is important to understand that business-aware transactions correlate application-level events and situations with the supporting infrastructure. For example if the manufacturer requests a change in the order, can we accept the change in connection with agreed-upon KPIs? Alternatively, if an infrastructure-level change has been made, we can assess its impact on the application (SBA) level. More importantly, we can deduce if the processes are still working to plan, if there are any bottlenecks and where they appear, if there was a process improvement or worsening, and so on. Correlating lower level activities, e.g., from the service composition or the infrastructure level, with higher level business events in the form of transactions, provides opportunities to continuously monitor and measure the lifecycle of a transaction, while providing data and events to trigger and populate controls, as well as time-based data for on-time measurement.

2.5 Business Transaction Model

From the definition of a business transaction, it has been shown that it is a mission-critical task that spans across the organizational boundaries and encompasses different types of business concepts to achieve business awareness, such as SLA awareness. To provide a more in-depth understanding of a business transaction, this section first scrutinizes the business concepts that significantly pertain to a business transaction and proposes a business transaction model on the basis of which the transaction language described in Section 2.6 can be developed. Section 2.5.1 describes high-level business transaction concepts concentrating on rationalizing their indispensability in a BTL. Then, an advanced model of business transaction is presented is Section 2.5.2 that captures these business transaction concepts either explicitly or implicitly.

2.5.1 High-Level Concepts

A collaborative business environment involves multiple partners (or organizations) that foster the indispensability of mutual obligations. The mutual obligations can be presented by an SLA that explicitly defines the common requirements and policies that have to be satisfied by the committed trading partners. Business transaction is deemed as one of the main drivers of an end-to-end process that involves multiple trading partners. Hence, any transaction mechanism that is used to enforce the business transaction in an end-to-end process needs to be cognizant of SLA concepts. These SLA concepts are described in this section including application-level atomicity that governs the transaction process. The concept of Business Collaboration is also discussed to present a brief overview of business transaction in a collaborative business environment.

Business Collaborations and Business Transactions

One key requirement for enabling cross-enterprise business process automation is the ability to describe the collaboration aspects of the business processes, such as commitments and exchanges of monetary resources in a standard form that can be used by applications and consumed by tools for business process implementation and monitoring [18]. Business collaboration captures the information and message exchange requirements between trading partners. Additionally, trading partners may have multiple interactions in an end-to-end process. The sequence of these interactions is captured by a business protocol. A business protocol identifies and captures all behavioural aspects that have cross enterprise business significance [21]. Behavioural aspects may include the messaging behaviours between trading partners, which help the participants understand and plan their interactions that conform to the agreed-upon business protocol. We have incorporated a set of important business artefacts (e.g., business protocol) in our business transaction model.

Application-Level Atomicity

The governing factors (e.g., SLAs) of a business transaction foster the atomic behaviour. This type of atomicity is called application-level atomicity or non-conventional atomicity [18] and consists of a set of application related atomicity criteria. Each criterion is treated as a single individual logical unit of work that determines a set of vital or viable outcomes for a business transaction. The outcomes of a business transaction may involve non-critical partial failure, or selection among contending service offerings rather than the conventional strict atomicity (all or nothing).

Application-level atomicity can be deemed as the criteria for checking consistency and correctness of a business transaction against predefined standard operations. According to [18], there may be different types of application-level atomicity that are discussed in the following:

- Contractual Atomicity: Business transactions are generally governed by contracts and update accounts. Contractual atomicity may include messaging sequence (or interactions), QoS parameters (e.g., time to perform), and security parameters (e.g., non-repudiation). Electronic contracts define both the legal terms and conditions and technical specification that a trading partner must implement to put an electronic trading relationship into effect. As an example, if a contract enforces an obligation to acknowledge a purchase order within specified time frame, seller has to send an acknowledgement to buyer. Otherwise, the transaction should be aborted and failed. Consequently, the contract will be null and void. A business transaction is completed successfully only when the contractual provisions are satisfied.

- Operational Level Atomicity: Business transactions usually involve the actual delivery of purchased items (tangible and non-tangible) [18]. This type of atomicity has been well defined by [22] and refined by [18]. The operational atomicity is decomposed into Payment Atomicity, Goods Atomicity, and Delivery Atomicity. This research does not include Goods Atomicity but emphasizes on payment atomicity and delivery atomicity since Goods Atomicity can be realized by Delivery Atomicity.
  - Payment Atomicity: Payment atomic protocol affects the monetary transaction from one party to another. It is the basic level of atomicity that each operation level business transaction should satisfy. This type of atomicity has greater influence on the entire transaction process because if the payment process fails, all the cohorts must have to be failed. Notably, a payment atomic transaction can be contract atomic.
  - Delivery Atomicity: Delivery is typically the last phase of an end-to-end process chain. The purpose of this type of atomicity is to ensure that the right goods are delivered to the buyer. The delivery atomic protocol also guarantees the quality of the specified products is maintained and they are delivered within the specified. A business transaction cannot be completed successfully unless a Delivered notification arrives from the buyer, since the failure of delivery may cause failure of all the (sub-) transactions that participated in the transaction process.

Our business transaction model adapts the application-level atomic criteria to achieve a consistent and business-aware transaction mechanism since application-level atomicity contains higher business significance involving contract, constraints, and also the operations. The association of business transaction with these business aspects is shown in the business transaction model in the next section. Noticeably, these atomicity (contract and operational level atomicity) are not shown explicitly in the business transaction model instead they are represented by contractual primitive and operational primitive.

2.5.2 Business Transaction Model Overview

An important requirement in making end-to-end process automation happen is the ability to describe the collaboration aspects of the processes, such as commitments and mutual obligations, in a standard form that can be consumed by tools for business process implementation and monitoring. This gives rise to the concept of a business transaction model that provides a comprehensive set of concepts and several standard primitives and conventions that can be utilized to develop complex Service Based Applications (SBAs) (see Section 2.1). Central to the business transaction model is the notion of a business transaction (see Section 2.2.1 for the definition). Business transactions cover many domains of activity that businesses engage in, such as request for quote, supply chain execution, purchasing, manufacturing, and so on. The purpose of a business transaction is to facilitate specifying common (and standard) business activities and operations that allow expressing business operational semantics and associated message exchanges as well as the rules that govern them. The combination of all these primitives enforces SN partners to achieve a common semantic understanding of the business transaction and the implications of all messages exchanged.

The business transaction is initiated by a service client and brings about a consistent change in the state of a relationship between two or more network parties. A business relationship is any distributed state held by the parties, which is subject to contractual constraints agreed by those parties.

There are four key components in a business transaction model that help differentiate it from (general) message exchange patterns. These are:

- Commitment exchange.
- The party (or parties) that has the ability to make commitments.
- Business constraints and invariants that apply to the message exchanged between the interacting parties.
- Business objects (documents) that are operated upon by business activities (transactional operations) or by processes.

We have developed a meta-model in UML that captures the key constructs of the business transaction model and their interrelationships (see Figure 2.3).

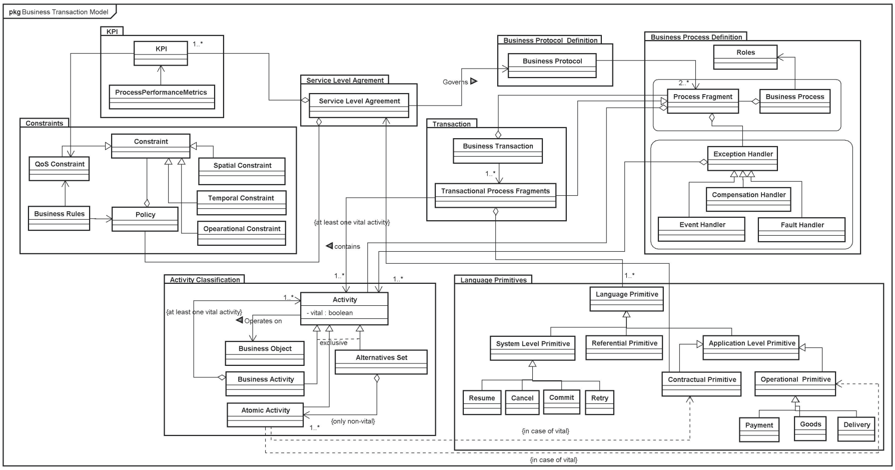


Fig. 2.3. The Business Transaction Model

Business transactions make up the core of the transaction model, and may incorporate a blend of transactional and non-transactional process fragments that are coordinated through business protocols and collectively make up an end-to-end process.

Transactional process fragments are characterized by universally acceptable system level primitives — such as resume, cancel, commit and retry. There are also referential primitives that correlate an activity with other activities using control or data flow, e.g., payment refers to an associated order. Application level primitives comprise of contractual primitives and operational primitives. Contractual primitives define mutual agreements between network parties relying on constructs such as authorizations, obligations and violations. Lastly, operational primitives help to enforce network partner commitments. They introduce a mandatory set of three operational level atomicity criteria that reflect the operational semantics of three standard business activities (ordering, payment, and delivery). For instance, payment atomicity affects the transfer of funds from one party to another in the transaction. This means that the transaction would fail if payment were not made within a pre-specified time period that was agreed between a supplier and a customer. Delivery atomicity, on the other hand, implies that the right goods will be delivered to a customer at the time that has been agreed.

Transactional process fragments embody one or more activities that fall apart in the following two mutually exclusive activity types: vital or non-vital. Atomic activities are short-lived and atomic actions which in some cases may be part of long-running business transactions, referred to as Business Activities (adopting the WS-Transaction terminology). The alternatives set is a group of alternative atomic activities and can be vital or non-vital. Alternatives set can contain only non-vital atomic activities since it is not pre-defined which alternative is executed at run-time.

Business activities usually operate on business (document-based) objects. These are traditionally associated with items such as purchase orders, catalogues (documents that describe products and service content to purchasing organizations), inventory reports, shipping notices, bids and proposals. Such objects may also be associated with agreements, contracts or bids. This allows business transactions to interchange everything from product information and pricing proposals to financial and legal statements.

Activities could also belong to the Exception Handler, e.g., Compensation Handler, Event Handler, and Fault Handler. They are special kinds of activities that are performed in case of particular activity fails or repair the transactions after they were disrupted.

An SLA is defined as a coherent set of explicitly stated policies that prescribe, limits, or specifies any aspect of a business protocol that forms part of the commitment(s) mutually agreed to among the interacting parties, and stipulates agreed upon QoS constraints (some of which stem from business rules). An SLA also outlines what each party can do in the event the intended actions are not carried out (e.g., promised services not rendered, services rendered but payment not issued).

Policies may encompass one or more constraints. Spatial constraints regulate the access to protocols based on criteria such as spatial point and route. Temporal constraints stipulate timing restrictions such as time, date, duration and interval. The business transaction model also furnishes some standard operational constraints for governing message exchanges. The model could easily express sequencing semantics, which for instance require that "ordering" occurs first and is followed by "transport" and "delivery". "Payment" can happen either before or after the "delivery" function.

2.6 Initial Design of Business Transaction Language (BTL)

After describing the business transaction model we can now develop the BTL to transform our transaction model into language constructs. This section presents some initial and elementary constructs that we have developed and captured in a transaction language named as BTL. The BTL is a declarative transaction language and its constructs are XML-based representing the transactional elements specified at design time. BTL is also planned to facilitate annotating the granular process properties for each of the activities in the process fragments that compose end-to-end processes. Noticeably, a detailed description of the run-time environment as well as run-time transformation between the BTL and equivalent executable constructs is not within the scope of this chapter. However, the mapping will most likely be from BTL to an execution language, e.g., an extended transactional BPEL. Hence, the purpose of this section is confined only to our elementary work on the BTL constructs.

The business transaction model in the previous section consists of various domains (e.g., SLA, Constraints, Primitives, etc) that are related to business transactions, their elements and associations between the elements as well as domains. In this section, we codify the model considering all these elements and relations.

This XML snippet shown in Listing 2.1 is an example of using the business transaction language constructs defined in Section 2.5.2 to describe part of the Integrated Logistics Scenario in Figure 2.2. In particular, we focus on the process fragment "Payment Processing" at the car manufacturer, which is in fact a transactional process fragment. First, the fragment uses a business protocol, which prescribes the sequence order of activities and interactions performed by the car manufacturer that must be visible for other participants. An activity can be vital or non-vital. For instance, "Process Payment" is considered to be vital since it actually executes the payment, while "Receive Invoice" and "Send Remittance Advice" are non-vital since it may be no problem at all for the supplier to resend an invoice a few times more and to check the payment without the remittance advice.

Listing 2.1. Sample code snippet of Business Transaction Language

```
<BTL>
<TransactionalProcessFragment name='Payment␣Processing'>
  <BusinessProtocol>
    <Sequence>
      <Activity vital='false'>Receive Invoice</Activity>
      <Activity vital='true'>Process Payment</Activity>
      <Activity vital='false'>Send Remittance Advice</Activity>
    </Sequence>
  </BusinessProtocol>
  <LanguagePrimitives>
    <applicationLevelPrimitive>
      <operationalPrimitive>
        <Activity vital='true'>Process Payment</Activity>
        <extend>Payment</extend>
      </operationalPrimitive>
      <ContractualPrimitive>
        <SLA>
          <Policy>
            <timeToComplete>1 month</timeToComplete>
            <creditMaximumAllowance>
                  \$1000
            </creditMaximumAllowance>
          </Policy>
        </SLA>
      </ContractualPrimitive>
    </applicationLevelPrimitive>
  </LanguagePrimitives>
</TransactionalProcessFragment>
</BTL>
```

Second, the transactional process fragment may be driven by many language primitives, which are in this case the operational primitives and contractual primitives that are the subsets of application-level primitive.

The operational primitive indicates that in order to realize the vital activity “Process Payment”, we will extend the existing predefined operational primitive “payment”. The contractual primitive prescribes the agreed-upon SLA between the participants, which comprises of a number of constraints embedded in a policy definition.

In this case there are two policy constraints, one prescribes that the completion time for this fragment must be within a month, and the other one allows the payment using credit card only up to $1000.

2.7 Summary and Outlook

The discipline of Business Process Management emerged as both a management principle and a suite of software technologies focusing on bridging diverse systems, organizations, people, and business processes. Its objective is to manage the lifecycle of a process starting from business goals over process definition, through deployment, execution, measurement, analysis, change and redeployment.

In the context of S-Cube's research vision, and particularly the ongoing research efforts on the first research challenge dealing with languages and mechanisms for Agile Service Networks (see Section 1.2.5 of Chapter 1), this chapter has considered the management of business processes that are increasingly complex and integrated both within internal corporate business functions (e.g., manufacturing, design engineering, sales and marketing, and enterprise services) and across end-to-end processes. Particularly, the trend will be to move from relatively stable, organization-specific applications to more dynamic, high-value ones where business process interactions and trends are examined closely to understand more accurately application needs and dynamics. In such environments there is a clear need for advanced business process management concepts and mechanisms to coordinate multiple services into a multi-step business transaction. This requires that several service operations or processes attain transactional properties reflecting business semantics, which are to be treated as a single logical (atomic) unit of work that can be performed as part of a business transaction.

The recent advent of Web service technologies and open standards such as WSDL, BPEL, and WS-Policy has helped to evolve our thinking about how distributed applications can connect and work together. However, none of these core Web service specifications were designed to provide mechanisms by themselves for describing how individual services can be connected to create dependable business critical solutions with the appropriate level of complexity that can guarantee absolute completion and accuracy of interacting business processes.

Indeed, there is a need for explicitly managing fine grained tenets of SBAs such as business data, events, operations, process fragments, local and aggregated QoS and associated KPIs, and so on, to guarantee a continuous and cohesive information flow, correlation of end-to-end process properties, termination and accuracy of interacting business processes that is driven by application control(-integration) logic.

The above considerations give rise to a multi-modal transaction processing scheme by enabling reliable business transactions that span from front-end SBAs to back-end system-level transaction support and beyond to organizations that are part of a SN. Thus, the need for application-level management technologies that can ensure highly reliable application (not only system)-level transactions and end-user performance via rapid problem diagnosis and resolution — while at the same time support change and capacity planning — is paramount. In particular, we argue in favour of the need to provide rich features and QoS similar to that offered by transaction processing monitors but at the level of application management and application control logic, giving rise to the concept of a business transaction.

The business transaction then becomes the framework for expressing detailed operational business semantics. Conventional approaches to business transactions, such as Open EDI, UMM and ebXML, focus only on the documents exchanged between partners, rather than coupling their application interfaces, which inevitably differ.

This book chapter targeted the concept of a business transaction and explored how process fragments, and particularly transactional process fragments, fit in the context of a running scenario that possesses transaction properties. Conventional (ACID) and unconventional (application-based) types of atomicity were introduced, including contract and delivery atomicity, in the frame of a business transaction model. The transaction model provides a comprehensive set of concepts and several standard primitives and conventions that can be utilized to develop complex SBAs involving transactional process fragments.

The main goal of this chapter was to survey the field of business process management, concentrating on business-aware transactions, and introducing an initial version of a business transaction language (BTL) to represent and develop them. In particular, the initial BTL specification presented in this book chapter defines a transaction model and mechanisms for transactional interoperability between end-to-end service constellations in SNs and provides a means to define and enforce transactional QoSs within SBAs. Both the model and language are firmly grounded on a requirements analysis, involving an in-depth literature survey while taking into account requirements stemming from other work packages in the S-Cube research framework. The approach taken mimics business operational semantics and does not depend upon underlying technical protocols and implementations. Mission-critical composite applications will differ from the smaller-scale composite applications built using BPEL extensions. The mission-critical composite applications will be built on the established foundation of service networks, enterprise systems, Web service standards and platforms.

References

1. Aissi, S., Malu, P., Srinivasan, K.: E-Business Process Modelling: the next big step. IEEE Computer 35(5), 55–62 (2002)
2. Amit, R., Zott, C.: Value Creation in eBusiness. Strategic Management Journal 22, 493–520 (2001)
3. Cabrera, L.F., Copeland, G., Feingold, M., Freund, R.W., Freund, T., Johnson, J., Joyce, S., Kaler, C., Klein, J., Langworthy, D., Little, M., Nadalin, A., Newcomer, E., Orchard, D., Robinson, I., Shewchuk, J., Storey, T.: Web Services Atomic-Transaction Specification, Version 1.0. IBM Web Service Transactions Specifications (2005)
4. Cabrera, L.F., Copeland, G., Feingold, M., Freund, R.W., Freund, T., Johnson, J., Joyce, S., Kaler, C., Klein, J., Langworthy, D., Little, M., Nadalin, A., Newcomer, E., Orchard, D., Robinson, I., Storey, T., Thatte, S.: Web Services Atomic-Transaction Specification. IBM Web Service Transactions Specifications (2005)

5. Cabrera, L.F., Copeland, G., Feingold, M., Freund, R.W., Freund, T., Joyce, S., Klein, J., Langworthy, D., Little, M., Leymann, F., Newcomer, E., Orchard, D., Robinson, I., Storey, T., Thatte, S.: Web Services Business Activity Framework (WS-BusinessActivity). IBM Web Service Transactions Specifications (2005)
6. Clark, J., Casanave, C., Kanaskie, K., Harvey, B., Smith, N., Yunker, J., Riemer, K.: ebXML Business Process Specification Schema, Version 1.01. UN/CEFACT and OASIS Specification (2001)
7. Ferguson, D.F., Stockton, M.L.: Enterprise Business Process Management – Architecture, Technology and Standards. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 1–15. Springer, Heidelberg (2006)
8. Harmon, P.: Business Process Change. Morgan Kaufmann, San Francisco (2007)
9. Hofman, W.J.A.: Conceptual Model of a Business Transaction Management System. PhD thesis, Uitgeverij Tutein Nolthenius (1994)
10. International Organization for Standardisation (ISO). ISO/IEC JTC1/SC30: The Open-EDI Reference Model. Committee Draft 14662 (1995)
11. Kambil, A., van Heck, E.: Reengineering the Dutch Flower Auctions: A Framework for Analyzing Exchange Organizations. Information Systems Research 9(1), 1–19 (1997)
12. Kim, H.: Conceptual Modelling and Specification Generation for B2B Business Process based on ebXML. ACM SIGMOD Record Archive 31(1), 37–42 (2002)
13. Kratz, B.: Protocols for Long Running Business Transactions. Technical Report 17, Infolab, Tilburg University (2004)
14. Liu, C., Kuo, D., Lawley, M., Orlowska, M.E.: Gehrmann Laboratories. Modeling and Scheduling of Transactional Workflow (1996)
15. Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A., Elmagarmid, A.: Business-to-Business Interactions: Issues and Enabling Technologies. VLDB 12(1), 59–85 (2003)
16. Moeller, M., Ceylan, S., Bhuiyan, M., Graziani, V., Henley, S., Veress, Z.: End-to-End e-business Transaction Management Made Easy. IBM Redbooks (2004)
17. OASIS Web Services Business Process Execution Language (WSBPEL) TC. Business Process Execution Language Specification, Version 2.0. Oasis specification, OASIS (April 2007)
18. Papazoglou, M.P.: Web Services and Business Transactions. World Wide Web: Internet and Web Information Systems 6(1), 49–91 (2003)
19. Papazoglou, M.P.: Web Services: Principles and Technology. Prentice-Hall, Englewood Cliffs (2007)
20. Papazoglou, M.P., Georgeakopoulos, D.: Service Oriented Computing. Communications of the ACM 46(10), 25–28 (2003)
21. Papazoglou, M.P., Kratz, B.: A Business-Aware Web Services Transaction Model. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 352–364. Springer, Heidelberg (2006)
22. Tygar, J.D.: Atomicity in Electronic Commerce. In: Proceedings of the 15th annual ACM symposium on Principles of distributed computing, pp. 8–26 (1996)
23. UN/CEFACT. UMM Meta Model – Foundation Module Candidate for 2.0. Part of UN/CEFACT's Modeling Methodology, UMM (2009)
24. Yang, J., Papazoglou, M.P.: Interoperation Support for Electronic Business. Communications of the ACM 43(6), 39–47 (2000)
25. Yang, Y., Humphreys, P., McIvor, R.: Business Service Quality in an E-commerce Environment. Supply Chain Management: An International Journal 11(3), 195–201 (2006)

3

Service Composition

George Baryannis[1], Olha Danylevych[2], Dimka Karastoyanova[2],
Kyriakos Kritikos[1], Philipp Leitner[3], Florian Rosenberg[3], and
Branimir Wetzstein[2]

[1] University of Crete, Greece
[2] University of Stuttgart, Germany
[3] Technische Universität Wien, Vienna, Austria

Chapter Overview. In the S-Cube research framework, the Service Composition and Co-ordination (SCC) layer encompasses the functions required for the aggregation of multiple services into a single composite service offering, with the execution of the constituent services in a composition controlled through the Service Infrastructure (SI) layer. The SCC layer manages the control and data flow between the services in a service-based application by, for example, specifying workflow models and using a workflow engine for runtime control of service execution.

This chapter presents an overview of the state-of-the-art in service composition modeling and covers two main areas: service composition models and languages and approaches to the synthesis of service compositions including model-driven, automated, and QoS-aware service composition. The contents of this chapter can be seen as a basis for aligning and improving existing approaches and solutions for service composition and provide directions for future S-Cube research.

3.1 Introduction

Service oriented computing (SOC) [34] is a paradigm that uses services as building blocks to create loosely-coupled software solutions. In this context, services are provided by software components that are described, discovered and composed in an interoperable way. They implement functionality ranging from low-level technical aspects to complete business processes [63]. Service-based applications (SBAs) are built on a service-oriented architecture (SOA). Web services [18], including standards such as WSDL, SOAP, and WS-BPEL are, currently, the most popular realization of a SOA.

An important property of services taking part in a SOA is their composability [18]. A *service composition* combines several services together to achieve a certain goal. There are several different usages for service composition which also result in different languages and technologies. Business processes can be implemented by orchestrating existing business functions implemented as services. Such a *service orchestration* implements a part of a

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 55–84, 2010.

business process and is again exposed as a service that can be used in other business processes. *Service coordination* is needed when services need to follow a coordination protocol driven by a central coordinator in order to perform a distributed activity. Consider, for example, an auction scenario, where the actions of sellers and bidders and the outcome of the auction are driven by a auctioneer. *Service choreography* models are used for describing service interactions from a global point of view. They are often used to describe the public processes of different business partners, in contrast to service orchestrations which describe the executable private process of each single partner. Finally, *service assembly* models define how different services should be packaged together into a deployable software solution. Some of these composition model types can often be combined together. For example, after creating a service choreography model, it can be refined to service orchestrations for each partner.

Considering the lifecycle of service compositions, we can distinguish between four phases: synthesis, execution, monitoring, and adaptation. In the synthesis phase, service compositions are created in either a top-down or a bottom-up fashion. A *model-driven service composition* approach involves creating more abstract models, such as business process models, and then generating the service composition or parts of it automatically. Bottom-up approaches involve automated service composition and QoS-aware service composition. In *automated service composition*, existing services are composed automatically based on a predefined abstract goal, e.g., by using AI planning techniques. *QoS-aware service composition* ensures that the overall composition achieves certain QoS targets when services are selected. In the second phase of the lifecycle, after the composition is created, it can be deployed and executed on the corresponding service infrastructure. For example, a service orchestration engine is used for execution of service orchestration models. At process runtime, the service composition is monitored and based on monitoring results can be adapted, thus closing the lifecycle. In this chapter we focus on the synthesis phase while other phases are dealt with in other chapters of this book as described further below.

The goal of this chapter is to present an overview of the state-of-the-art in service composition modeling. It is structured into two main sections. In Section 3.2 we describe and compare different service composition models and languages. In Section 3.3 we then present approaches to synthesis of service compositions including model-driven, automated, and QoS-aware service composition. In each subsection we present and classify research directions and approaches and elaborate on the most important papers in each category. Please note that this is not meant to be a complete report on all existing publications in the area. Often only a single and more prominent approach has been included in the survey.

3.2 Service Composition Models and Languages

In a broader sense, service composition can be seen as a combination of a set of services for achieving a certain purpose. Interchangeably, the term *service aggregation* can be used in this context [39]. In a narrower sense, the term service composition is typically used alternatively for service orchestration, as discussed below.

We will use the term in its broader sense and thus can consider four types of service composition models:

- **Service Orchestration:** In service orchestration, a new service is created by combining several existing services in a process flow. The standard language for orchestrating Web services is WS-BPEL.
- **Service Choreography:** A service choreography defines the interaction protocol between services. Service choreographies are specified by means of languages such as WS-CDL or BPEL4Chor.
- **Service Coordination:** Service coordination models are needed when several services have to agree on the outcome of a distributed process by using a central coordinator and a coordination protocol. Service coordination models in the context of Web services can be defined using WS-Coordination.
- **Service Assembly:** Service assembly is performed during deployment of service based applications. A service assembly is a deployable artifact, which is deployed to an enterprise service bus. Service Component Architecture implements the service assembly model.

In addition to the four listed composition types, we will also present *Semantic Web service Composition* in this section. It is not a separate composition type, but extends existing models and languages, in particular service orchestration, by use of semantic technologies in the context of Semantic Web services.

In the following subsections we will describe each composition type in detail by describing its concepts, and presenting and comparing the main languages and approaches.

3.2.1 Service Orchestration

A service orchestration composes a new service by reusing existing services. Service orchestration is a recursive model. Services which are orchestrated can be atomic services, i.e., services which do not use any other services, or again service orchestrations.

A service orchestration can be seen as a proactive composition that drives the interactions with the services it orchestrates. It defines execution semantics on a set of activities that perform these interactions. These execution semantics can be defined in different ways, e.g., calculus-based (XLANG), statechart-based [11], or graph-based (WSFL).

Service orchestrations can be created using a multi-purpose 3GL programming language, such as Java, but more typically a special language is used which deals with services as first-class citizens. In the context of Web services, there have been several efforts related to Web service orchestration using workflow languages, e.g., WSFL [46], XLANG [73], BPML [7], BPEL [60]. In the context of Web services, these service orchestrations are called WS-flows. Besides WS-Flow languages, there are other approaches to service orchestration, such as JOpera [64] which is a visual service composition approach, not constrained to orchestration of (WSDL-based) Web services.

WS-Flow Languages

Web service flow languages enable construction of service orchestration using workflow like constructs whereby the invoked applicaions out of the workflow are Web services. There have been several efforts related to the creation of WS-Flow languages, in particular, WSFL [46], XLANG [73], BPML [7]. WS-BPEL [60] (BPEL) has superseded these languages and has emerged as a standard in the Web services community. It can be seen as the successor of WSFL from IBM and XLANG from Microsoft and it combines concepts and constructs from both languages. In the following, we will focus on WS-BPEL.

WS-Flows can be thought of having two dimensions: the "what?" dimension, and the "what with?" dimension. The "what?" dimension is represented by the control flow and the data flow, and is also referred as business logic. The control flow models alternative paths of execution, defines how exceptional situations are handled (fault handling and compensation handling) and how the process reacts to events signalled by the environment (event handling). The control flow of a BPEL process is specified using a combination of graph-based (WSFL) and calculus-based (XLANG) approaches. Another aspect of the "what?" dimension is the data flow which defines how data is exchanged among activities and between the workflow and its participants. The data flow is explicitly present in only some of the existing WS-Flow languages, such as WSFL, and XLANG. In BPEL, however, data flow is implicit. Activities can access data variables, which can be defined either globally for the whole process or for specific (nested) scopes (which restrict their visibility).

The second dimension of WS-flows, the "what with?" dimension, is the one assigning to each interacting activity a participant, namely a Web service. Interacting activities are those tasks in the model of a WS-flow that stand for interaction with a partner Web service. The definitions of these activities specify the participating Web services only on the abstract level, namely only their portTypes and operations are to be supplied, and refer to no actual endpoints that implement these portTypes. Mechanisms for binding service instances to a process are intentionally left up to the runtime and therefore out of the process definition. This is where service assembly mechanisms come into play (see Section 3.2.4).

BPEL only supports the orchestration of Web services described using WSDL. There are, however, extensions to BPEL which relax this restriction. BPEL4People enables incorporating human tasks into a BPEL process. BPELlight [58] removes the dependency on WSDL altogether, and describes just conversations as message exchanges with partner services.

BPEL process models can be defined as abstract or executable. An executable process model specifies a service orchestration which can be executed by a BPEL engine. In contrast, an abstract process model hides some activities (a part of the process model) by defining them as “opaque”. Abstract process models can be used to define process templates or behavioral interfaces. A behavioral interface contains mostly messaging activities which denote how the service requester should communicate with the process thus specifying its public process which can be seen by external requesters. Private information (process logic) is replaced by opaque activities.

3.2.2 Service Choreography

Service choreography provides the global point of view on existing and future multi-party collaborations, as opposed to the local perspective provided by service orchestration. Each participant in a service choreography can be modeled as a service orchestration, called participant implementations. The choreography “ties” the participant implementations in a global collaboration by specifying the one-to-one and one-to-many message-based interactions among them. The internal structure of each participant implementation is outside the focus of choreography, and it is interesting only insofar it affects the participant’s behavior as perceived from the outside. In fact, a participant of choreography is described only in terms of its messaging behavior. The messaging behavior (also known as business protocol) can be represented for example as an abstract process whose actions are the consumption and production of messages.

The goal of choreography notations is to support the description of message-based interactions among their participants. The expressiveness of a choreography notation can be estimated by its ability to express (describe) different types of message-based interactions. The Service Interaction Patterns [10, 78] describe the recurring interaction scenarios among services in SOA. They outline the different ways of realizing interactions, e.g., bilateral, multilateral, competing, atomic and causally related. Since participants in a choreography are usually implemented as Web services, the Service Interaction Patterns have been used to evaluate the expressiveness of some choreography notations. Examples of such classifications can be found in [23, 21].

Alternatively to the support of Service Interaction Patterns, choreography notations can be categorized on the basis of the following two dimensions: (1) *Interaction modeling paradigm:* how the notation describes the interactions among the participants; (2) *Level of abstraction:* to which extent the choreography notation is tied to a particular set of technologies.

The main interaction modeling paradigms are *Interaction* and *Interconnected Interface* see [41]. The Interaction paradigm describes the conversations by combining elementary interaction blocks (e.g., request-response and one-way) into more complex interactions that define the dependencies among them. The data and control flow are defined globally instead of being assigned to the single roles. The advantage of interaction paradigm is that it supports the early stages of service development lifecycle by capturing the overview of interactions between the identified potential services. However, the drawback is that, since the interactions are defined globally, "outside" the participants, it is possible to define interactions that are not *realizable* by the choreography participants (e.g., [28, 20]).

The Interconnected Interface paradigm takes an opposite approach with respect to the Interaction one. The logic of the interactions is not defined globally, but spread across the participants. In fact, the Interconnected Interface paradigm specifies separately the logic of each participant. The overall logic is (implicitly) given by the data and control flow dependencies defined between the send and receive activities inside each participant's logic. Because of its focus on each participant separately, the Interconnected Interface paradigm is well suited for both adapting existing collaboration-enabled processes and creating new ones. Moreover, since each participant's logic is defined separately, this paradigm is not concerned with the problem of realizability. However, the interfaces of participants may be incompatible, e.g., resulting in deadlocks during the enactment [52].

Regarding the level of abstraction, the various choreography notations can be classified as either *implementation specific* or *implementation independent*. On one hand, implementation specific choreography notations cover technical aspects of the interactions such as communication protocol, security issues and exception handling. Implementation specific choreography notations usually assume participant implementations to be Web services, and specify the technical details using the related technologies (SOAP, WS-Security, WS-ReliableMessaging, WS-Policy, etc.). On the other hand, implementation-independent notations do not encompass technical details. Their goal is generally to support the Business Process Management community in making decisions about the collaborations on the basis of provided overview of interactions from business processes.

Table 3.1 presents the classification of the main choreography languages in terms of the "Interaction modeling paradigm" and "Level of abstraction" criteria. In Table 3.1 we classify two types of notations:

1. Industrial: WS-CDL [80], Let's Dance [86], BPMN [59] and BPEL4Chor [22]. These languages are focused on usability for the end user, and generally result from industrial efforts (e.g., consortia) or collaborations between industry and academy. Unfortunately, they often lack a formal underpinning (e.g., a formal semantics), which is often added a posteriori instead of being an integral part of the specifications [9].

Table 3.1. A Categorization of Choreography Notations

	Implementation independent	**Implementation specific**
Interaction	Let's Dance, automata-based notations, Process Algebras	WS-CDL
Interconnected Interface	BPMN, Petri Net-based notations	BPEL4Chor

2. Formal: these notations are usually developed in academic approaches. They can be split into three categories:
   - Automata-based, e.g., timed automata [51], conversation protocols [28] and Mealy services [13];
   - Petri Net-based, e.g., [24, 33];
   - Process algebras, e.g., [14, 56, 84, 69].

The formal notations are specifically useful because they enable the analysis and verification of different properties of the choreographies built on the wide spectrum of available verification methods for their underpinning formalisms. The verification of choreographies covers many different properties. Some examples are: *Conformance*, i.e., whether a participant implementation behaves in terms of message exchanges as mandated by the choreography; *Realizability*, i.e., whether it is possible to create the implementations of participants that conform to the given choreography (already covered when addressing the Interaction paradigm); *Deadlock freeness*, i.e., whether the enactments of the choreography can deadlock; *Synchronizability*, i.e., whether the enactments of the choreography can execute the same traces assuming both synchronous and asynchronous messaging among the participants. The various verifications methods for choreographies have been investigated in an extremely broad and increasing body of research. For a more complete overview of the verification of choreographies, the interested reader is referred to overviews such as [76].

3.2.3 Service Coordination

Service coordination denotes a model in which a set of participating service instances perform a distributed activity by following a coordination protocol. A coordinator decides on the outcome of the protocol (e.g., success or failure) and informs the participants of the result.

Service choreography is similar to service coordination in that it also models the interactions between services, however service coordination is much more loosely constrained as the participants do not have to communicate with each other, but communicate with the coordinator who drives the coordination protocol. One could use service choeography languages to model the interactions between the participant and the coordinator and also the protocol logic in the coordinator.

The typical usage of coorditnation is the implementation of transactional processing. For example, as described further below, there are existing coordination protocols for implementing ACID transactions ot long-running business transactions, which can be used between services to agree on the outcome. There are also non-transactional use cases, such as the already mentuioned auction scenario, which can leverage coordination frameworks [47].

Web Service Coordination

In the context of Web services, there are two competing specifications, WS-Coordination [4] and Web services Composite Application Framework (WS-CAF) [83]. As they are relatively similar in their approaches, we will in the following present only WS-Coordination.

WS-Coordination [4] defines a framework for coordinating interactions between Web services. It enables participting Web services to reach agreement on the outcome of activities using a coordinator based on a coordination protocol. WS-Coordination is extensible in respect to coordination protocols. Two such protocol specifications already exist specifying transaction protocols, namely the WS-AtomicTransaction specification [82] for atomic (2PC) transactions and the WS-BusinessActivity [3] specification for long-running business transactions.

In WS-Coordination, coordinated interactions are called *activities*. An activity ties together several participants into a (distributed) application. A *participant* is a Web service that contributes to an activity. A *coordinator* uses the coordination protocol to mediate between participants on the outcome of the activity. Therefore, participants and the coordinator exchange messages specified in a coordination protocol in order to agree on an outcome of the activity. A coordination protocol consists of a set of messages and a specification of how these messages are to be exchanged.

The coordinator provides three services needed for using the framework: (1) *Activation service:* The coordinated activity is started when a service in its role as an *initiator* requests a *coordination context* from the activation service. The coordination context consists of an activity identifier, the coordination type (e.g., atomic transaction), and the endpoint reference of the registration service. The initiator distributes the coordination context to the participant Web services. (2) *Registration service:* Before starting its work, the participant registers at the registration service of the coordinator. (3) *Protocol service:* At some later point the protocol service is started which coordinates the outcome according to the specific protocol of the coordination type. The first two are needed to initialize the distributed activity, the latter is used for running the coordination protocol resulting in an outcome.

3.2.4 Service Assembly

A service-based application typically consists of several services which interact with each other. Each service thereby provides an interface to other services, but also defines requested interfaces of other services which it uses. Services specify provided and requested interfaces in form of operations with inputs and outputs, e.g., BPEL orchestrations use partnerlink types in order to specify provided and requested service interfaces.

In order to create an executable service based application, requested and provided service interfaces have to be *wired* together, thus creating a so called *service assembly*. A service assembly is a deployable artefact which is installed in an enterprise service bus. It is exposed to the outside as a service over a certain protocol such as SOAP/HTTP, namely by using the provided interface of the first service in the wiring chain. A service assembly can again be recursively wired to other service assemblies.

Service Component Architecture

Service Component Architecture (SCA) [19] is a set of specifications that provide an assembly model for building composite applications based on a SOA. It is based on the idea that a service-based application consists of several services, which are wired together to create solutions for implementing a particular business function. SCA provides a model for both the creation of service components and assembly of these components into composite applications. It supports a wide range of technologies for the implementation of service components and for the communication mechanisms which are used to connect them.

The SCA Assembly Model is the main specification which defines the configuration of an *SCA System*. The SCA Assembly Model consists of a series of artifacts. The main artifact is the *Composite*, which is the unit of deployment and which can be accessed remotely. A Composite contains one or more *Service Components*. Components offer their function as *Services*, which can either be used by other Service Components within the same Composite or which can be made available for use outside the module. Service Components may also depend on Services provided by other Components. These dependencies are called *References*. References can either be wired to services provided by other components in the same module, or References can be linked to Services provided outside the Composite, which can be provided by other Composites or arbitrary external services. All Service Components in the Composite are linked together, by connecting References with corresponding Services using *Wires*. A Component consists of an implementation, where an implementation is the piece of program code implementing a business function (e.g., in Java or BPEL). The configuration of an SCA system has a standardized XML representation. That configuration can be seen as a deployment descriptor for the SCA System. The SCA System is deployed to an SCA Runtime which is part of an Enterprise Service Bus.

Finally, for supporting non-functional properties such as security, SCA provides a Policy Framework to support specification of constraints, capabilities and Quality of Service (QoS) expectations, from component design through to concrete deployment.

3.2.5 Semantic Web Service Composition

Whilst promising to revolutionize e-Commerce and enterprise-wide integration, current standard technologies for Web services (e.g., WSDL) provide only syntactic-level descriptions of their functionalities, without any formal definition to what those syntactic definitions might mean. This lack of machine readable semantics necessitates human intervention for automated service discovery and composition within open systems, thus hampering their usage in complex business contexts.

Semantic Web services (SWSs) relax this restriction by augmenting Web services with rich formal descriptions of their capabilities, thus facilitating automated composition, discovery, dynamic binding and invocation of services within an open environment. A prerequisite to this, however, is the emergence and evolution of the Semantic Web (SW), which provides the infrastructure for the semantic interoperability of Web services.

Semantic annotation of Web services description has been the issue of many initiatives, projects and languages introduced, where the most significant among them are: the OWL-S Semantic Markup for Web services [77] and the Web service Modeling Ontology (WSMO) [43]. We also analyze an effort to extend the BPEL standard with ontological annotations. Finally, we compare all of these languages based on their Web service composition capabilities.

OWL-S

OWL-S consists of a set of ontologies designed for describing and reasoning over service descriptions. It consists of three main upper ontologies: the *Profile*, *Process Model* and *Grounding*. The *Profile* contains a description of service properties for the purposes of service discovery. These properties include both functional (i.e., inputs, outputs, preconditions, and effects – IOPEs) and non-functional (e.g., QoS) ones.

OWL-S process models describe the behavioral interface of a service and are not executable, similarly to an abstract BPEL process. They are used both for reasoning about possible compositions (validation, verification, etc.) and for controlling the invocation of a service. The *atomic* process is a single, black-box process description with exposed IOPEs. *Composite* processes are hierarchically defined workflows, consisting of atomic, simple, and other composite processes, constructed using a number of different composition constructs.

The profile and process models provide semantic frameworks whereby services can be discovered and invoked, based upon conceptual descriptions defined within OWL ontologies. The *grounding* provides a pragmatic binding between this concept space and the physical data/machine/port space, thus facilitating service execution. The process model is mapped to a WSDL description of the service. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types.

WSMO

WSMO (Web service Modeling Ontology) is a conceptual model for describing semantically-enhanced Web services. The WSMO framework consists of four main modeling elements: *Ontologies*, *Web services*, *Goals* and *Mediators*. *Ontologies* provide terminologies for defining the formal semantics of information resources. WSMO ontologies are classic ontologies with classes, relations, instances and axioms, plus functions (n-ary relations) and some additional information coded as non-functional properties such as QoS or metadata.

WSMO Web services descriptions consist of functional, non-functional and behavioral aspects, of a Web service. The functional aspect of a WSMO Web service is described through one *capability* using similar concepts with OWL-S IOPEs. A capability is linked to several goals via mediators. An interface (behavioral aspect) describes how the functionality of the Web service can be achieved by providing a twofold view on the operational competence of the Web service: a) a *choreography* and b) an *orchestration*. A WSMO choreography specifies a behavioral interface of the service, and not a complete choreography model. Both WSMO choreography and WSMO orchestration are based on Abstract State Machines (ASMs) [12].

Goals are limited descriptions of Web services (a capability and an interface) that would potentially satisfy the user desires. WSMO model definition follows a goal-driven approach, which means that requests and services are strongly decoupled. Finally, Mediators connect heterogeneous components of a WSMO description which have structural, semantic or conceptual incompatibilities. Mismatches can arise at the data or process level. There exist four types of mediators: ontology, goals, Web service-to-goal and Web service-to-Web service mediators.

BPEL4SWS

The Web service stack of standards (i.e., WS-*) currently relies on BPEL for executing orchestrations. BPEL is using WSDL descriptions to identify partner/component services in the process models. Unfortunately, effective dynamic service binding cannot be performed by solely matching WSDL messaging interfaces. Restricting service descriptions to the expressivity of strictly syntactic WSDL interfaces limits the integration of service partners that operate on messages that have different syntax but are semantically compatible. The solution to the dynamic selection of Web services comes with Semantic Web services (SWS). SWS describe services not in terms of an interface but rather describe their functionality and capability semantically and in a machine processable manner.

As a result, a new language called BPEL4SWS [1] was introduced that extends BPEL. This language uses BPELlight [58] as basis and allows to attach SWS descriptions to BPELlight such that SWS frameworks like OWL-S and WSMO and corresponding implementations can be used to discover and select SWS that implement the functionality required for an activity. As WSMO distinguishes between a service provider (Web service) and a service consumer (goal) and thus enables asynchronous communication between Web services, it is preferred to OWL-S. Both, the SWS description and the process itself are partly grounded to WSDL to facilitate WS-* based communication. BPEL4SWS processes are able to use both SWSs as well as conventional Web services intermixed within a single process. Moreover, BPEL4SWS processes themselves can be exposed both as SWSs and conventional Web services. Current SWS frameworks use ontologies as data model to facilitate semantic discovery. For that reason, SAWSDL [27] is used to enable a seamless mapping of data between its XML representation and its ontological representation. This is required because in BPEL4SWS Web services and SWSs can be mixed.

Comparison

From all SWSs languages, OWL-S is the most mature and has been used in many Web service discovery research approaches that include both functional and non-functional characteristics of Web services. However, it has been proven that OWL-S has some drawbacks [8] that prevent it from being used in practical real-world scenarios. Concerning composition synthesis, OWL-S has been successfully used in AI planning [53] for automatically producing and executing Web service composition plans. Moreover, it can be used for the analysis and validation of Web service compositions as it can be mapped to (colored) Petri-nets [53, 49]. Unfortunately, as far as orchestration is concerned, OWL-S presents two major drawbacks. First of all, its process model is neither an orchestration model (although it resembles one) nor a conversation model but it just describes the behavioral interface of the service. Secondly, OWL-S describes the execution of a Web service as a collection of remote procedure calls. Unfortunately, this is only correct for a small percentage of the cases in business processes [1] as the communication is typically asynchronous.

WSMO has the same discovery capabilities as OWL-S and has also been used in many Web service discovery research approaches. It is starting to attract many SWS researchers as there exist a lot of SW tools that support it. Concerning Web service composition, WSMO contains a choreography model that can be easily mapped to ASMs. In this way, WSMO inherits the core principles of ASMs which include the modelling of state changes by guarded transition rules. However, the current transition rules in WSMO Choreography can only represent local constraints. In addition, WSMO Choreography needs to be formalized in order to allow reasoning and verification about it. Moreover, WSMO contains an orchestration model which is quite primitive. So, WSMO can only be used in a limited way to orchestrate the interactions of Web service-based processes.

BPEL's lack of dynamic service binding and choreography model led to the synergy with WSMO and SAWSDL with BPEL4SWS as the result. Through this synergy, the goal of dynamic service binding has been achieved. Now, BPEL4SWS is better in comparison to OWL-S concerning orchestration. Concerning the composition synthesis capability, there are many research approaches that have been applied to BPEL. So BPEL4SWS seems to be the right language for the Web service composition process.

3.3 Service Composition Synthesis Approaches

The lifecycle of service compositions consists of four phases: synthesis, execution, monitoring and adaptation. In this Section we will focus on the first phase, as the other phases are described in other chapters of this book (see Section 1).

When creating service compositions, we can distinguish between top-down and bottom-up approaches. In a top-down approach, the composition logic is first created on a higher abstraction level, e.g., as a business process model, and then this abstract model is refined to a service composition model by selecting appropriate services. In *model-driven service composition*, the composition model is automatically generated from a more abstract process model. In *QoS-aware service composition*, services are selected based on local and global QoS constraints. A bottom-up approach tries to come up with the composition logic by combining existing services in order to achieve a certain abstract goal. In *automated service composition*, the services are combined automatically, e.g., based on their pre- and postconditions using AI planning techniques.

In the following subsections, we will discuss these three main synthesis approaches, namely model-driven, QoS-aware, and automated composition, in detail.

3.3.1 Model-Driven Service Composition

Model-Driven Service Composition is the application of model-driven development (MDD) ideas, such as the Model-Driven Architecture (MDA), to the problem of service composition. MDA has been put forward by the Object Management Group (OMG) in 2001, and is currently the de facto standard for MDD. In MDA, all software functionality is specified using platform-independent models (PIMs). Platform definition models (PDMs) are used to translate PIMs to various platform-specific models (PSMs), which can either be compiled into executable code or run directly. Many authors have presented approaches to service composition, which exhibit similar characteristics [66].

One important high-level language to model orchestrations is the UML [74]. Here, UML Activity Diagrams are used to define the orchestration, which are then transformed into executable representations. The initial prototype in [74] supports both WS-BPEL and WorkSCo.[4] One advantage of this approach is the endorsement of a complete roundtrip model, where existing WSDL contracts can be refactored into UML models, which are in turn arranged into orchestrations; these can then be transformed to executable code as discussed above, and deployed on a composition engine. Since both WS-BPEL and WorkSCo are XML-based languages the actual transformation process is implemented using XSLT.

This work has later been extended to also consider semantic Web service composition and QoS attributes [31] (see Section 3.3.2 below for more details on QoS-aware composition). This extension considerably improved on the support for run-time discovery of services and run-time service binding: in [74] the service bindings are statically defined at design-time, while [31] considers abstract bindings to specific functionality defined using semantic annotations; concrete services implementing this functionality are discovered at run-time using semantic discovery and matchmaking technologies. Additionally, services are ranked based on QoS properties, i.e., if more than one service instance is discovered during run-time the one exhibiting the best QoS properties is selected. However, the work is based on a set of assumptions (e.g., the existence of a semantic matchmaking and service discovery entity), which are not fulfilled even today.

Already in 2003, Koehler et al. have presented an approach that is more clearly aligned to the ideas of MDA [40] and Business Driven Development (BDD) [55]. Their work presents the transformation of business models (represented in ADF and again in the UML) to technology-level WS-BPEL code. Unlike other approaches, their work considers not only the top-down transformation from high-level models to code, but also the other direction, i.e., the bottom-up re-engineering of existing WS-BPEL compositions into business processes. This is different to the round-trip model of [74] in that it also allows for bottom-up construction of high-level composition models. The transformation architecture consists of four levels: business models are firstly transformed to process graphs, which are in turn refined to flow graphs; subflows can then be compiled to solution components using platform-specific transformation. Bottom-up, solution components are combined to create flow graphs, which are merged to process graphs, and finally the business model can be restored. Later on, these ideas have been refined in [32]. In this paper

[4] `http://www.esw.inesc-id.pt/worksco/`

the authors detail how graph-based process models can be transformed into executable code using graph transformation and compiler theory.

Ouyang et al. use a different language, OMGs Business Process Modeling Notation (BPMN), to represent business models [62]. They argue that BPMN is more common in the business world than the more software-centric UML. Additionally, BPMN is well supported by business analyst tools. However, the transformation from BPMN to WS-BPEL is rather complex, since BPMN is a graph-oriented language and fundamentally different than the block-based WS-BPEL. However, the authors have still presented a complete and fully automated transformation approach, which produces WS-BPEL code that is also comprehensible for humans and well-structured.

In [61], Orriens et al. have presented a model-based composition approach based on composition rules. These rules are expressed in OCL (the Object Constraint Language, part of the UML), and are used to model constraints on possible compositions. Examples of such rules are `activity.function = ‘‘FlightBooking’’` (i.e., the function of the activity always has to be “FlightBooking”) or `activity.input - > notEmpty` (i.e., there always has to be an input message for this activity). Generally, they distinguish between rules which are used to structure activities within the composition (structural rules), rules that relate messages to each other (data rules), rules that protect the integrity of the composition, by enforcing preconditions, postconditions and invariants (behavioral rules), rules concerning the usage of resources (resource rules) and rules that specify the behavior of the composition in case of errors or unexpected behavior (exception rules). An automated dynamic service composition engine is used to create an orchestration from a set of input rules. However, of course this approach implies that the composition has to be fully specified using business rules, which is not always easy in practice.

Another form of model-driven service composition is mapping service orchestrations specified in WS-CDL to WS-BPEL. This approach is discussed e.g., by Mendling and Hafner [54]. WS-CDL defines the externally visible behavior of business entities, i.e., which messages are transferred to which business partner at which point in the interaction, and what options are available. Additionally, WS-CDL relates operations to concrete states in the business processes. Mendling and Hafner propose to use the inter-business WS-CDL model as basis to generate WS-BPEL code for each specific participating partner. They show that many WS-CDL constructs can be mapped directly (such as `cdl:sequence`, `cdl:parallel`, `cdl:assign`, and some others), but other constructs are more complex to map. For example, it is not easy to identically map the semantics of a `cdl:workunit` construct, since no similar construct is available in WS-BPEL. In this case, the authors propose to map the construct to a `bpel:scope`, but a human BPEL engineer is required to eventually add additional information that cannot be deduced from the choreography automatically. Therefore, this approach can be considered only semi-automated — a fully autonomous mapping from WS-CDL to WS-BPEL without human interaction is currently not possible.

Table 3.2. Comparison of Model-Driven Composition Approaches

	Business Model	Composition Model	Top-Down?	Bottom-Up?	QoS	Semantics
[74]	UML AD	WS-BPEL and WorkSCo	+	-	-	-
[31]	UML AD	Not Explained	+	-	+	+
[40]	ADF and UML AD	WS-BPEL	+	+	-	-
[62]	BPMN	WS-BPEL	+	-	-	-
[54]	WS-CDL	WS-BPEL	+	-	-	-

The most important presented approaches are summarized in Table 3.2. In the table, "Not Explained" represents properties which are not mentioned in the concerning paper. UML Activity Diagrams are abbreviated as "UML AD".

As can be seen from the table and the discussion above, many approaches exist which map widely used higher-level languages such as the UML to service compositions (mostly WS-BPEL, since WS-BPEL is the de facto standard for specifying compositions nowadays). However, current research issues such as the inclusion of QoS or service semantics are rarely taken into account. Additionally, little work exists in the area of Bottom-Up model-driven service composition, i.e., the extraction of higher-level models from existing service compositions.

3.3.2 QoS-Aware Service Composition

Service-oriented systems in general and service compositions in particular are often required to adapt themselves to different situations. For example, a service in a composition might yield intermittent or even permanent failures, therefore, exchanging such a service by an equivalent service automatically is necessary to meet certain service level agreements. A key enabler for realizing such adaptive behavior is Quality of Service (QoS) as a means to describe all non-functional attributes of a service. QoS attributes can be grouped into deterministic and non-deterministic attributes [72]. These include performance-specific aspects such as response time or throughput of a service, dependability-specific aspects such as availability or cost-related data [48]. Deterministic QoS attributes, on the one hand, indicate that their value is known before a service is invoked, including price or the supported security protocols. On the other hand, their non-deterministic counterpart includes all attributes that are uncertain at service invocation time, for example the service response time. Therefore, the availability of accurate non-deterministic information QoS (through QoS monitoring) plays a crucial role during development and execution of a composite application.

Firstly, QoS enables a QoS-aware dynamic binding to concrete services that are available in registries known at runtime. Secondly, QoS enables an optimization of composite services in terms of its overall QoS and adaptation of services whenever QoS changes. We denote a composite service leveraging QoS to enable adaptive behavior as *QoS-aware composite service* and the engineering process as *QoS-aware service composition.*

Over the last years a number of approaches have been introduced to deal with the problem of how to compose service to build QoS-optimized compositions using a variety of approaches and algorithms. In the following, we describe selected composition and optimization approaches.

Combinatorial Approaches

Zeng et al. [87, 88] present a QoS optimization approach by splitting a composition into multiple execution paths based on their notation that a composition is specified using a state chart diagram. Each execution path is considered to be a directed acyclic graph (DAG) given the assumption that the state chart has no cycles. Additionally, the authors define an execution plan which is basically a set of pairs expressing that for every task in the composition, a service exists that implements the operations required for that task. For the local optimization, the system tries to find all candidate Web services that implement the given task. Each service is assigned a quality vector and user defined scores for the different quality constraints. These constraints are then used to compute a score for each candidate service. Based on Multiple Criteria Decision Making (MCDM), a service is chosen which fulfills all requirements and has the highest score. In order to find a global optimum, the authors propose an optimization approach based on Integer Programming (IP). To achieve a QoS-aware optimization of the composition, global and local QoS constraints can be specified. In addition, an objective function has to be maximized. The optimization problem is then solved using an IP solver. In addition to optimizing a composition, the authors also describe an approach to re-plan and re-optimize a composition based on the fact that QoS can change over time.

Yu et al. [85] discuss algorithms for Web services selection with end-to-end QoS constraints. Their approach is based on different composition patterns similar to [36] and they group their algorithms according to flows that have a sequential structure and ones that solve the composition problem for general flows (i.e., flow that can have splits, loops etc). Based on this distinction, two models are devised to solve the service selection problem: a combinatorial model that defines the problem as multidimensional multi-choice knapsack problem (MMKP) and the graph model that defines the problem as a multi-constrained optimal path (MCOP) problem. These models allow the specification of user-defined utility functions that allow to optimize some application-specific parameters and the specification of multiple QoS criteria taking global QoS into account. In the case of the combinatorial model, the authors use an MMKP algorithm, that is known to be NP-complete. Therefore the authors apply different heuristics to solve it in polynomial time. For the general flow structure the authors use an IP programming solution (also NP complete), thus they again apply different heuristics to reduce the time complexity.

Jaeger et al. [36, 35] present a different approach of deriving QoS of a composite services by following an aggregation approach that is based on the well-known workflow patterns by van der Aalst et al. [79]. The authors analyze all workflow patterns for their suitability and applicability to composition and then derive a set of seven abstractions that are well suited for compositions, called composition patterns. Additionally, the authors define a simple QoS model consisting of execution time, cost, encryption, throughput, and uptime probability and QoS aggregation formulas for each pattern. The computation of the overall QoS of a composition is then realized by performing a stepwise graph transformation, that identifies a pattern in a graph, calculates its QoS according to the aggregation functions and the replaces the calculated pattern with a single node in the graph. The process is repeated until the graph is completely processes and only one single node remains (which is itself a sequence according to their composition pattern). For optimizing a composition, the authors analyze two classes of algorithms, namely the 0/1-Knapsack problem and the Resource Constrained Project Scheduling Problem (RCSP). For both algorithms a number of heuristics are defined to solve the problems more efficiently.

Evolutionary Approaches

Canfora et al. [29] propose an approach that tries to solve the QoS-aware composition problem by applying genetic algorithms. Firstly, the authors describe an approach to compute the QoS of an aggregated service, similar to [16, 36, 37]. Secondly, the main issue of their approach, namely the encoding of QoS-aware service composition problem as genome, is presented. The genome is represented by an integer array with the number items equals to the number of distinct abstract services composing our service. Each item, in turn, contains an index to the array of the concrete services matching that abstract service. The crossover operator is a standard two-point crossover, while the mutation operator randomly selects an abstract service (position in the genome) and randomly replaces the corresponding concrete service with another one from the pool of available concrete service. The selection problem is modeled as a dynamic fitness function with the goal to maximize some QoS attributes (e.g., response time) while minimizing others (e.g., costs). Additionally, the fitness function must penalize individuals that do not meet the QoS constraints. The authors evaluate their approach by comparing it to well-known integer programming techniques. In another paper, the authors also describe an approach that allows re-planning of existing service compositions based on slicing [15].

Stochastic Approaches

Mukhija et al. [57] present the Dino framework that is targeted for the use in open dynamic environments. Their main idea is that no global view on a service composition is available and thus each service specifies which services it requires for its own execution. The service composition is formed at runtime by the infrastructure. A key aspect is the fact that service requirements can change dynamically (triggered for example by changing application context). Dino also supports QoS-aware service composition by describing it formally using an ontology. QoS is computed by using the actual and estimated QoS values that are monitored by the Dino broker. QoS computation is modeled by using a continuous-time Markov chain that enables the association of a probability value that expresses the confidence of the QoS specification.

Wiesemann et al. [81] present a QoS-aware composition model that is based on stochastic programming techniques. Their model takes QoS-related uncertainty into account that minimized the average-value-at-risk (AVaR) - a particular quantile-based risk measure - from the stochastic programming domain. The trade-off between different QoS criteria is modeled by a combination of goal weighting and satisficing two prominent techniques in multi-criteria optimization with the goal to minimize the AVaR.

Combined Approaches

The aforementioned approaches focus very specifically on the optimization part of QoS-aware compositions and do not address, for example, other concerns such as the specification of QoS-aware composition or runtime-specific issues. Rosenberg et al. [71] provide a holistic, end-to-end environment for a constraint-based specification and generation of QoS-aware compositions using a domain-specific language (DSL) called VCL (Vienna Composition Language). VCL allows to specify microflow compositions using global and local constraints by providing Composition as a Service (CaaS). These constraints can be separated in soft- and hard-constraints. The former specify QoS constraints that are nice to have and are associated with a strength value to express their importance that is desired by the user whereas the latter specify constraints that need to be fulfilled. One the composition runtime has satisfied all the constraints and optimized the composition with respect to its QoS properties and executable and adaptive composition is generated.

Comparison

Table 3.3 summarizes the approaches according to key aspects of QoS-aware composition.

As it can be seen from Table 3.3, most approaches support simple and complex composition structures (such as sequences, loops, conditionals and parallel execution). Moreover, various constraint types can be specified, in

Table 3.3. Comparison of Approaches

Paper	**Composition**				**Reoptimization**
	Composition Constructs	**Constraint Types**	**Optimization Technique**	**Complexity**	
Zeng et al. (2004)	Simple and complex structures	Local and global (hard)	IP	NP-complete	Region-based
Yu et al. (2007)	Simple and complex structures	Local and global (hard)	MMKP, MCSP-K, IP	NP-complete	n/a
Jaeger et al. (2005)	Complex structures	Local and global (hard)	0/1 Knapsack and RCSP	NP-complete	n/a
Mukhija et al. (2007)	Simple structures	Local (hard)	Not specified	Unknown	n/a
Wiesemann et al. (2007)	Complex structures	Probabilistic local and global constraints	Stochastic Programming	Unknown	n/a
Rosenberg et al. (2009)	Simple and complex structures	Local and global (soft and hard)	IP, SA, GA, TS	NP-complete	On Invocation, Periodic Re-optimization

particular local and global constraints, however, mostly only hard constraints are allowed. Rosenberg et al. provides the only approach also allowing to specify soft constraints. With regards to the optimization approach, integer programming (IP) is the predominant approach, however, other approaches such as genetic algorithms (GA), simulated annealing (SA) and tabu search (TS) are also popular. Runtime re-composition of existing compositions is only supported by Zeng et al. and Rosenberg et al. (which use a re-composition mechanism every time a request is sent to the composite service and additionally also provide a periodic rebinding running in the background).

3.3.3 Automated Service Composition

A major family of approaches to Web service composition aim to fully or partially automate the process of composition in order to deal with its high level of complexity. Manual implementation of a composition schema is a time-consuming, error-prone and generally hard procedure and is most certainly not scalable. Therefore, the need for automation in Web service composition should be apparent. In [70], the authors propose a general framework for automatic Web service composition. Through this framework, one can extract five distinct phases that constitute a complete automatic composition approach.

Atomic Services Description

The first phase deals with the description and advertisement of atomic services that will be used as building blocks for the composition. Service advertisement can be achieved using UDDI or using the Service Profile class of OWL-S [77], which provides semantically annotated descriptions for Web services.

Internal Processes Description

The second phase focuses on service specifications that describe the internal processes of the service, essentially specifying how the service works. Formal and precise languages such as logic programming are required in this case in order to effectively capture the constraints describing the internals of a service.

Generation of Composition Process Model

This phase is the heart of the composition process, as it involves generating the composition process based on the requester's requirements. The result should be one or more process models that describe which services participate in the composition as well as the control and data flow among the participants.

Composition Evaluation and Execution

The final two phases involve the evaluation of the composition process models produced in the previous phase, the selection of the optimal one based on a set of non-functional attributes such as QoS aspects and the execution of the composite service according to the selected process model.

Automatic service composition approaches can be grouped into two distinct categories: workflow-related approaches and AI planning approaches. The first group draws mainly from the fact that a composite service is conceptually similar to a workflow, making it possible to exploit the accumulated knowledge in the workflow community in order to facilitate Web service composition. On the other hand, AI planning techniques involve generating a plan containing the series of actions required to reach the goal state set by the service requester, beginning from an initial state.

Workflow-Based Approaches

Composition frameworks based on workflow techniques were one of the initial solutions proposed for automatic Web service composition. In [42], the authors define the Transactional WorkFlow Ontology (TWFO), an ontology used to describe workflows. A registry containing OWL-S service descriptions is used to find services corresponding to the tasks in the composite workflow. In contrast to the previous work, where concrete workflows associated to services are combined to a master workflow, [5] and [50] deal with abstract workflows. [5] argues that business process flow specifications should be abstractly defined and only bound to concrete services at runtime. The authors present a BPEL engine that is able to take an abstract workflow specification and dynamically bind Web services in the flow taking into consideration any relations and dependencies between services as well as domain constraints. [50] advances one step further by allowing for the automatic generation of an abstract workflow based on a high-level goal.

More recent work seems to focus on bridging the gap between traditional workflow composition methods and techniques based on AI planning that will be presented in the next sections. For instance, [75] presents a workflow framework that contains a Planning Module and a Constraint Solving Problem Module. The first one produces candidate composite plans based only on functional requirements while the second one takes the resulting plans of the first module and selects the most appropriate service based on non-functional attributes.

AI Planning Approaches

AI research has extensively covered a complex problem known as planning, where an agent uses its beliefs about available actions and their consequences, in order to identify a solution over an abstract set of possible plans. An AI planning problem can be described as a quintuple $\{S, s_0, G, A, \Gamma\}$ [17], where S is the set of all possible world states, including initial state s_0 and goal states G, A is the set of actions that can be performed in order to reach a goal state and $\Gamma \subseteq S \times A \times S$ is a transition relation from one state to another when a particular action is executed. This problem definition can be applied to the case of Web service composition if we consider A to be the available services, G to be the goal set by the requester, and S, s_0 and Γ to refer to a state model of the available services. It should be noted that this correlation is not followed by all approaches in this category.

Logic-Based Planning

A large number of AI planning techniques involves expressing a domain theory in classical logic. A set of rules is defined and plans are derived based on these rules. In [6], the authors extend composition rules to include new constraints known as invariants, in order to capture the knowledge that a property must not change from a state to another. The extended composition rules are used in a backward planning approach that can reason about service compositionality.

Another AI planning technique in the field of logic is planning with situation calculus. The situation calculus represents the world and its change as a sequence of situations, where each situation is a term that represents a state and is obtained by executing an action. McIlraith and Son [53] adapt and extend Golog, a logic programming language built using situation calculus in order to allow customizable user constraints, nondeterministic choices and other extensions to better satisfy the requirements of service composition. In [67], the authors present a set of methods written in the logic programming language Prolog, for translating OWL-S service descriptions into Golog programs.

Other Planning Techniques

Despite the advantages of the aforementioned logic-based approaches, such as precise semantics and the ability to prove properties of domain theories, the AI planning community developed several formalisms to express planning domains. Planning Domain Definition Language (PDDL) [30] is widely recognized as a standardized input for state-of-the-art planners. In [65], WSDL descriptions along with service-annotation elements are transformed to PDDL documents. The framework allows for replanning when additional knowledge has been acquired.

Graph-based planners have also been used in Web service composition approaches such as the one presented by Lecue et al. [44]. Their framework includes an Automatic Composition Engine that can take as input both natural language and formalized requests. The suggested framework uses the Causal Link Matrix (CLM) formalism [45], extended to support non-functional properties, in order to facilitate the computation of the final service composition as a semantic graph.

Hierarchical Task Network (HTN) planning [25, 26], which provides hierarchical abstraction to deal with the complexity of real-world planning domains has also been exploited to facilitate Web service composition [38]. Finally, some planning techniques are based on model checking, a formal verification technique which allows determining whether a property holds in a certain system formalized as a finite state model, i.e., using a Finite State Machine (FSM). Pistore et al. [68] present such a composition framework that covers both cases of extended goals and cases of partial observability.

Comparison

The research approaches presented in this section are summarized and compared in Table 3.4. The table examines if there are approaches in each category that support the the following criteria:

- Domain independence: The method is not exclusive to a specific domain (e.g., travelling, automotive) but can be applied to any, allowing for the solution of a broad range of problems.
- Partial observability: The method is able to reason on incomplete information.
- Non-determinism: The method deals with actions that may lead to different states depending on the values of some parameters (e.g., an if-else construct).
- Scalability: The method is able to solve real-world composition problems which often deal with a large number of services. High scalability ensures that there is support for large numbers of services and/or high levels of complexity.
- Applicability: The method uses well-known and widely used standards (mentioned in the table).

3.4 Summary

In S-Cube, service compositions are the middle layer in the layered structure of service-based applications and have a crucial importance in developing and executing such applications. In this chapter we have presented the state-of-the-art in the field of service compositions in a structured manner, although the classification of approaches and related solutions may not give the full overview of existing research results. We presented solutions to three different approaches to developing service compositions, namely the model-driven, QoS-aware and automated service composition. In addition, we presented existing well-established languages for service orchestration, choreography, coordination and assembly. The material in this book chapter can serve as a basis for homogenizing the existing approaches for service composition and improving the existing solutions.

Table 3.4. Comparison of Automated Service Composition Methods

Automated Composition Method	Domain Independence	Partial Observability	Non-Determinism	Scalability	Applicability
Workflow	Yes	No	Yes	Average	BPEL
Logic-based pl/ng Rule-based	Yes	No	No	Varies	OWL-S
Logic-based pl/ng Situation calculus	Yes	Yes	Yes	Average	OWL-S PDDL
Logic-based pl/ng Event calculus	Yes	No	No	Low	OWL-S
PDDL planning	No	Yes	No	Good	PDDL
Graph-based planning	No	Yes	Yes	Average	BPEL
HTN planning	Varies	Yes	Yes	Good	OWLS PDDL
Model-checking and FSM	Yes	Yes	Yes	Varies	BPEL OWL-S

References

1. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: BPEL for semantic web services (BPEL4SWS). In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2007, Part I. LNCS, vol. 4805, pp. 179–188. Springer, Heidelberg (2007)
2. Alonso, G., Dadam, P., Rosemann, M. (eds.): BPM 2007. LNCS, vol. 4714. Springer, Heidelberg (2007)
3. Web Services Business Activity Framework (WS-BusinessActivity). Version 1.1 (April 2007), http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os.pdf
4. Web Services Coordination (WS-Coordination) Version 1.1 (April 2007), http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-os.pdf
5. Akkiraju, R., Verma, K., Goodwin, R., Doshi, P., Lee, J.: Executing Abstract Web Process Flows. In: Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services of the ICAPS '04 Conference, Whistler, British Columbia, Canada (2004)
6. Alevizou, V., Plexousakis, D.: Enhanced Specifications for Web Service Composition. In: ECOWS '06: Proceedings of the European Conference on Web Services, Zurich, Switzerland, pp. 223–232. IEEE Computer Society (2006)
7. Arkin, A.: Business Process Modeling Language (BPML) (November 2002)
8. Balzer, S., Liebig, T., Wagner, M.: Pitfalls of OWL-S – A Practical Semantic Web Use Case. In: ICSOC' 04: Proceedings of the 2nd International Conference on Service Oriented Computing, New York, NY, USA, November 2004, pp. 289–298 (2004)
9. Barros, A., Dumas, M., Oaks, P.: Standards for Web Service Choreography and Orchestration: Status and Perspectives. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 61–74. Springer, Heidelberg (2006)

10. Barros, A., Dumas, M., ter Hofstede, A.H.M.: Service Interaction Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)
11. Benatallah, B., Dumas, M., Maamar, Z.: Definition and Execution of Composite Web Services: The SELF-SERV Project. IEEE Data Eng. Bull. 25(4), 47–52 (2002)
12. Börger, E.: High Level System Design and Analysis Using Abstract State Machines. In: Hutter, D., Traverso, P. (eds.) FM-Trends 1998. LNCS, vol. 1641, pp. 1–43. Springer, Heidelberg (1999)
13. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: WWW, pp. 403–410 (2003)
14. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Towards a formal framework for choreography. In: WETICE, pp. 107–112. IEEE Computer Society (2005)
15. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: QoS-Aware Replanning of Composite Web Services. In: Proceedings of the IEEE International Conference on Web Services (ICWS'05), Orlando, FL, USA, pp. 121–129 (2005)
16. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of Service for Workflows and Web Service Processes. Journal of Web Semantics 1(3), 281–308 (2004)
17. Carman, M., Serafini, L., Traverso, P.: Web Service Composition as Planning. In: Workshop on Planning for Web Services in ICAPS'03, Trento, Italy, AAAI (2003)
18. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing 6(2), 86–93 (2002)
19. Curbera, F., Ferguson, D.F., Nally, M., Stockton, M.L.: Toward a programming model for service-oriented computing. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 33–47. Springer, Heidelberg (2005)
20. Decker, G.: Realizability of interaction models. In: ZEUS. CEUR Workshop Proceedings, vol. 438, pp. 55–60. CEUR-WS.org (2009)
21. Decker, G., Barros, A.: Interaction modeling using BPMN. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 208–219. Springer, Heidelberg (2008)
22. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS, pp. 296–303. IEEE Computer Society (2007)
23. Decker, G., Overdick, H., Zaha, J.M.: On the suitability of ws-cdl for choreography modeling. In: Weske, M., Nüttgens, M. (eds.) EMISA. LNI, vol. 95, pp. 21–33, GI (2006)
24. Decker, G., Weske, M.: Local enforceability in interaction petri nets. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 305–319. Springer, Heidelberg (2007)
25. Erol, K., Hendler, J., Nau, D.S.: Semantics for HTN Planning. Technical Report CS-TR-3239, UM Computer Science Department (1994)
26. Erol, K., Hendler, J.A., Nau, D.S.: UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In: AIPS' 94: 2nd International Conference on AI Planning Systems, Chicago, IL, USA, pp. 249–254. Morgan Kaufmann, San Francisco (1994)

27. Farrell, J., Lausen, H.: Semantic Annotations for WSDL and XML Schema. W3C Member Submission (2007)
28. Fu, X., Bultan, T., Su, J.: Realizability of conversation protocols with message contents. Int. J. Web Service Res. 2(4), 68–93 (2005)
29. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An Approach for QoS-aware Service Composition based on Genetic Algorithms. In: Proceedings of the Genetic and Computation Conference (GECCO'05), Washington DC, USA, ACM Press, New York (2005)
30. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL – The Planning Domain Definition Language. Technical Report CVC TR–98–003/DCS TR–1165, Yale Center for Computational Vision and Control, Version 1.2 (1998)
31. Gronmo, R., Jaeger, M.C.: Model-Driven Semantic Web Service Composition. In: APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference, Washington, DC, USA, pp. 79–86. IEEE Computer Society (2005)
32. Hauser, R., Koehler, J.: Compiling Process Graphs into Executable Code. In: Karsai, G., Visser, E. (eds.) GPCE 2004. LNCS, vol. 3286, pp. 317–336. Springer, Heidelberg (2004)
33. Huang, Y., Wang, H.: A petri net semantics for web service choreography. In: SAC, pp. 1689–1690. ACM (2007)
34. Huhns, M.N., Singh, M.P.: Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing 9(1), 75–81 (2005)
35. Jaeger, M.C., Mühl, G., Golze, S.: QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 646–661. Springer, Heidelberg (2005)
36. Jaeger, M.C., Rojec-Goldmann, G., Mühl, G.: QoS Aggregation for Service Composition using Workflow Patterns. In: Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC'04), Monterey, California, USA, September 2004, pp. 149–159. IEEE Computer Society Press, Los Alamitos (2004)
37. Jaeger, M.C., Rojec-Goldmann, G., Mühl, G.: QoS Aggregation in Web Service Compositions. In: Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05), Hong Kong, China, March 2005, IEEE Press, Los Alamitos (2005)
38. Jianhong, Z., Shensheng, Z., Jian, C., Yujie, M.: Improved HTN Planning Approach for Service Composition. In: SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing, Shanghai, China, pp. 609–612. IEEE Computer Society (2004)
39. Khalaf, R., Leymann, F.: On web services aggregation. In: Benatallah, B., Shan, M.-C. (eds.) TES 2003. LNCS, vol. 2819, pp. 1–13. Springer, Heidelberg (2003)
40. Koehler, J., Hauser, R., Hauser, R., Kapoor, S., Wu, F.Y., Kumaran, S.: A Model-Driven Transformation Method. In: EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing, Washington, DC, USA, p. 186. IEEE Computer Society (2003)
41. Kopp, O., Leymann, F.: Choreography design using WS-BPEL. IEEE Data Eng. Bull. 31(3), 31–34 (2008)
42. Korhonen, J., Pajunen, L., Puustjärvi, J.: Automatic Composition of Web Service Workflows Using a Semantic Agent. In: WI '03: Proceedings of the 2003

IEEE/WIC International Conference on Web Intelligence, Halifax, Canada, p. 566. IEEE Computer Society,

43. Lausen, H., Polleres, A., Roman, D.: Web Service Modelling Ontology (WSMO). W3C Member Submission (2005)
44. Lécué, F., Silva, E., Pires, L.F.: A Framework for Dynamic Web Services Composition. In: Pautasso, C., Gschwind, T. (eds.) WEWST07: Proceedings of the 2nd ECOWS Workshop on Emerging Web Services Technology, Halle, Germany, vol. 313, CEUR (2007)
45. Lécué, F., Léger, A.: A formal model for semantic web service composition. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 385–398. Springer, Heidelberg (2006)
46. Leymann, F.: Web services flow language (wsfl 1.0). Technical report, IBM Corporation (May 2001)
47. Leymann, F., Pottinger, S.: Rethinking the Coordination Models of WS-Coordination and WS-CF. In: Third IEEE European Conference on Web Services (ECOWS 2005), November 2005, pp. 160–169. IEEE Computer Society (2005)
48. Liu, Y., Ngu, A.H.H., Zeng, L.: QoS Computation and Policing in Dynamic Web Service Selection. In: Proceedings of the 13th International Conference on World Wide Web, WWW'04 (2004)
49. Luo, N., Yan, J., Liu, M.: Towards efficient verification for process composition of semantic web services. In: Proceedings of the 2007 IEEE International Conference on Services Computing (SCC 2007), Salt Lake City, Utah, USA, 9-13 July 2007, pp. 220–227 (2007)
50. Majithia, S., Walker, D.W., Gray, W.A.: A Framework for Automated Service Composition in Service-Oriented Architectures. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 269–283. Springer, Heidelberg (2004)
51. Mancioppi, M., Carro, M., van den Heuvel, W.-J., Papazoglou, M.P.: Sound multi-party business protocols for service networks. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 302–316. Springer, Heidelberg (2008)
52. Martens, A., Moser, S., Gerhardt, A., Funk, K.: Analyzing compatibility of bpel processes. In: AICT/ICIW, p. 147. IEEE Computer Society (2006)
53. McIlraith, S.A., Son, T.C.: Adapting Golog for Composition of Semantic Web Services. In: Fensel, D., Giunchiglia, F., McGuinness, D.L., Williams, M.-A. (eds.) KR, pp. 482–496. Morgan Kaufmann, San Francisco (2002)
54. Mendling, J., Hafner, M.: From inter-organizational workflows to process execution: Generating BPEL from WS-CDL. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 506–515. Springer, Heidelberg (2005)
55. Mitra, T.: Business-driven Development (2005), http://www-128.ibm.com/developerworks/webservices/library/ws-bdd/index.html
56. Montangero, C., Semini, L.: A logical view of choreography. In: Ciancarini, P., Wiklicky, H. (eds.) COORDINATION 2006. LNCS, vol. 4038, pp. 179–193. Springer, Heidelberg (2006)
57. Mukhija, A., Dingwall-Smith, A., Rosenblum, D.S.: QoS-Aware Service Composition in Dino. In: Proceedings of the Fifth European Conference on Web Services (ECOWS'05), Halle (Saale), November 2007, pp. 3–12 (2007)

58. Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: BPEL^{light}. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 214–229. Springer, Heidelberg (2007)
59. OMG: Business Process Modeling Notation Version 1.1. OMG Recommendation, OMG (February 2008), http://www.bpmn.org/Documents/BPMN%201-1%20Specification.pdf
60. Web Services Business Process Execution Language Version 2.0 – OASIS Standard. Technical report, Organization for the Advancement of Structured Information Standards (OASIS) (Mar. 2007)
61. Orriëns, B., Yang, J., Papazoglou, M.P.: Model Driven Service Composition. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 75–90. Springer, Heidelberg (2003)
62. Ouyang, C., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Translating BPMN to BPEL. Technical Report BPM-06-02, BPM Center Report (2006)
63. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. Computer 40(11), 38–45 (2007)
64. Pautasso, C., Alonso, G.: The jopera visual composition language. Journal of Visual Languages and Computing (JVLC) 16, 119–152 (2005)
65. Peer, J.: A PDDL based tool for automatic web service composition. In: Ohlbach, H.J., Schaffert, S. (eds.) PPSWR 2004. LNCS, vol. 3208, pp. 149–163. Springer, Heidelberg (2004)
66. Pfadenhauer, K., Kittl, B., Dustdar, S.: Challenges and Solutions for Model Driven Web Service Composition. In: WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, Washington, DC, USA, pp. 126–134. IEEE Computer Society (2005)
67. Phan, M., Hattori, F.: Automatic Web Service Composition Using ConGolog. In: ICDCS Workshops, IEEE Computer Society (2006)
68. Pistore, M., Barbon, F., Bertoli, P.G., Shaparau, D., Traverso, P.: Planning and monitoring web service composition. In: Bussler, C.J., Fensel, D. (eds.) AIMSA 2004. LNCS (LNAI), vol. 3192, pp. 106–115. Springer, Heidelberg (2004)
69. Qiu, Z., Zhao, X., Cai, C., Yang, H.: Towards the theoretical foundation of choreography. In: WWW, pp. 973–982. ACM (2007)
70. Rao, J., Su, X.: A survey of automated web service composition methods. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005)
71. Rosenberg, F., Leitner, P., Michlmayr, A., Celikovic, P., Dustdar, S.: Towards composition as a service - a quality of service driven approach. In: Proceedings of the 25th International Conference on Data Engineering (ICDE2010), March 2009, pp. 1733–1740 (2009)
72. Rosenberg, F., Platzer, C., Dustdar, S.: Bootstrapping Performance and Dependability Attributes of Web Services. In: Proceedings of the IEEE International Conference on Web Services (ICWS'06), Chicago, USA, IEEE Computer Society (2006)
73. Thatte, S.: XLANG - Web Services for Business Process Design. Microsoft Corporation (2001)

74. Skogan, D., Gronmo, R., Solheim, I.: Web Service Composition in UML. In: EDOC '04: Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International, pp. 47–57. IEEE Computer Society (2004)
75. Song, X., Dou, W., Song, W.: A workflow framework for intelligent service composition. In: Workshops at the Grid and Pervasive Computing Conference, 0:11–18 (2009)
76. Su, J., Bultan, T., Fu, X., Zhao, X.: Towards a theory of web service choreographies. In: WS-FM, pp. 1–16 (2007)
77. Sycara, K., et al.: OWL-S 1.0 Release. OWL-S Coalition (2003), http://www.daml.org/services/owl-s/1.0/
78. van der Aalst, W.M.P., Mooij, A.J., Stahl, C., Wolf, K.: Service interaction: Patterns, formalization, and analysis. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM. LNCS, vol. 5569, pp. 42–88. Springer, Heidelberg (2009)
79. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(3), 5–51 (2003)
80. W3C. Web Services Choreography Description Language Version 1.0. Candidate Recommendation, W3C (November 2005), http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/
81. Wiesemann, W., Hochreiter, R., Kuhn, D.: A Stochastic Programming Approach for QoS-Aware Service Composition. In: Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08), Lyon, France (May 2008)
82. WS-AtomicTransaction Working Committee. Web Services Atomic Transaction (WS-AtomicTransaction). Version 1.1. OASIS Specification (April 2007)
83. WS-CF Working Comittee. Web Services Coordination Framework Specification (WS-CF) Version 1.0. OASIS Specification (December 2004)
84. Yang, H., Zhao, X., Qiu, Z., Pu, G., Wang, S.: A formal model for web service choreography description language. In: ICWS, pp. 893–894. IEEE Computer Society (2006)
85. Yu, T., Zhang, Y., Lin, K.-J.: Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. ACM Trans. Web 1(1), 6 (2007)
86. Meersman, R., Tari, Z.: Let's dance: A language for service behavior modeling. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 145–162. Springer, Heidelberg (2006)
87. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality Driven Web Services Composition. In: Proceedings of the 12th International Conference on World Wide Web (WWW'03), New York, NY, USA, pp. 411–421. ACM Press, New York (2003)
88. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)

4

Architectures & Infrastructure

Françoise André[1], Ivona Brandic[2], Erwan Daubert[1], Guillaume Gauvrit[1], Maurizio Giordano[3], Gabor Kecskemeti[4], Attila Kertész[4], Claudia Di Napoli[3], Zsolt Nemeth[4], Jean-Louis Pazat[1], Harald Psaier[2], Wolfgang Renz[5], and Jan Sudeikat[5,6]

[1] Institut National de Recherche en Informatique et Automatique (INRIA), France
[2] Technische Universität Wien, Vienna, Austria
[3] Consiglio Nazionale delle Ricerche (CNR), Naples, Italy
[4] MTA Computer & Automation Research Institute (MTA-SZTAKI), Budapest, Hungary
[5] Multimedia Systems Lab. (MMLab), Hamburg University of Applied Sciences, Germany
[6] Department of Informatics, University of Hamburg, Germany

Chapter Overview. The third of the S-Cube technology layers provides infrastructure capabilities for defining basic communication patterns and interactions involving as well as providing facilities for providing, for example, contextual and qualitative information about a service's and their client's environment and performance. Providing these capabilities to other layers allows service developers to use contextual information when building service based systems and provide cross layer and pro-active monitoring and adaptation of services (see research challenges). This chapter provides an overview of service infrastructures for the adaptation, monitoring and management of services which will provide these functions and concludes with a discussion of more detailed research challenges in the context of service infrastructures and their management.

4.1 Introduction

A high-level, logical view of an infrastructure is shown in Figure 4.1. This picture illustrates all the relevant concepts of the architecture for the execution of service oriented applications. In most cases, this run-time architecture should itself be service-oriented and assume the run-time mechanisms and components are realized as services and are exposed on a communication backbone. This view guarantees that the run-time mechanisms can be integrated and exploited in a synergistic way, at least at the conceptual level.

In Figure 4.1, we distinguish between core services and application-specific services. The core services are middleware services that the run-time architecture provides to all SBA in order to support the different aspects of the SBA

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 85–116, 2010.

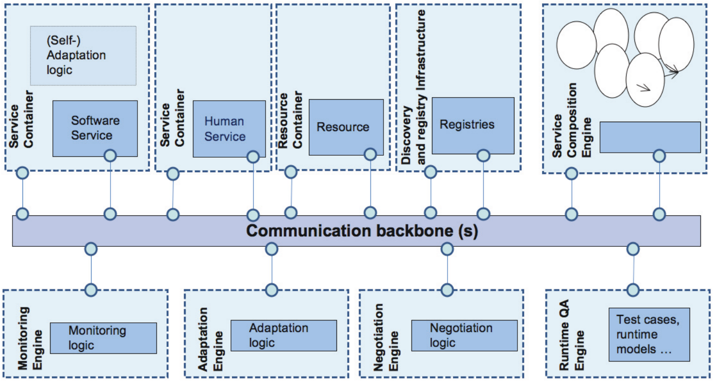


Fig. 4.1. Logical Infrastructure View

execution. Examples of such core services are a discovery service, an engine for executing service compositions or an engine for monitoring the behavior of a SBA or the performance of a business network. Some of these core services act as containers for application-specific services, i.e., services that are specific of the SBA in execution, and that encapsulate part of the application-specific logic. This is the case of the engine for executing service compositions. The core services acting as containers for application-specific services are also referred to as *service containers*.

Other core services contain other parts of the application-specific logic (which is not exposed as a service). This is the case, for instance, of the monitoring engine, which contains the application-specific logic and properties that describe what should be monitored and how. For lack of a more precise name, this type of core services will be referred to in this chapter as *engines*. Finally, there are core services that are *generic*. This is the case, for instance, of discovery services. When a SBA is deployed, specific pieces of the application logics will hence be deployed in the service containers and in the engines — depending on whether these pieces of application logics are themselves services or not — but no application-specific code is deployed in generic core services.

The communication backbone supports the communication among any kind of services, regardless of whether they are core services or application-specific services. In particular, in the case of service containers, the communication backbone allows accessing both the core service and the application-specific services deployed within the container. The core service will offer a management interface for controlling the behavior of the container and in particular the deployment and operation of application-specific services. The application-specific services expose interfaces that allow access to application-specific service functions.

4.2 Service Infrastructures for Adaptation, Monitoring & Management of Services

4.2.1 Introduction

Recently, computing infrastructures have grown in size and manage multiple interconnections to cover all requirements of functionality, coordination, and cooperation. An example are corporate infrastructures that provide the means to use, combine and exploit various communication, collaboration, and productivity tools. The situation becomes more complex the larger the infrastructure grows and the number of users and applications to be catered for increases. IBM describes the dilemmas, problems and open issues in operating such infrastructures today [35]:

- The huge amount of budget spent on preventing or recovering from system crashes.
- The effort spent in keeping a deployed system running.
- About 40% of the system outages are caused by an operator error.
- The costs provoked by long system downtimes.
- The complexity of todays applications and the related extended requirements on testing.

The current situation in providing a multipurpose computing, storage and communication infrastructure is that users want to use the infrastructure according to their individual ideas and requirements. As a result, some of their requests are difficult to fulfill because of the involved dependencies between, e.g., applications and hardware resources. This becomes especially challenging if these dependencies interfere with the requirements of other users of the same infrastructure. As an example users could lock each-others requests when accessing the same infrastructure resources. It could also occur that users request the same resources even if there is enough alternative resources but they are not aware of the fact. As a result, there is a need for the monitoring, management and adaptation of the infrastructure to balance these requirements, hide the details from the users and give sound and timely responses to their requests.

The issues described above are substantial research challenges. As mentioned, the result of the complexity in an infrastructure's components, dependencies and applications can causes unreliable and unpredictable behavior in the system. However, these problem cannot be solved by human support alone as human administrators can become overwhelmed with the task of maintaining complex systems. Even highly-sophisticated on-line monitoring and analysis instruments cannot assure that the humans using them react in time and correctly to malfunctions in the infrastructure. Thus, the main consensus of the research is that current and future infrastructures require some means of autonomy and ability to self-manage.

The research challenges in providing such an infrastructure are manifold as it is not only difficult to design and implement self-managing systems, but also to integrate such a system with existing systems. The existing, production system is usually already build and running and may have been designed to function as a stand-alone unit with few interfaces that allow it to be managed and adapted. Thus integrating observation and adaptation are crucial to the success of any self-management technique.

4.2.2 Self-Adaptation

Self-adaptation can be realised at several levels in the system being considered: the operating system level, application level or in middleware.

At the operating system level, adaptations should be transparent to the applications using it. So applications developers do not need to rewrite their applications to cope with the variations of the environment but the adaptation is not very accurate. Generic adaptation mechanisms are included into the operating system.

The direct application adaptation is a more specialized adaptation since it requires placing adaptation mechanisms inside each application. The advantages of this approach is that the adaptation is aware of application-specific qualities, which makes it more reactive and efficient but also makes it much more difficult for applications developers as they need to develop their code with this extra requirement in mind.

Moreover, it is impossible to coherently manage multiple applications. The adaptation by middleware, also called Applications-aware adaptation is a compromise between the two other solutions. This adaptation can use the specific aspect of each application and manage multiple simultaneous executions of different applications.

4.2.3 Self-Management

Self-management refers to a system component closely aligned with or integrated into a system. This binding must allow self-managing applications to observe the system in a timely and careful manner through the interfaces provided. The observations are then analyzed and possible adaptation requirements for the current system may be discovered. These requirements can be identified according to different criteria, including system degradation, the need to optimize or support new configurations or respond to security threats. When an adaptation requirement is met, a self-managing implementation can take decisions regarding which changes are to be made in current system configuration and plan the timing of those changes to minimise system downtime.

The preconditions for a self-managing component include the knowledge of itself, i.e., that the component is aware of its functionality, the resulting capabilities and possible combinations of new capabilities. This last ability already hints that a self-managing design needs to be extensible and self-adaptable.

Arriving conditions might require the design to reconfigure its internals on the fly. For example, acceptable threshold values can vary related to daytime. In other cases such a threshold value could represent the result of monitoring data and fluctuate constantly. Actions depend on the observed and, thus, a range of possible combinations must be permitted to the self-management component. Apart from the knowledge of its own capabilities self-management must also maintain a clear picture of the surroundings. Once the interfaces with the observed system are identified, the self-management component learns and adopts the system's capabilities of adaptation. It is essential to furnish the self-management with all the details about the correct operation modes. This gives the component a model to relate the current observed data and detect extraordinary events indicating deviations from normal. However, together with new requirements the system changes over time. Therefore, self-management's view on the system must also adapt along with the development of the system.

To describe the outline of an infrastructure with monitoring, adaptation, and management capabilities Figure 4.2 provides a layered overview of the main parts and dependencies. Although infrastructures are multipurpose an the details of the layer combinations differ from operation and purpose, the illustrated assembly is considered general.

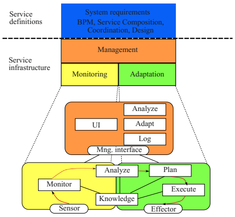


Fig. 4.2. Overview

On top, the management layer provides configuration, observation, and control facilities. Once new requirements derive from adapted process models in the BPM, polished composition, advanced coordination, and updated design, the layer provides a control interface for human operators. This supports system management by a simple interface to apply the requirements and a offline analysis of the running system's status and properties. It empowers the human administrator to define high level requirements at the management console and deploy them to the system. Thereby, the administrator is relieved from the deployment details. Generally, the management layer is supported by monitoring and adaptation layers. These two and their internal functions enhance the infrastructure with self-management. They assure that management can provide a comprehensible and reliable representation of the system's state. It is essential for the design of a self property enhanced infrastructure that the capabilities of self-management are understood and do not interfere with management decisions.

4.2.4 Monitoring Infrastructure

Monitoring (lat. monere: to remind, to warn) refers to the continuous process of observing and recording the behavior of an environment affected by dynamic influences. The monitoring process is responsible for collecting and correlating data from sensors and converting them to behavioral patterns and symptoms. The process can be realized through event correlation, or simply threshold checking, as well as other methods.

In general monitoring is motivated by the unpredictable behaviors exposed by system's parts. As a result prior to the deployment of a monitoring infrastructure it is necessary to assess the relevant intersections with the system for a precise positioning of the sensors. These are usually identified by experience gathered from historical observations that identify typical locations of unpredictable behavior in the system. An example are all kind of sensors that observe a service environment behavior. These could include hardware sensors, such as sensors for temperature, hardware load, etc., sensors for applications, e.g., aspect-oriented implementations, and communication channels sensors. The result of a deployed monitoring infrastructure is timely event information about status changes in the critical locations of the system. In the final step it lies in the responsibility of the monitoring component to decide which events to filter and provide to management and analysis. There might be different representations of the event expected by the two other components. Furthermore, monitoring is also expected to run its own analysis on the received events. Current events are then compared with and related to past events.

A combination of the new and related events can result in a composite event with information which allows for a better conclusion on the current system states. Figure 4.3 illustrates the interfaces and data-flow of a monitoring component in an infrastructure.

The figure shows at the bottom the infrastructure with some common resources available in a system. An essential prerequisite for monitoring is a reliable interface to sensing facilities. Sensors are placed at important infrastructure building blocks such as hosts, applications, communication channels, etc., that can provide reliable environment status information.

Sensor Interface

Sensor management is established by the sensor interface. This provides control over the functionality and fine-tuning of the sensors. As the system runs through changes during the course of time and the environment changes its characteristics it might be necessary to also adapt the sensors to the new conditions. Sensors might react sensitive to the changes and a calibration is necessary to avoid false alarms. Most of the time, however, the sensor interface's function is to provide the sensor data for processing by the monitoring process.

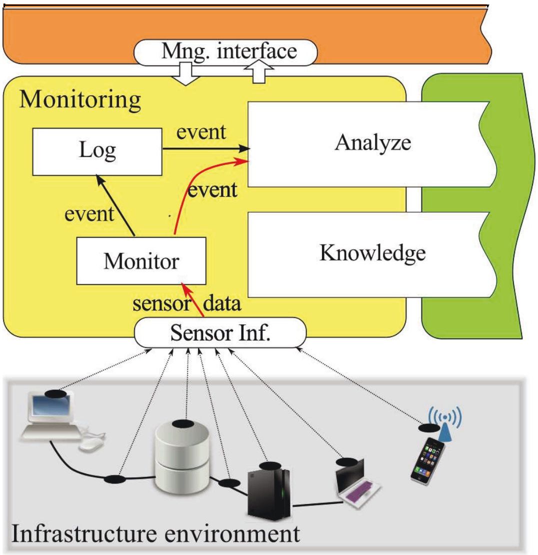


Fig. 4.3. Monitoring an Infrastructure

Monitor

The central process of the monitoring component gathers all the sensor data and turns it into an event. Thereby, it usually extends the event with accounting data, such as, time-stamp of arrival. The event is logged and forwarded to the analysis process. The monitoring process also provides the means of adapting the sensors through the sensor interface. On a recommendation by analysis, the system knowledge, or the management layer, monitoring calibrates the sensors to the new requirements.

Log

The logging process comprises the list of historical events gathered by the system. This event log is essential in estimating the root cause for any incident in the system. A interface to the monitoring process adds the new events to the database. Another interface to the analysis must provide a convenient access to the historical data.

Analysis

The analysis process is split between monitoring and adaptation component. The configuration of the analysis adjusts the monitoring intentions. The main function of analysis in monitoring is to filter the events according to the provided configuration and adjust their presentation to match the expectations from adaptation and management. At the more analysis uses event logs to create high level composite events. These are events only observed and measured over a time interval. Finally, analysis shares the collected experience to the knowledge component of the system. These allows the processes in the other components to decide also on the knowledge gathered in the monitoring process. This is especially important for self-managing infrastructures that need to *learn* from past events to adapt to a changing situation.

Summarizing, monitoring reduces and transforms the "shower" of sensor data to essential and better processable information. This process is crucial for an adequate reaction estimation. Furthermore, the quality of the event composition and thresholds (filters) define the detection capabilities of the monitoring process. An accurate filter must be reconfigurable by the management interface but also itself adapt to the eventually changing environment conditions. Next a list of the most common monitoring techniques is provided [56]:

Log Sensors

Logging states from all parts is one of the main techniques available in many of today's systems. The challenge in monitoring logs is that logs usually don't provide a common format. Thus, a log monitor needs to implement individual sensors able to process the syntax used by the different log formats. Log sensor data arrives from different sources. Therefore, monitoring is required to filter and analyze the sensor data. There are generally two possibilities. The data can be matched against admissible thresholds, or more sophisticated, data from different sources can be combined for analysis. This particular analysis allows to detect state changes that affect composed parts of the system and gives a clearer view on the overall system state[7].

Event Sensors

A event sensor provides monitoring already with an event in a certain format. Usually the monitoring process subscribes to the event or waits for an event to happen. The advantage compared to log sensors is that the event format and contained status data is in a predefined format and, thus, easily processed by the monitoring process[8].

Protocol & Standard Sensors

The sensors of established protocols and standards generate event data from data channels that transmit information in a certain standard. Similar to the log sensors, these sensors require a configuration that matches the protocol

[7] c.f., IBM: Generic Log Adapter (GLA), Log Trace Analyzer (LTA)

[8] c.f., DMTF: Common Information Model (CIM), IBM: Common Base Events (CBE)

subtleties of the observed data. The most prominent examples are IP networks transmitting various protocols[9].

Signal Sensors

These sensors transmit data to the monitoring process in regular intervals. The absence of a signal indicates a predefined event has occurred in the sensors place. According to the meaning, a event is reported by the monitoring process to the neighbor processes. Examples are heartbeat and pulse messages from system resources.

Profiling Sensors

This type of sensor operates on the application layer and observes the execution behavior of an application. Similar to other sensors information this data has first to be gathered and analyzed over time intervals to detect possible incidents. A composite event is the result of analysis[10] .

Aspect-Oriented Programming

Aspect-oriented programming is a possibility to precisely monitor an application's behavior. However, this method is intrusive because the sensors are coupled to application code through *joinpoints*.

Management Frameworks

Complete management frameworks exist for all parts of the monitoring, adaptation, and management components. One of the best known to-date is the Java Management eXtension (JMX) that provides interfaces for many kinds of sensors, sensor data formats, extensions for monitoring, management and adaptation logics. The package also includes a management interface that provides an overview of all monitoring data and allows the configuration of attached sensors.

4.2.5 Adaptation Infrastructure

Types of Adaptation

Dynamic adaptation actions can be classified according to several criteria; if one considers what can be changed in a service or in a composition of services, five possible cases can be found:

[9] c.f., Simple Network Management Protocol (SNMP), Web-Based Enterprise Management (WBEM)

[10] c.f., JVM Tool Interface (JVMTI)

1. Parameter adaptation: the simplest, "parameter adaptation" consists in changing the value of an interface parameter, for example to modify the frame-rate of a streamed video.
2. Functional adaptation: there is also "functional adaptation" that consists of changing the code that realizes the service, without visible logical consequences for the outside (the users) of the service. For instance, one can change the bytecode of a Java service to improve its performance.
3. Behavioral adaptation: "behavioral adaptation" consists of changing the algorithm of a service which has some visible effect to the service's users.
4. Environmental adaptation: this adaptation allows the modification of the service execution environment. For example, service migration is an environmental adaptation because after this adaptation, the service will not run in the same context and its performance, security level and available resources may be different. This migration may be useful when infrastructure is overloaded through hosting too many services. If a service A is migrated from an overloaded node in the infrastructure X towards a node Y, resources allocated to the service on node X are released and can reallocated to the others services, improving their Quality of Service (QoS). The QoS of service A is also improved as it is moved on a node that is not overloaded. Migration could be useful in other cases, for example to bring a service closer to its users. There are several approaches to performing service migration. The simplest do not consider the state of a service and consist of installing and starting the service on the new location. In this case, the "binary representation" (or in some cases, the executable file) which represents the service is migrated and re-started on the new node. This method is convenient for stateless services and when the service can be stopped before migration. In the other approaches, migration also consists of the migrating the execution context of the service. This context is represented by data stored in memory and the service execution can be resumed on the new node in a transparent way for the users of the service. However, this type of migration needs to take into account the state of threads created by the service, which is a complex task [31, 62, 3].
5. Structural adaptation: "structural adaptation" is concerned with the modification of compositions of services by changing one or more links between the services in each composition.

Location of Adaptation

In addition to this distinction between adaptation actions, we can also consider the location where adaptations are performed. Execution may be done 'locally', i.e., when an adaptation strategy is executed on the same node of an infrastructure. Distributed execution occurs when the adaptation strategy is executed on different nodes in the infrastructure. For example, a structural adaptation which adds a new service SNEW between two already composed services S1 and S2, if these services reside on the same platform this is a 'local' adaptation, but if S1 is located on a node N1, S2 on a different node N2 and SNEW on a third one N3, the adaptation will be distributed on these 3 nodes, needing some cooperation and synchronization capabilities.

From these possibilities, adaptation strategies can be applied to one or more services, applications, complete environments, composed nodes, clusters, Grids or dynamic clouds.

Analysis

The purpose of adaptation infrastructure is to react to monitored changes in the system by deciding when and how to adapt services, compositions and service infrastructure according to an adaptation strategy. Therefore, adaptation infrastructure lies at the interface between the monitoring and adaptation infrastructures.

Decision Guides and Strategies

A *strategy* is a reification of a chosen configuration to apply. This configuration can equally be represented by the complete configuration to adopt or as an expression of the difference between the current configuration and the one to adopt. The latter might be seen as a high-level view of a collection of actions. However, those actions might not be directly executable.

The use of two different terms for the configuration and its reification is used to make a clear distinction between the concept and the entity. Since a configuration can be represented in multiple ways, such as by a graph representing a network configuration or by a list of parameters, multiple languages can be used to represent a configuration and thus a strategy. Which language is used is dependent from the adaptation infrastructure and might vary from one to another. A strategy represents a goal to reach and multiple means might be available to reach it, and different ways might exist to execute the strategy. The task of planning a particular strategy in executable actions is done by the planning part of the adaptation infrastructure.

To create a strategy, the analysis function uses a reasoner that follows an algorithm. Various generic algorithms are available, each with different advantages and disadvantages for different purposes. For instance, some are fast to compute a strategy but grow easily in complexity and can become hard to maintain while others are slower but can efficiently model complex systems.

Using generic algorithms enables to use the same component or code in different applications. However the adaptation goals and means vary between applications. For instance, some can seek performance while other the economy of energy. Thus, the generic algorithms have to be specialized to each application. This is done using *decision guides.*

A decision guide is a document where is expressed a logic to follow using an algorithm to make decisions of adaptation when appropriate, producing strategies. A guide is followed each time — and only when — the inner representation of the system by the analysis function is changed, since there might be a need for adaptation only when changes occur. Following a guide usually can, depending on the algorithm, change the representation of the system used by the algorithm, thus making the algorithm recursive. Changing the inner representation enables to make intermediate states, states or values to compute a strategy.

Among the different algorithms or mechanisms used by reasoners are event-condition-action rules (ECA), expert systems, utility functions and learning mechanisms. Multiple reasoners — frameworks or libraries — can be found for each of those methods, generally using different languages to express their guides.

Behavior Modeling

A self-* service is a service able to react to changes in its environment by adapting itself. To do so, a self adaptable service has to be able to decide if a change is needed and how to achieve it. A behavioral model of the service and its environment can be used to compare different configurations by estimating how they would perform relatively to the current one.

As an example to illustrate the modeling of the behavior of a system, we take an component-based application running on a computing grid. One of the components in this application is used to assign requests sent by a master component to worker components. That is, this broker component implements a master-worker pattern. A self-optimization in this application is to switch between implementations of the pattern, in order for the application to make a better use of the grid according to a high-level objective.

The objective used in this example is to maximize the execution speed of the application. It is measured by the average number of executed requests by seconds. This objective is a performance oriented one, but price or consumption oriented ones can be considered.

We consider in this example three master-workers patterns. The round-robin one distributes request to workers in circular order. This pattern is fast but doesn't take into account the dynamism of the workload on the grid. Another pattern is the load-balancing one, as implemented by NetSolve, which use a queue to ensure that every worker gets the same load to process, as long as there are more requests than workers at any time. The DIET pattern uses a modified version of the DIET framework, which uses a request sequencing policy, a distributed architecture with many agents and has probes to its disposal to estimate the workers's processing speed. It differs from the original by using a scheduler which sorts the requests by the estimated time to process the requests, when possible.

Planning

The planning step specifies the actions that can implement the adaptation strategy and schedule them. A planning algorithm is used for that purpose. It takes as input a source configuration and a target configuration that are the result of the decision step and represent the strategy. It also use a domain of actions that defines the set of all possible actions to implement the strategies. For the planning step, the actions need not be defined with too much details of implementation. One may prefer to use "abstract actions" that will only be converted as concrete ones at the execution step. For instance the planning may use the abstract action "start service" without needing to know if the service is an OSGi service or a Web service. At execution time the abstract "start service" will be converted either in "register the service on the OSGi platform" or in "register the service in the Web repository".

Preconditions and postconditions are associated with actions : they help in choosing the actions which could lead from the source configuration to the target configuration. Sometimes choices are possible between two or more actions or subset of actions to achieve the goal. Some planning algorithms are able to take into account constraints associated to actions in order to choose the actions that will the best achieve the objectives. For example constants may express the time to execute an action or the resource consumption. The objectives of the planning algorithm could be to minimize the total execution time to perform the adaptation or to minimize the resource consumption. For instance the planning language PDDL (Planning Domain Definition Language), first introduced in 1998 [36], allows to describe such constraints. Indeed since 1998 the PDDL language has evolved to take into account new constraints, for instance on time duration for actions or on resource consumption. Current version of PDDL is constituted by a basic kernel and a set of extensions for each added feature. The user of PDDL needs to choose among the extensions, those that he needs depending on the planning algorithm used.

The planning step has also to schedule the selected actions. Indeed some actions may be dependent of some others. For example, it is not possible to start a bundle on an OSGi platform if the bundle is not already installed and its dependencies resolved. At the opposite, some actions can be independent of some others, leading to a partial order between them. For example, two bundles can be started on two distinct OSGi platforms at the same time. So, if the applications that are subject for adaptation are running on a distributed architecture, it could be useful to exhibit the potential parallelism between the adaptation actions. On distributed platforms, for the actions that have to be executed in a predefined order, explicit synchronization operations have to be added.

Within a distributed architecture, the designer of the adaptation system can choose to instantiate several planners in order to distribute the planning step. This could be of interest either to achieve fault tolerance in case of a node where the planner is located breaks down, or to decompose the planning operations into parallel tasks. In that case a distributed parallel planning algorithm should be designed which probably will need some cooperation mechanisms in order to achieve a common goal without inconsistency.

Execution

If one considers that the planning phase produces a set of abstract actions, they should be first translated into concrete ones, before to be executed (concrete actions are also called effectors). Some adaptation designers may prefer not to use abstract actions as a result of the planning, in that case this step is not necessary. Abstract actions are used to hide the concrete SOA implementation. More than facilitating the task of planning, it is also particularly useful if several different SOA such as SCA, OSGi, may be used to build the adaptable applications. Each specific platform can then choose the best possible concrete actions to implement an abstract one (see Figure 4.4). In the following we will consider that the actions specified by the planner are abstract ones. The set of actions produced by the planning phase respects the schedule computed by the planning algorithm. This means that the actions are ordered by steps. A step regroups actions that could be done simultaneously. But the actions included in a step N can not be started before all the actions in step N-1 are not finished.

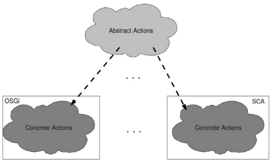


Fig. 4.4. Abstract and Concrete Actions

4.2.6 Management Infrastructure

In Figure 4.2 we can see a layered overview of the main parts of a service infrastructure. The management layer provides configuration, observation, and control facilities supported by monitoring and adaptation layers. Their main task is to reduce the complexity, since modern infrastructures tend to become incomprehensible and unreliable, therefore there is an emerging need for self-* properties.

The Management layer is best exemplified with an overview and insight of an existing solution. Therefore in this subsection we introduce, how the management layer appears and operates in an autonomic Service-level Agreement-based Service Virtualization architecture (SSV). SSV provides a way to ease service executions in a diverse, heterogeneous, distributed and virtualized world of services. The architecture consists of the combination of negotiation, brokering and deployment using SLA-aware extensions implementing autonomic computing principles for achieving reliable service operations.

Agreement negotiation, brokering and service deployment are closely related and each of them requires extended capabilities in order to interoperate smoothly. In the following we focus on illustrating how autonomic management operations appear in the components of SSV the architecture. Figure 4.5 shows the management interfaces and connections of the three main components: agreement negotiation, brokering and service deployment.

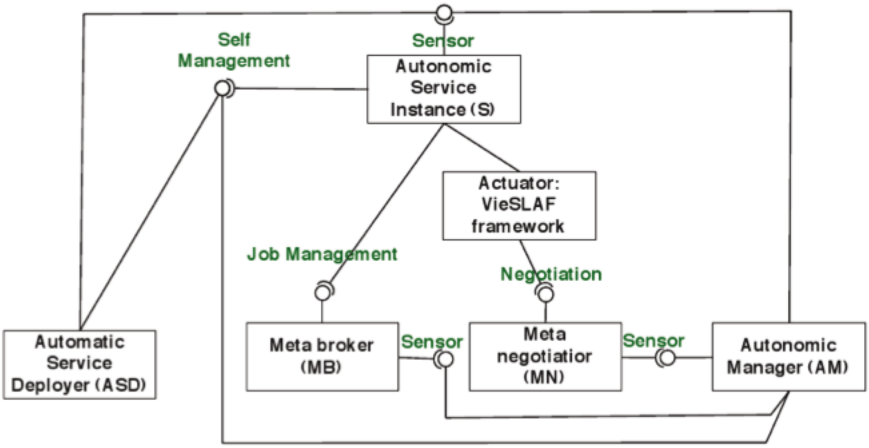


Fig. 4.5. Management Interfaces in the SSV Architecture

We distinguish three types of interfaces in this architecture: the *job management interface*, the *negotiation interface* and the *self-management interface.* Negotiation interfaces are typically used by the monitoring processes of brokers and meta-brokers during the negotiation phases of the service deployment process. Self-management is needed to re-negotiate established SLAs during service execution. The negotiation interface implements negotiation protocols, SLA specification languages, and security standards as stated in the meta-negotiation document.

Job management interfaces are necessary for the manipulation of services during execution, for example for the upload of input data, or for the download of output data, and for starting or canceling job executions. Job management interfaces are provided by the service infrastructure and are automatically utilized during the service deployment and execution processes. In the following we focus on the management interface. The *Autonomic manager* in the SSV architecture is an abstract component, that specifies how self-management is carried out. All components of the architecture is notified about the system malfunction through appropriate sensors (see Figure 4.5). This interface specifies operations for sensing changes of the desired state and for reacting to that changes. Sensors can be activated using some notification approach (e.g., implemented by the WS-Notification standard). Sensors may subscribe for specific topic, e.g., violation of the execution time. Based on the measured values (provided by the monitoring infrastructure) notifications are obtained, if execution time is violated or seems to be violated very soon. After the activation of the control loop, i.e., propagation of the sensed changes to the appropriate component, the service actuator reacts and invokes proper operations, e.g., migration of resources.

Based on various malfunction cases, the autonomic manager propagates the reactions to the *Meta negotiatiator*, *Meta-broker* or *Automatic Service Deployer*. Before service executions, the user and the provider may enter into negotiations that determine the definition and measurement of user QoS parameters, and the rewards and penalties for meeting and violating them respectively. The term negotiation strategy represents the logic used by a partner to decide which provider or consumer satisfies his needs best. A negotiation protocol represents the exchange of messages during the negotiation process. Many researchers have proposed different protocols and strategies for SLA negotiation, however, these not only assume that the parties to the negotiation understand a common protocol but also assume that they share a common perception about the goods or services under negotiation. In reality however, a participant may prefer to negotiate using certain protocols for which it has developed better strategies, over others. Thus, the parties to a negotiation may not share the same understanding that is assumed by the earlier publications in this space. In order to bridge the gap between different negotiation protocols and scenarios, SSV uses a so-called meta-negotiation architecture. *Meta-negotiation* is needed by means of a meta-negotiation document where participating parties may express: the pre-requisites to be satisfied for a negotiation, for example a specific authentication method required or terms they want to negotiate on (e.g., time, price, reliability); the negotiation protocols and document languages for the specification of SLAs that they support; and conditions for the establishment of an agreement, for example, a required third-party arbitrator. These documents are published into a searchable registry through which participants can discover suitable partners for conducting negotiations.

Management includes brokering-related aspects of the SSV architecture. Brokers are the basic components that are responsible for finding the required services with the help of an Automatic Service Deployer (ASD). This task requires various activities, such as service discovery, matchmaking and interactions with information systems, service registries, repositories. In this architecture brokers need to interact with ASDs and use adaptive mechanisms in order to fulfill agreements. A higher-level component is also responsible for management in SSV: the Meta-Broker. *Meta-brokering* means a higher level service management that utilizes existing resource or service brokers to access various services. In a more generalized way, it acts as a mediator between users or higher level tools (e.g., negotiators or workflow managers) and environment-specific resource managers. The main tasks of this component are: to *gather* static and dynamic broker properties (availability, performance, provided and deployable services, resources, and dynamic QoS properties related to service execution), to *interact* with MN to create agreements for service calls, and to *schedule* these service calls to lower level brokers, i.e., match service descriptions to broker properties (which includes broker provided services). Finally the service call needs to be *forwarded* to the selected broker. Three main tasks need to be done by the Meta-Broker: the first, namely the information gathering (which relies on the monitoring infrastructure), the second one is negotiation handling and the third one is service selection (which relies on the Adaptation infrastructure). They need the following steps: During the negotiation process the Meta-Broker interacts with the Meta-Negotiator: it receives a service request with the service description and SLA terms and looks for a deployed service reachable by some broker that is able to fulfill the specified terms. If a service is found, the SLA will be accepted and the and meta-negotiator is notified, otherwise the SLA will be rejected. If the service requirements are matched and only the terms cannot be fulfilled, it could continue the negotiation by modifying the terms and wait for user approval or further modifications. Autonomic behavior is needed by brokers basically in two cases: the first one is to survive failures of lower level components, the second is to regain healthy state after local failures. To overcome these difficulties brokers in SSV use the help of the ASD component to re-deploy some services.

Further on we discuss deployment, which is also part of management processes, and refer to *low-level services* as the ones which are used during the automation of the service deployment process. A low-level service could operate on the service instances residing on the same node where the low-level service is running. Typical low-level services are including the management, adaptation, configuration and installation services. Adaptation, configuration and installation services are offering the functionality of a given deployment step discussed in the previous paragraphs. Management services however, are usually composite of the adaptation, configuration, and installation triplet. As a bare minimum installation services should be available on each node of the service infrastructure because they let other service instances to be installed locally. It would be beneficial that all the other previously mentioned services are available on the nodes, however using the installation service it is possible to install and activate the other low-level services on-demand. Using the autonomic manager self-healing services use local adaptation strategies that are based on the management, configuration and adaptation services. For example they might use management services to suspend under-utilized service instances on a node where there is a highly demanded service. As an exception, installation services are usually not useful for self-healing. Because installations can further degrade the health state of the already deployed parts of the service, which is the situation self-healing tries to avoid. In the scope of deployment local adaptation is a simple extension of self-healing by extending the decision making policies with conditions about the service instance's context. As an example the decision making process should take into consideration the health state of the surrounding service instances that offer the same functionality. Local adaptation might exist *without self-healing* services, however it is useful to have self-healing services on the lowest level. Then one can build local adaptation on top of the self-healing capabilities. In case the local adaptation is not using self-healing capable services, then at least the *service instance level monitoring facilities* should be implemented before doing local adaptation. Service instances should offer interfaces to *share their health status* independently from an information service. The health status of a service instance does not need to be externally understood, the only requirement that other service instances, which offer the same interface should understand them. A monitoring infrastructure should be built similarly to the self-healing monitoring solutions, however the events and adaptation strategies should take into consideration the health state of the connected service instances. For example the service instance now can make decisions whether it has to prepare for an increased amount of requests. As a result it should use its management interfaces to *reconfigure itself* to make sure it will bear the future load. In case the local adaptation is offered without self healing capabilities, then the management interfaces will not be available for the service instance, therefore the locally adaptable services should decide together to use the automatic service deployment system to *deploy a new instance* which can cope with the increased needs and it could also decide to request the *decommission of a underperforming service* instance from the group.

Service instances should not build on centralized discovery mechanisms to find other instances offering the same service interface in the SBA. Service instances should have *embedded discovery mechanisms* and they should use it as a failsafe solution. For example by using peer-to-peer mechanisms the service instances can decide to *locally increase the processing power* of a given service by deploying new instances in the neighborhood without even affecting the entire SBA. This could be useful when the SBA becomes partitioned or the service instances further away cannot feasibly serve the locally increased service requests. The packages should be stored in a repository as part of the automatic service deployment (ASD) system. This repository is a package repository, and it is not a single entity in the infrastructure but replicated. Packages should be replicated among them this is one of the reason why the automatic service deployment system should be aware of the repositories. In case of new deployments, frequently used components can be replicated and also merged when the package retrieval patterns suggest – e.g., two packages are frequently downloaded together. Packages should also be stored with their configuration options, because healing strategies are usually simple maps between different situations and configuration options. Automatic service deployment (ASD) is a higher-level service management concept, which provides the dynamics to SBAs — e.g., during the SBA's lifecycle services can appear and disappear without the disruption of their overall behavior. To interface with a broker the ASD should be built on a repository. All the master copies of all the deployable services should be stored in the repository. In this context the master copy means everything what is needed in order to deploy a service on a selected site – which we call the virtual appliance (VA). The virtual appliance could be either defined by an external entity or the ASD solution should be capable of acquiring it from an already running system. The repository allows the broker to determine which services are available for deployment and which are the static ones. Thus the repository would help to define a schedule to execute a service request taking into consideration those sites where the service has been deployed and where it could be executed but has not yet been installed. If the deployed services are not available, it checks whether any of the latter resources can deliver the service taking into account the deployment cost. Regarding component interactions, the ASD needs to be extended with the following in order to communicate with brokers: Requested service constraints have to be forced independently from what Virtual Machine Monitor is used. To help the brokers making their decisions about which site should be used the ASD has to offer deployment cost metrics which can even be incorporated on higher level SLAs. The ASD might initiate service deployment/decommission on its own when it can prevent service usage peaks/lows, to do so it should be aware of the agreements made on higher levels.

In the management infrastructure of SSV, there is a bidirectional connection between the ASD and the service brokers. First the service brokers could instruct ASD to *deploy* a new service. However, deployments could also occur independently from the brokers as explained in the following. After these deployments the ASD has to *notify* the corresponding service brokers about the infrastructure changes. This notification is required, because information systems cache the state of the SBA for scalability. Thus even though a service has just been deployed on a new site, the broker will not direct service requests to the new site. This is especially needed when the deployment was initiated to avoid an SLA violation.

4.3 Future Challenges

4.3.1 Self-* Properties: Main Research Directions

The most cited cornerstone of the research on self-* properties is probably the autonomic computing initiative by IBM currently comprising several papers and research directions. One of their initial works [35] introduces the most prominent self-* properties comprised by their self-managing idea. Self-management is integrated into existing or novel complex system to mask the complexity. The goal and result is a system which becomes manageable again. According to their vision the following four self properties are required to gather a self-managing system:

- Self-configuring: This reflects the ability to readjust itself "on-the fly". The necessity for configuration emerges usually as a response to changes by installing, updating, integrating, and composing/decomposing entities.
- Self-healing: A system with this property can discover, diagnose, and react to its disruptions It can also anticipate potential problems, and accordingly take proper actions to prevent a failure.
- Self-optimizing: This property tries to maximize resource utilization to meet end-user needs. Examples of important optimization concerns are response time, throughput, utilization, and workload.
- Self-protection: A self-protecting implementation can anticipate, detect, identify, and protect itself from attacks. It has two aspects, namely defending the system against malicious attacks, and anticipating problems and taking actions to avoid them or to mitigate their effects.

Early research directions that support the research on self-* properties include fault-tolerant and self-stabilizing systems. Fault-tolerant systems handle transient and mask permanent failures in order to return to a valid state [52]. Self-stabilizing systems [26] are considered a non fault masking approach for fault-tolerant systems. These systems have two distinct properties. These are (i) the system is guaranteed to return to a legal state in a finite amount of time regardless of interferences (convergence) and (ii) once in legal state it tries to remain in the same (closure).

With overlapping intentions to the autonomic computing research the research on self-adaptive systems has evolved. According to [56] the four main self-* properties are the same for both directions. One distinction is the fact that self-adaptive systems try to state their challenges at a more general level. Most of their contributions cover higher level functionality such as the autonomous management, control, evaluation, maintenance, and organization of a whole systems. Autonomic computing also includes this areas, but extends their research also to sublayers of middleware.

Over the years the list of self-* properties, also known as self-X properties, has grown substantially [63]. Example are self-governing, self-regulation, self-correction, self-organization, self-scheduling, self-planning, self-management, self-administration, self-optimization, self-monitoring, self-adjustment, self-tuning, self-configuration, self-diagnosis of faults, self-protection, self-healing, self-recovery, self-learning, self-knowledge (awareness), self-modeling/representation, self-evolution, self-assessment of risks, etc. Most of them relate to each-other such as the original four properties and have been picked up by researchers to motivate and explain parts of their works.

As an example the research on self-healing properties for systems includes surveys [37, 54], but also various specific research works and applications. These include higher layers such as models and systems' architecture [24, 17], application layer, and large-scale agent-based systems [8, 71, 18], Web services [39] and their orchestration [7]. In the middle, self-healing ideas can be found for middleware [9], and at a lower layer self-healing designs include operating systems [61], embedded systems, networks, and hardware [38].

4.3.2 Bio-inspired Decentralized Self-Organization in Service Infrastructures

The *self-organization* of the configurations of system entities is typically regarded as an alternative approach to the construction of self-adaptive systems [56, pp. 5–23]. The self-organization concept, as known from biological, physical and social systems, describes dynamic adaptive processes among autonomous entities that give rise to structures at the macroscopic system level [53]. The integrations of these processes is an attractive design approach to distributed systems in dynamic environments, since the established structures are continuously maintained and adapted. The systematic handling of these collective processes is an active research area [53, 59] that enables system self-adaptivity by designing the collective, concurrent adjustments of system elements. In addition, this development stance inherently supports non-functional system properties, e.g., the scalability and robustness. Adaptive features do not depend on dedicated system entities, but *emerge* from entity interactions [53].

Challenges of Decentralized Software Management

The embedding of self-organizing process requires two foundational design efforts. First, system entities are required to be locally adaptable, i.e., these are able to reason about their local configuration and adjust themselves. In [56], *internal* and *external* approaches for the construction of self-adaptive software components are distinguished. Internal approach refers to intertwining element functionality and their adaptation logic. An alternative approach is the encapsulation of the adaptation logic on external computational elements. Approaches to the construction of adaptation logic are discussed in Section 4.2.5.

The second foundational design effort is to establish information flows among system elements. These perceptions are inputs to the localized adaptation and follow a *locality* principle. Adaptation elements are enabled to perceive changes in their immediate context and perceptions diminish with the logical or spatial distance between elements and/or the age of the contained information. Dedicated interaction mechanisms [59] are available to the control of the dynamics of the information transport and the attenuation of aging information.

A prominent technological foundation for the conception of self-organizing applications [59] is *agent technology* [43]. This research area provides tools and concepts to design applications by concerting the interplay of autonomous, pro-active actors that are situated in an environment. The local reasoning inside agents decides the local activities and/or changes of agent configurations. In addition, communication models and infrastructures are available to decouple agents and realize differing communication modes. These range from message-based communication to environment-mediated interactions [74].

Self-Organization in Self-Adaptive Application

Besides the recognition as an alternative development approach, the integration of concerted decentralized coordination techniques is relevant for *self-adaptive* software architectures [44], as it enables developers to relate entity adaptation to high level properties, which are established by multitudes of system elements. Development efforts benefit from top-down architectural principles, but it is necessary to plan for the collective effects among system entities [42]. Collectives of sub-systems may not per se behave effectively, when they are arbitrarily combined [55]. Unexpected self-organizing effects can arise [49], which may diminish performance and/or work against the management of superordinates. Consequently, the ability to plan for the dynamics that arise in managed collectives, e.g., managed system elements and subsystems, has been identified as a research challenge [42, 44].

Designing self-organizing dynamics addresses these challenges by providing a conceptual framework plan for the decentralized concerting of element activities. If the dynamics of information flows and local adaptation policies are well matched, the collective interplay of system elements enables the rise of self-organizing structures. These structures can be used to control the spatial or temporal correlation of the local adaptations of system entities. Two basic construction approaches exist. First, system designs can be gradually *evolved* to exhibit the intended collective dynamics, e.g., by using *evolutionary algorithms* or training *neural systems* [32]. Secondly, decentralized coordination can be *built-in* by resembling the dynamics of natural systems. In the following, we focus on the latter approach.

Implementing Self-Organization

Architectural guidelines for the construction of self-organizing applications acknowledge the significance of situating inter-operating agents, e.g., [75]. The elaboration of foundational design principles for these systems (e.g., [12]) prepares the software-technological utilization of self-organizing processes. A key challenge is the provision of architectural models that separate the application logic within agents from the coordination logic, i.e., when and how to engage in activities that are conceptually related to the coordination. Recent research explores the utilization of software engineering techniques to realize this structuring. Examples are the utilizations of software services [60], OSGi components [25], and aspect-orientation [58].

An attractive development approach is to supplement coordination to existing applications. Dedicated frameworks (e.g., [64]) provide means to externalize prescriptions of coordinating processes as well as run-time environments to their enactment. This approach allows to equip applications with self-organizing features. Consequently, the presence of a self-organizing feature is not necessarily an initial design objective in a software project, but it can be supplemented to existing applications when system tests/simulations reveal the need for decentralized, adaptive features.

In [64], a corresponding reference architecture for the concerting of agent activities via self-organization is discussed. The operating principle of this architecture is illustrated in figure 4.6 (I). On the top-layer, distributed system elements provide application functionality (Application Layer). The application in conceived as a *Multiagent* System (MAS), thus a subset of system elements is realized as autonomous, pro-active agents [43]. A subjacent *Coordination Layer* provides the Mechanisms to enact Coordination Strategies within the agent population. *Information Exchange Mechanisms* are encapsulated in *Coordination Media*, i.e., virtual coordination spaces. Media control the dynamics of information exchanges among agents and different mechanisms are encapsulated in a generic usage interface. Media are accessed by *Coordination Endpoints*, i.e., agent modules that control the engagement and responses to information exchanges.

These modules (c.f., Figure 4.6, II) separate the activities that are conceptually related to coordination, including the local adaptation of entities, from the agent model. These activities are enacted *transparently* as a background process within agents, as the endpoints are enabled to observe and manipulate the agent execution (1). Endpoint modules exchange *Coordination Information* (2) to inform each other about individual behavior adjustments via Coordination Media. The activities of the endpoints are prescribed by the coordination process (3). Underlying layers, which are omitted in Figure 4.6 provide the middleware services and execution environment for the coordinated MAS.

A recent application case study is the decentralized management of application servers. The challenge is to balance the deployments of services and the utilization of servers with fluctuating user demands. The supplementation of *honey-bee foraging*-inspired coordination model is demonstrated in [65] and in [67] the management of J2EE application servers is discussed. In the latter case, software agents use the SUN *Appserver Management EXtensions*[11] (AMX) to control the deployment of Web services.

4.3.3 Nature Inspired Models for Service Management

Service management involves the cooperation of a large number of entities to perform service selection, scheduling, deployment, enactment, coordination, (re-)configuration and others tasks in such a way that management operations fulfill their requirements and are optimal according to some criteria. Needless to say, controlling systems of complexity is far beyond the capabilities of humans, so automated machinery is necessary for ensuring these management

[11] `https://glassfish.dev.java.net/javaee5/amx/index.html`

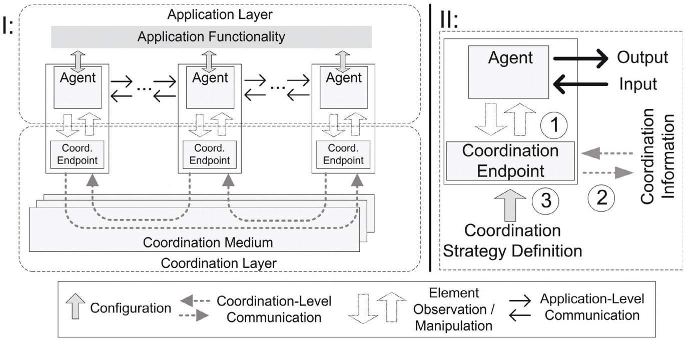


Fig. 4.6. The Enactment of Externalized, Decentralized Coordination Strategy Definitions, Following [66].

functions take place. Furthermore, the complexity of this control is increasing so that even machines cannot cope with all possible or potential cases based on predefined conditions and algorithms. This is why service management systems must exhibit some degree of autonomy, they should be able to adapt to changes and they should provide some essential self-* properties, such as self-configuration, self-optimisation, self-healing and self-protection [45] and possibly advanced self-* properties like self-adjustment, self-adaption, self-organization or self-recovery, to mention a few.

Metaphors from nature were used as inspiration for tackling complexity some decades ago when some problems reached a complexity that could not be tackled with conventional solutions. The most notable problems came from the distributed computing area, process control and artificial intelligence applications. A recent survey of nature inspired models and approaches [50] listed neural networks, ant colonies, cells, swarms, genetic algorithms, and many of their subtypes and combinations, and targeted application domains including medical imaging to wireless networks, grid, routing, data mining, mobile computing, electrical design and many others for solving scheduling, optimization, diagnosis, adaptation and security configuration.

The motivation for seeking models inspired by nature to computational problems are twofold:

- In cases where problems such as coordinating large-scale distributed systems (or even a part of one) can be formalized in a much easier and more efficient way. Most of our algorithms nowadays are sequential, expressed in imperative programmimg languages built on the notion of the von Neumann computing model. This has the effect of making the description of algorithms complex and potentially incomplete. Often nature-inspired formalisms offer a well-defined mechanism to specify the goal of the computation and not the detailed steps of the computation.

- In cases where an exact algorithm is difficult to formalize, nature metaphors may help by providing some heuristic approaches. In these cases certain nature phenomena, e.g., physical processes, chemical reactions, cells, tissues, various biological interactions, colonies, etc. are modeled and certain parameters are observed. These nature phenomena follow the laws of nature, e.g., minimum of energy, equilibrium, lower entropy, equal distribution, ideal shape, and evolve into some well defined states. By establishing an adequate relationship between the modeled nature phenomenon and the process to be controlled, certain parameters can emulate the laws of nature and converge towards a known state.

A Survey of Nature Inspired Approaches

One of the earliest nature based algorithms was *simulated annealing* (SA) [46] where the crystal structure of a cooling metal is simulated and its observed energy level is tied to certain parameters to be optimized. Simulated annealing is still a current method, e.g., in mobile networks [70] and routing problems [57].

Particle swarms are on the boundary of physical and biological systems. They model flowing particles whose position represents a solution and their velocity changes partly randomly partly depending on their position with respect to the best position so far [40]. Recent works focus on using Particle Swarm Optimization to tackle aspects of scheduling, like metaheuristics [2] or workflow scheduling [51].

Ant colonies realize probabilistic optimization by mimicking ants as finding optimal routes between their colonies and food. Individuals in the colony are extremely simple yet they are organized in a structured way that enable to accomplish complex tasks. Stigmergy (self-coordination) is achieved by marking their ways with pheromone trails and its strength signals other ants to optimize their ways [29]. The behavior of ant colonies provides models of distributed organization, optimization and routing, most notably able to solve the traveling salesman problem known to be NP-hard [28].

Artificial immune systems (AIS) exhibit self-organizing properties. The biological immunity is a reaction to foreign intrusions whereas the immunity system is a distributed adaptive system with decentralized control and using feature extraction, signaling, learning, associative retrieval. Learning and recalling self and foreign entities is adaptive so that the reaction speed and accuracy improves. Intrusion detection, anomaly and misbehavior characterization of systems are obvious candidates for applying AIS [23]. Since immune systems are especially good at classifying certain objects, application scenarios involve various image analysis, fault detection and other recognition tasks [22].

Genetic algorithms (GA) resemble natural evolution. Chromosomes, (binary strings) representing certain solutions of an optimization or search problem, reproduce and by crossover, mutation and selection better chromosomes (solutions) remain alive in each turn. Genetic algorithms are able to evolve complex structures. The most common application field of genetic algorithms is multiparameter optimization [33].

Complex biological symbiotic systems emulate the collective actions and interactions of multiple living entities. Such systems are aimed at a high level of self-organization by defining various potential behavior like reproduction, death, migration and attributes like health and energy. For instance, social insect behavior inspired methods for designing distributed control and optimization algorithms that are being applied successfully to a variety of scientific and engineering problems. Such techniques tend to exhibit a high degree of flexibility and robustness in a dynamic environment [10]. An example for such symbiotic system is SymbioticSphere [14] [16].

Different aspects of *chemical reactions* can be taken from precise simulation of interacting atoms to the abstract mathematical notion of chemical modeling. Their notion is representing the computation as reactions, i.e., data and procedures are molecules that react and yield new molecules. The central idea of chemical system is adaptation to unpredictable situations as they always take place according to actual conditions.

Outlook: Service Management and Nature Metaphors

It is anticipated that future service infrastructures will be more autonomous and possess self-* capabilities. Both the utilization of these infrastructures and their internal behavior involve many issues of optimization, coordination, adaptation that, according to the discussion and the survey above, nature inspired approaches may help to tackle. For instance, the rationale for (self-)adaptive service selection and composition is summarized as: the evolving behavior of a service (mobility, quality, faults, etc.), uninformed evolution of external services, inadequacy of pre-deployment information [48], extreme dynamicity, unreliability, and large scale [4], and a highly complex task, already beyond the human capability to deal with [27]. Also, it has been argued and generally accepted, that such self-adaptable, evolvable and context-aware systems require innovative and fundamentally novel approaches that take inspiration from nature. These approaches consider devices, data, and services uniformly as entities interacting in the same way as individuals of an ecosystem [73] and can effectively organize large numbers of unreliable and dynamically changing components (cells, molecules, individuals, etc.) into robust and adaptive structures [4].

Applying nature inspired models to service oriented systems is a quite new research area. Viroli et al. [73] designed a conceptual architecture for clarifying the concepts expressed and framing the several possible nature-inspired metaphors that could be adopted. They follow a biochemical approach where above a common environmental substrate (defining the basic "laws of nature"), individuals of different kinds interact, compete, and combine with each other, so as to serve their own individual needs as well as the sustainability and the evolvability of the overall service ecosystem. Ding et al. [27], Sun et al. [68] take the neuroendocrine-immune (NEI) system as a metaphor and create a decentralized, evolutionary, scalable, and adaptive system for Web service composition and management. Here, Web services are represented by bio-entities that are able to obtain the desirable characteristics by self-organizing, cooperating, and composing. Banâtre et al. [5], [6] use the Higher Order Chemical Language (HOCL) to model and express various issues related to services and service invocations. HOCL tries to grab the notion of chemical reactions as a computing paradigm. The aim is to realize a system that is self-organizing, i.e., able to react autonomously to changes in the environment. Csorba et al. tackle the problem of service deployment in a cloud, a notable example of service infrastructure management task. They investigate how virtual machine images can be mapped onto physical machines in a self-organizing way so that the reconfiguration improves the system performance. They apply a variation of *ant colony optimization* to achieve this goal [20], and an earlier version in [21]. The same problem of efficient service deployment has been addressed by *swarm intelligence heuristics* (ant colony optimization) where services are provided by collaborating components [19]. Canfora et al. apply genetic algorithms to assist QoS aware service composition [13]. The composition takes into consideration non-functional features, cost and time constraints and is traced back to an optimization problem where application of genetic algorithms is widespread. An adaptation framework for mobile services based on genetic algorithm is proposed by Vanrompay et al. [72]. Services must adapt (composition and deployment) to changes in the environment in a self-organizing and scalable way, the use of genetic algorithms provides an appropriate heuristic solution. Another scenario for service composition based on ant colony optimization is presented in [41]. They take into consideration composite multimedia services where quality issues are obviously strict. Ensuring such quality criteria is especially challenging due to the continuous flow, synchronization issues, dynamic characteristics and rich semantics.

The survey of existing literature and current trends shows nature-inspired solutions are good candidates for some challenges in (autonomous) service infrastructure management, like self-configuration, self-optimization, self-healing, self-protection and others. Scheduling approaches, for instance, based on nature metaphors are being investigated in several scenarios as discussed above, e.g., scheduling workflow applications in clouds [51] and Grids [1] or artificial immune-system based scheduling in multiprocessor systems [69]. It is anticipated these solutions will find their way into service based applications, where scheduling is a crucial issue in service composition. Data mining techniques are well-supported by nature based heuristics, e.g., artificial immune systems [34] and they will play an important role in future service discovery mechanisms. Many approaches address the issue of (re)configuration either as a means of adaptation, optimization or protection [47] [15]. The vast majority of nature inspired approaches focusing on optimizations, e.g., particle swarm optimization [11], ant colony optimization [30] and simulated annealing. These are likely to be adopted by any aspect of services infrastructures in the future.

4.4 Chapter Summary

This chapter has provided a review of service infrastructures for adaptation, monitoring and management of services. Providing these capabilities is essential in meeting several of the S-Cube research challenges described in Chapter 1, such as proactive monitoring and adaptation and allowing end-to-end quality provision for service-based systems. Future research challenges in this area have been presented that seek to achieve the optimal self-organization of services through self-configuration, healing, optimization and protection.

References

1. Abraham, A., Liu, H., Grosan, C., Xhafa, F.: Nature inspired meta-heuristics for grid scheduling: Single and multi-objective optimization approaches. In: Xhafa, F., Abraham, A. (eds.) Metaheuristics for Scheduling in Distributed Computing Environments, pp. 247–272. Springer, Heidelberg (2008)
2. Abraham, A., Liu, H., Zhao, M.: Particle swarm scheduling for work-flow applications in distributed computing environments. In: Metaheuristics for Scheduling in Industrial and Manufacturing Applications, pp. 327–342 (2008)
3. Artsy, Y., Finkel, R.: Designing a process migration facility: the charlotte experience. Computer 22(9), 47–56 (1989)
4. Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G.A., Ducatelle, F., Gambardella, L.M., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., Urnes, T.: Design patterns from biology for distributed computing. ACM Transactions on Autonomous and Adaptive Systems 1(1), 26–66 (2006)
5. Banâtre, J.-P., Priol, T.: Chemical programming of future service-oriented architectures. JSW 4(7), 738–746 (2009)
6. Banâtre, J.-P., Priol, T., Radenac, Y.: Service orchestration using the chemical metaphor. In: Brinkschulte, U., Givargis, T., Russo, S. (eds.) SEUS 2008. LNCS, vol. 5287, pp. 79–89. Springer, Heidelberg (2008)
7. Baresi, L., Guinea, S., Pasquale, L.: Self-healing bpel processes with dynamo and the jboss rule engine. In: ESSPE, pp. 11–20 (2007)
8. Bigus, J.P., Schlosnagle, D.A., Pilgrim III., J.R., Mills, W.N., Diao, Y.: Able: A toolkit for building multiagent autonomic systems. IBM Systems Journal 41(3), 350–371 (2002)
9. Blair, G.S., Coulson, G., Blair, L., Duran-Limon, H., Grace, P., Moreira, R., Parlavantzas, N.: Reflection, self-awareness and self-healing in openorb. In: WOSS, pp. 9–14 (2002)
10. Bonabeau, E., Dorigo, M., Theraulaz, G.: Inspiration for optimization from social insect behaviour. Nature 406, 39–42 (2000)
11. Brits, R., Engelbrecht, A.P., van den Bergh, F.: Locating multiple optima using particle swarm optimization. Applied Mathematics and Computation 189(2), 1859–1883 (2007)
12. Brueckner, S., Czap, H.: Organization, self-organization, autonomy and emergence: Status and challenges. International Transactions on Systems Science and Applications 2(1), 1–9 (2006)

13. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for qos-aware service composition based on genetic algorithms. In: GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, New York, NY, USA, pp. 1069–1075. ACM Press (2005)
14. Champrasert, P., Lee, C., Suzuki, J.: Symbioticsphere: Towards an autonomic grid network system. In: CLUSTER, pp. 1–2 (2005)
15. Champrasert, P., Suzuki, J.: A biologically-inspired autonomic architecture for self-healing data centers. In: COMPSAC (1), pp. 103–112 (2006)
16. Champrasert, P., Suzuki, J.: Symbioticsphere: A biologically-inspired autonomic architecture for self-managing network systems. In: COMPSAC (2), pp. 350–352 (2006)
17. Cheng, S.-W., Garlan, D., Schmerl, B.R., Sousa, J.P., Spitnagel, B., Steenkiste, P.: Using architectural style as a basis for system self-repair. In: WICSA, pp. 45–59 (2002)
18. Corsava, S., Getov, V.: Intelligent architecture for automatic resource allocation in computer clusters. In: IPDPS, p. 201.1 (2003)
19. Csorba, M.J., Heegaard, P.E.: Swarm intelligence heuristics for component deployment. In: EUNICE. LNCS, vol. 6164, pp. 51–64. Springer, Heidelberg (2010)
20. Csorba, M.J., Meling, H., Heegaard, P.E.: Ant system for service deployment in private and public clouds. In: BADS '10: Proceeding of the 2nd workshop on Bio-inspired algorithms for distributed systems, New York, NY, USA, pp. 19–28. ACM (2010)
21. Csorba, M.J., Meling, H., Heegaard, P.E., Herrmann, P.: Foraging for better deployment of replicated service components. In: DAIS '09: Proceedings of the 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems. LNCS, vol. 5523, pp. 87–101. Springer, Heidelberg (2009)
22. Dasgupta, D.: Advances in artificial immune systems. IEEE Computational Intelligence Magazine, 40–49 (Nov. 2006)
23. Dasgupta, D., González, F.A.: An immunity-based technique to characterize intrusions in computer networks. IEEE Trans. Evolutionary Computation 6(3), 281–291 (2002)
24. Dashofy, E.M., van der Hoek, A., Taylor, R.N.: Towards architecture-based self-healing systems. In: WOSS, pp. 21–26 (2002)
25. Devescovi, D., Di Nitto, E., Dubois, D., Mirandola, R.: Self-organization algorithms for autonomic systems in the selflet approach. In: Autonomics '07: Proceedings of the 1st international conference on Autonomic computing and communication systems, pp. 1–10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2007)
26. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643–644 (1974)
27. Ding, Y., Sun, H., Hao, K.: A bio-inspired emergent system for intelligent web service composition and management. Knowledge-Based Systems 20, 457–465 (2007)
28. Dorigo, M.: Ant algorithms solve difficult optimization problems. In: Kelemen, J., Sosík, P. (eds.) ECAL 2001. LNCS (LNAI), vol. 2159, pp. 11–22. Springer, Heidelberg (2001)

29. Dorigo, M., Bonabeau, E., Theraulaz, G.: Ant algorithms and stigmergy. Future Gener. Comput. Syst. 16(9), 851–871 (2000)
30. Dorigo, M., Di Caro, G., Gambardella, L.M.: Ant algorithms for discrete optimization. Artificial Life 5(2), 137–172 (1999)
31. Douglis, F., Ousterhout, J.: Transparent process migration: Design alternatives and the sprite implementation. Software - Practice and Experience 21, 757–785 (1991)
32. Floreano, D., Mattiussi, C.: Bio-Inspired Artificial Intelligence Theories, Methods, and Technologies. MIT Press, Cambridge (Sept. 2008)
33. Forrest, S.: Genetic algorithms. ACM Comput. Surv. 28(1), 77–80 (1996)
34. Freitas, A.A., Timmis, J.: Revisiting the foundations of artificial immune systems for data mining. IEEE Trans. Evolutionary Computation 11(4), 521–540 (2007)
35. Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. IBM Syst. J. 42(1), 5–18 (2003)
36. Ghallab, M., Ecole Nationale, Constructions Aeronautiques, Isi, C.K., Penberthy, S., Smith, D.E., Sun, Y., Weld, D.: Pddl - the planning domain definition language. Technical report (1998)
37. Ghosh, D., Sharman, R., Rao, H.R., Upadhyaya, S.: Self-healing systems - survey and synthesis. Decission Support Systems 42(4), 2164–2185 (2007)
38. Glass, M., Lukasiewycz, M., Reimann, F., Haubelt, C.D., Teich, J.: Symbolic reliability analysis of self-healing networked embedded systems. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 139–152. Springer, Heidelberg (2008)
39. Halima, R.B., Drira, K., Jmaiel, M.: A QoS-Oriented Reconfigurable Middleware for Self-Healing Web Services. In: ICWS, pp. 104–111 (2008)
40. Hinchey, M.G., Sterritt, R., Rouff, C.A.: Swarms and swarm intelligence. IEEE Computer 40(4), 111–113 (2007)
41. Hossain, M.S., Alamri, A., El-Saddik, A.: A biologically inspired framework for multimedia service management in a ubiquitous environment. Concurrency and Computation: Practice and Experience 21(11), 1450–1466 (2009)
42. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. 40(3), 1–28 (2008)
43. Jennings, N.R.: Building complex, distributed systems: the case for an agent-based approach. Comms. of the ACM 44(4), 35–41 (2001)
44. Kephart, J.O.: Research challenges of autonomic computing. In: ICSE '05: Proceedings of the 27th international conference on Software engineering, pp. 15–22. ACM Press, New York (2005)
45. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1), 41–50 (2003)
46. Kirkpatrick, S., Gelatt Jr., D., Vecchi, M.P.: Optimization by simmulated annealing. Science 220(4598), 671–680 (1983)
47. Lee, C., Suzuki, J.: An immunologically-inspired autonomic framework for self-organizing and evolvable network applications. TAAS 4(4) (2009)
48. Mei, L., Chan, W.K., Tse, T.H.: An adaptive service selection approach to service composition. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2008), IEEE Computer Society Press, Los Alamitos (2008)
49. Mogul, J.C.: Emergent (mis)behavior vs. complex software systems. Technical Report HPL-2006-2, HP Laboratories Palo Alto (2005)

50. Olariu, S., Zomaya, A.Y. (eds.): Handbook of Bioinspired Algorithms and Applications. CRC Press, Boca Raton (2005)
51. Pandey, S., Wu, L., Guru, S.M., Buyya, R.: A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: AINA, pp. 400–407 (2010)
52. Pierce, W.H.: Failure-tolerant Computer Design. Academic Press, London (1965)
53. Prokopenko, M.: Design vs. Self-organization. In: Prokopenko, M. (ed.) Advances in Applied Self-organizing Systems, pp. 3–17. Springer, London (2008)
54. Psaier, H., Dustdar, S.: A survey on self-healing systems - approaches and systems. Computing 87(1) (2010)
55. Saffre, F., Halloy, J., Shackleton, M., Deneubourg, J.L.: Self-organized service orchestration through collective differentiation. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 36(6), 1237–1246 (2006)
56. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. 4(2), 1–42 (2009)
57. Salleh, S., Sanugi, B., Jamaluddin, H., Olariu, S., Zomaya, A.Y.: Enhanced simulated annealing technique for the single-row routing problem. The Journal of Supercomputing 21(3), 285–302 (2002)
58. Seiter, L.M., Palmer, D.W., Kirschenbaum, M.: An aspect-oriented approach for modeling self-organizing emergent structures. In: SELMAS '06: Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems, pp. 59–66. ACM Press, New York (2006)
59. Serugendo, G.D.M., Gleizes, M.P., Karageorgos, A.: Self-organisation and emergence in mas: An overview. Informatica 30, 45–54 (2006)
60. Di Marzo Serugendo, G., Fitzgerald, J.: Designing and controlling trustworthy self-organising systems. Perada Magazine (2009)
61. Shapiro, M.W.: Self-healing in modern operating systems. ACM Queue 2(9), 66–75 (2005)
62. Stellner, G.: Cocheck: checkpointing and process migration for mpi. In: The 10th International Parallel Processing Symposium, 1996, Proceedings of IPPS '96, Apr. 1996, pp. 526–531 (1996)
63. Sterritt, R.: Autonomic computing. Innovations in Systems and Software Engineering 1(1), 79–88 (2005)
64. Sudeikat, J., Braubach, L., Pokahr, A., Renz, W., Lamersdorf, W.: Systematically engineering self-organizing systems: The sodekovs approach. Electronic Communications of the EASST 17 (2009)
65. Sudeikat, J., Renz, W.: MASDynamics: Toward systemic modeling of decentralized agent coordination. In: David, K., Geihs, K. (eds.) Kommunikation in Verteilten Systemen. Informatik aktuell, pp. 79–90. Springer, Heidelberg (2009)
66. Sudeikat, J., Renz, W.: Programming adaptivity by complementing agent function with agent coordination: A systemic programming model and development methodology integration. Communications of SIWN 7, 91–102 (2009)
67. Sudeikat, J., Renz, W.: Shoaling glassfishes: Enabling decentralized web service management. In: 3rd International Conference in Sef-Adaptive and Self-Organizing Systems, pp. 291–292 (short paper). IEEE Computer Society Press, Los Alamitos (2009)

68. Sun, H., Ding, Y.: A scalable method of e-service workflow emergence based on the bio-network. In: Fourth International Conference on Natural Computation (2008)
69. Swiecicka, A., Seredynski, F., Zomaya, A.Y.: Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support. IEEE Trans. Parallel Distrib. Syst. 17(3), 253–262 (2006)
70. Taheri, J., Zomaya, A.Y.: A simulated annealing approach for mobile location management. Computer Communications 30(4), 714–730 (2007)
71. Tesauro, G., Chess, D.M., Walsh, W.E., Das, R., Segal, A., Whalley, I., Kephart, J.O., White, S.R.: A multi-agent systems approach to autonomic computing. In: AAMAS, pp. 464–471 (2004)
72. Vanrompay, Y., Rigole, P., Berbers, Y.: Genetic algorithm-based optimization of service composition and deployment. In: SIPE '08: Proceedings of the 3rd international workshop on Services integration in pervasive environments, New York, NY, pp. 13–18. ACM (2008)
73. Viroli, M., Zambonelli, F.: A biochemical approach to adaptive service ecosystems. Inform. Sci. (2009)
74. Viroli, M., Holvoet, T., Ricci, A., Schelfthout, K., Zambonelli, F.: Infrastructures for the environment of multiagent systems. Autonomous Agents and Multi-Agent Systems 14(1), 49–60 (2007)
75. Weyns, D., Holvoet, T.: An architectural strategy for self-adapting systems. In: SEAMS '07: Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems, Washington, DC, USA, IEEE Computer Society (2007)

5

Adaptation of Service-Based Systems

Raman Kazhamiakin[1], Salima Benbernou[2], Luciano Baresi[3],
Pierluigi Plebani[3], Maike Uhlig[4], and Olivier Barais[5]

[1] Fondazione Bruno Kessler (FBK), Trento, Italy
[2] Université Claude Bernard Lyon 1, France
[3] Politecnico di Milano, Italy
[4] Universität Duisburg-Essen, Germany
[5] Institut National de Recherche en Informatique et Automatique (INRIA), France

Chapter Overview. The advances in modern technology development and future technology changes dictate new challenges and requirements to the engineering and provision of services and service-based systems (SBS). These services and systems should become drastically more flexible; they should be able to operate and evolve in highly dynamic environments and to adequately react to various changes in these environments. In these settings, adaptability becomes a key feature of services as it provides a way for an application to continuously change itself in order to satisfy new contextual requirements.

Events and conditions triggering application adaptation include: changes in the infrastructural layer of the application due to quality of service changes; changes of the (hybrid) application context and location; changes of the user types, preferences, and constraints that require application customization and personalization as a means to adapt the application behavior to a particular user; changes in the functionalities provided by the component services that requires modifying the way in which services are composed and coordinated; and changes in the way the service is being used and managed by its consumers, which in turn leads to changes in the application requirements.

5.1 Introduction

In the following we will exploit the vision of the adaptation and monitoring of services and service-based system continuously developed within the scope of the S-Cube project. At the high level of abstraction, the adaptation and monitoring framework can be described by the concepts represented in Figure 5.1. This figure identifies *Monitoring Mechanisms*, *Monitored Events*, *Adaptation Requirements*, *Adaptation Strategies*, *Adaptation Mechanisms*, and the relations between these concepts, as the key elements of the adaptation and monitoring framework. It is important to remark that the significance of this conceptual framework is not in the figure itself – it describes a standard sensing/planning/actuating control chain. The significance is in the very broad

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 117–156, 2010.

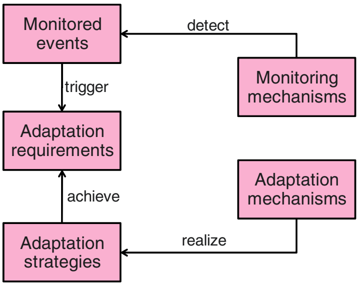


Fig. 5.1. Adaptation and Monitoring

meaning that we give to the different concepts, and to the capability of the chain to allow for a very general integration of a wide range of mechanisms, techniques and methodologies for monitoring and adaptation.

- With *Monitoring Mechanism* we mean any mechanism that can be used to check whether the actual situation corresponds to the expected one. The meaning we give to the monitoring mechanisms is very broad; in this way, we refer not only to "classical" run-time monitoring facilities, but also to techniques such as post-mortem log analysis techniques, data mining, online and offline testing and even verification/validation, etc. Realization of monitoring mechanisms is provided by the corresponding monitoring engines built on top of the monitoring infrastructures.
- Monitoring mechanisms are used to detect *Monitored Events*, i.e., the events that deliver the relevant information about the application execution, evolution, and context. These events represent the fact that there is critical difference with respect to the expected SBS state, functionality, and environment. The monitored events result from observing monitoring properties, derived from the adaptation requirements as a specification of the expected state and functionality of the SBS and its environment. The notion of monitored events may be very broad ranging from basic failures, deviation of QoS parameters, to complex properties over many executions of SBS, certain trends in the SBS environment, changes in business rules, etc.
- Monitored events in turn trigger *Adaptation Requirements*, which represent the necessity to change the underlying service or SBS in order to remove the difference between the actual (or predicted) situation and the expected one. They may include dependability and functional correctness requirements, optimality, interoperability, usability, etc.
- In order to satisfy adaptation requirements, it is necessary to define *Adaptation Strategies*, which define the possible ways to achieve those requirements given the current situation. Note that it is possible to have a set of different adaptation strategies applicable in the same situation. In this case the process requires certain decision mechanisms that operate autonomously or involve humans.

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

An important aspect of these conceptual elements is the necessity to define and implement the corresponding *decision mechanisms*, which correspond to the four arrows in the picture in Figure 5.1 and coordinate the work of the framework and realize the relations among them. In particular,

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

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

5.1.1 Aims and Focus of the Chapter

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

All these issues are often omitted by the state-of-the-art solutions both for monitoring and adaptation. In terms of the cross-cutting research challenges represented in Chapter 1 and constituting the core of the S-Cube project research agenda, the most relevant problems that the novel research results should address are

- *Cross-layer and proactive monitoring and adaptation*, as the different adaptation aspects can not be considered in isolation along the SBS life-cycle or across different functional SBS layers as it happens in most of the state of art approaches. Moreover, as we will see in the following sections, the existing adaptation approaches are mostly reactive; they aim to "recover" from the problem or to adapt to the change when it has already take place. In many settings, such a behavior may not be possible as the service invocations, process activities and executions go beyond simple software call and have certain, sometimes very critical, business value that is not always possible to revert. Proactive adaptation is the key mechanism in these settings.
- *considering contextual information for service-based systems*, since the context, and more specifically the user context, often becomes a key driver of the adaptation activities in the modern Internet of Service applications.

To understand better these phenomena, in this survey we will study and review the relevant monitoring and adaptation approaches. We will provide a comprehensive classification of the corresponding concepts, principles, techniques, and methodologies; we will identify the overlaps between various research activities and reveal the gaps and problems that the research community should address in this area. We remark that the state of art survey activities presented in this chapter as well as the adaptation taxonomy have been elaborated and developed in the scope of the S-Cube project.

The structure of the survey is organized as follows. Section 5.2 will provide a classification of the principles and concepts related to the problem of the service adaptation. Section 5.3 will provide a review of the existing works in the area. Finally, Section 5.3.3 identifies gaps and overlaps in the current research contributions on the basis of the results of the survey. Finally, Section 5.4 makes a review of the adaptation approaches in other types of information systems.

5.2 Adaptation Taxonomy

In order to provide a holistic, comprehensive, and integrated vision on the monitoring and adaptation across various research disciplines, in we will try to present a generalized and universal yet practical definition of the adaptation problem. We will present a generic conceptual model for the adaptation. Based on the conceptual model, this section will provide a classification of the adaptation concepts structured such as to answer the following questions about the corresponding concepts: "Why?", "What?" and "How?". In particular,

- the "Why?" dimension provides a description of the motivation for monitoring respectively adaptation;
- the "What?" dimension is used to classify the subject of monitoring respectively adaptation and the way it is described;
- the "How?" dimension describes the way the monitoring approach respectively adaptation approach is delivered.

5.2.1 Conceptual Model

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

High-level conceptual model of the adaptation concepts is represented in Figure 5.2.

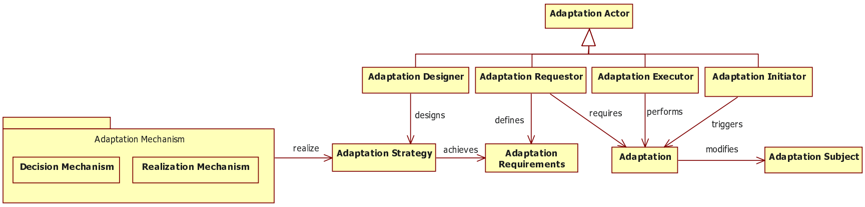


Fig. 5.2. High-Level Adaptation Model

The *Adaptation Requirements* identify the aspects of the SBS model that are subject to change, and what the expected outcome of the adaptation process is. *Adaptation Strategies* are the ways through which the adaptation requirements are satisfied. Examples of adaptation strategies are *re-configure* (i.e., modify the current configuration parameters of the SBS), *substitute* (replace one constituent service with another), *compensate* (remove the negative effect of the previously executed action by performing new actions), *re-plan* (modify the structure and the model of the application, which is more suitable for the current situation), *re-compose* (modify the way the services are composed), and *re-negotiate* (modify the service-level agreement with the service provider).

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

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

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

5.2.2 Adaptation Taxonomy

In this chapter, we describe a graphical representation of the adaptation taxonomy (depicted in Figure 5.3) that distinguishes approaches by **Why**, **Who**, **What**, and **How** software adaptation takes place.

Taxonomy Dimension: Why?

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

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

- *Perfective Adaptation*, which aims to improve the application even it runs correctly, e.g., to optimize its quality characteristics.
- *Corrective Adaptation*, which aims to remove the faulty behavior of a SBS by replacing it by a new version that provides the same functionality. Various faults can occur relatively often and unexpectedly in distributed

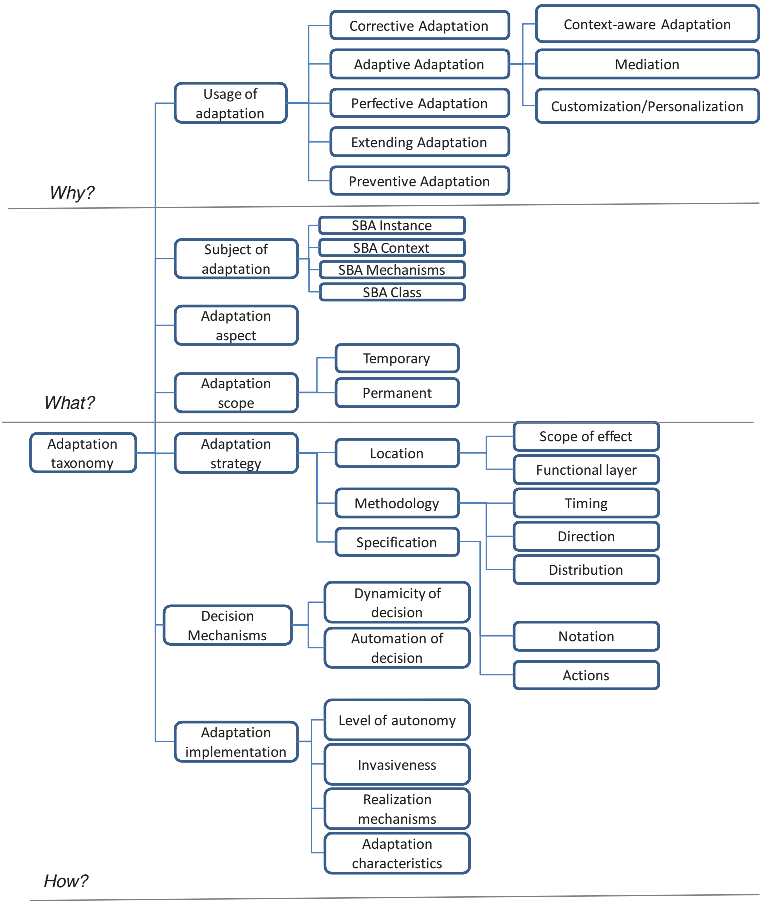


Fig. 5.3. Adaptation Taxonomy

systems. It is therefore necessary to handle failures reported during execution of the SBS in order to recover from undesired behavior, or to change the application logic in order to remove the possible fault.

- *Adaptive Adaptation*, which modifies the application in response to changes affecting its environment. The need for this kind of adaptation in SBAs is dictated by (*i*) the necessity to accommodate to the changes in the SBS context (execution context, user context, or physical context); (*ii*) the need to ensure interoperability between interacting parties by providing appropriate adapters or mediators; (*iii*) the necessity to customize or personalize the application according to the needs and requirements of particular user or customers.
- *Preventive Adaptation*, which aims to prevent future faults or extra-functional issues before they occur.

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

These classes may be further decomposed given particular problems in mind. For example, adaptive class may be divised into *context-aware adaptation*, *mediation*, and *customization/personalization*.

Taxonomy Dimension: What?

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

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

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

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

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

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

Taxonomy Dimension: How?

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

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

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

- *Scope* of adaptation effect ("horizontal" placement) says whether the changes are *local* (shallow), i.e., the small-scale incremental changes localized to a service or are restricted to the clients of that service, or whether they are *global* (deep), i.e., large-scale transformational changes cascading beyond the clients of a service possibly to entire value-chain (end-to-end processes) - clients of affected services e.g., outsourcers or suppliers.
- Affected *Functional SBS Layer* ("vertical" placement), where one can distinguish between infrastructural changes that affect service signatures, protocols, and the run-time execution environment; changes at service composition level, when the behavioral protocols and/or operational semantics of SBS are affected; level of business process management, when the change involve business rules and requirements, organizational models, clients, and even entire value chain. Finally, cross-layer changes affect different functional layers.

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

- *Timing* defines the moment of time when the adaptation is performed. *Reactive adaptation* refers to the modification in reaction to the changes already occurred; *proactive adaptation* aims to modify SBS before a deviation will occur during the actual operation and before such a deviation can lead to problems; *post-mortem* adaptation is characterized by a significant gap between the triggering event is detected and the modification performed. Typically, the post-mortem adaptation is accomplished by redesigning/re-engineering the application.
- *Direction* of the adaptation distinguishes between *forward adaptation*, where the adaptation strategy that directs the system to a new state, where the adaptation requirements are met, and *backward adaptation*, where the adaptation strategy reverts the system to a state, previously known to meet the adaptation requirements.
- *Distribution* of the adaptation distinguishes between *centralized adaptation*, where the actions are defined and executed on all the affected components in the controlled and integrated way, and *distributed adaptation* performed locally and then propagated among components.

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

- *action-based* specification consists of situation-action rules which specify exactly what to do in certain situations or upon occurrence of a certain event. The situation part corresponds to the specification of variation, while the second part prescribes concrete adaptation actions to be performed. The action-based approaches differ in the way the instructions and the primitive actions are defined and structured.
- *goal-based* specification is a higher-level form of behavioural specification that establishes performance objectives, leaving the system or the middleware to determine the actions required to achieve those objectives;
- *utility function-based* specification exploits utility functions to qualify and quantify the desirability of different adaptation alternatives, and, therefore, permit, on the fly, determination of a "best" feasible state;
- *explicit variability* approach associate the situations, where the adaptation should take place (adaptation points), with a set of alternatives (variants) that define different possible implementations of the corresponding application part.

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

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

- *Dynamicity of decision* refers to the flexibility, with which the adaptation approach may decide on the strategy to be applied. One can distinguish: *static selection*, when the adaptation strategy is predefined and explicitly associated with the given adaptation requirement, situation or event; *dynamic selection*, when the adaptation strategy is selected at run-time

based on a concrete situation, information, and context properties; and *evolution-based selection*, when the adaptation strategy is chosen taking into account not only the current situation, but also the history of previous decisions, adaptations, and their results.
- *Automation of decision* characterizes the degree of the human involvement in the decision process. The degree can range from totally *automatic* (no user intervention is needed), to *interactive* (where the user makes the choice).

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

- *Autonomy* characterizes the involvement of the human in the adaptation execution. It can be done in a *autonomous* way (self-adapt), *manually*, or in an *interactive* form, where the execution of adaptation actions requires human involvement.
- *Invasiveness* characterizes the adaptation framework from the perspective of how tightly it is integrated with the subject of adaptation and the execution framework. We distinguish between the cases, when the adaptation facilities are *integrated with the subject*, the cases, when the adaptation facilities are *integrated with the platform*, where the subject operates, and the cases, when the adaptation facilities are completely *separated* and independent from the subject of adaptation.
- *Realization mechanisms* define the tools and facilities, necessary to enable a given adaptation methodology, to implement the adaptation strategies, and to build the corresponding adaptation architecture. Realization mechanisms strongly depend on a given adaptation problem and on the approach used for that. Typical examples include, in particular, *reflection* wich refers to the ability of a program to reason about, and possibly alter, its own behavior; *automated composition* that provides a support for the automated service composition in order to accomplish composition (or adaptation) goals; *service discovery / binding* that allows to find, select, and exploit a new service as a replacement of the incorrect one; *SLA negotiation* that allows to dynamically agree on the service quality, *aspect weaving* techniques to inject the adaptation facilities into the SBS code, *design facilities* and tools supporting manual adaptation of SBS, etc.
- *Adaptation characteristics* address some important challenges that adaptation process should satisfy, such as safety, security, optimality, cost, performance of the adaptation process.

5.3 Survey Results

In this section we present a survey of the adaptation approaches for Service-Based Applications. We start from the adaptation approaches in business processes and workflow systems as they usually are typical applications based on top of service-oriented architectures. The adaptation solutions, however, in those approaches are different from those in service compositions. First, different levels of abstraction are applied in those adaptations. Second, the degree of automation is radically different in business process and in service composition management. Below we consider both layers.

5.3.1 Adaptation in Business Process Management

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

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

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

Business Process Adaptation by Process Variants

Many approaches in the business process adaptation deal with the problem of defining and dynamically managing process variants. The need to deal with various versions of the process models is motivated by the necessity to accomodate to specific business context, customers, and situations. As the management of variants becomes a complex and error-prone procedure, when the complexity of process models and the number of variants grows, specific approaches are proposed in the literature.

In [28] the PROVOP (PROcess Variants by OPtions) approach for managing large collections of process variants. The basic idea is to keep the variants in the one model. For this purpose, the basic (or the most common case) process is defined, and its variants are represented by the set of change operations that allow the migration of the basic case model into a specific variant model. The transformation operations are defined as action templates, where actions are insert, delete or move process fragment, and modify process attributes. Additionally to the option definition, the PROVOP approach allows for specifying constraints on their usage. The constraints include dependency, mutual exclusion, execution order constraints, and hierarchy. In order to associate the process variants to the process context, the latter is defined explicitly using special context variables and the rules that define their relations and evolution. At run-time the relevant variants are selected and filtered according to the contextual information and the selected options are applied and executed in a process engine. As the context variables may change dynamically, the platform also aims at providing support for run-time migration from one option to another.

Relatively different solution is presented in [24, 45]. While the works address the variability of the business process models, the main focus of the approach is on the problems related to dynamic transformation of one process to another. On the one side, this requires specific approaches for the definition of the process transformation actions and instructions. On the other side, in many cases the transformation should be applied to the already running process instances, potentially leading to unpredictable problems and situations. Apart from modeling process variants, the approaches presented in [24, 45] define the notations and formalisms for describing and implementing transformation actions. These actions define the instructions for changing both control flow and data flow model. In [24] the list of possible actions includes add/remove variable, insert task, add/remove successor of a task, change connection type, or change transition condition. In [45] the actions also include changes in the order of the operations, serialization, task deletion, etc. The presented works also propose solutions for dynamic transformation of already running process instances. The solutions are based on the Petri Net-based formalization of the transformation activities and the transformation correctness conditions.

Semantic Correctness of Process Adaptation

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

Dynamic transformation of enterprise process models is also addressed in [47, 38]. Both approaches propose a formal framework for representing both the underlying dynamic models and changes in them. Differently to other approaches, these works also come up with semantic constraints on the changes in these models, and with notion of correctness and consistency of the changes with respect to those constraints. The underlying adaptation frameworks take into account verification and enforcement of the constraints when the changes are performed. Specifically, in [47] the underlying models correspond to organization models, where one models the structure of the organizations in terms of roles, persons and their relations, as well as the access rules and privileges of these roles. Semantic constraints have a form of correctness conditions of these rules with respect to the mode. The authors provided a way to perform the semi-automated adaptation of access rules in response to changes made on the organizational model. In [38] business process modeling is addressed. Similar to the approach of [24], the authors formally define the language and semantics of changes over business process model. Additionally, semantic constraints are introduces in the form of dependency and mutual exclusion of applicable changes. Using these models, a framework for automated verification of process changes with respect to the constraints is proposed.

Recovery in Workflow Systems

Focusing on the corrective adaptation, recovery actions have been widely used in workflow systems. The workflows models of [12, 22, 46] provide specific capabilities for recovery actions. The approach in [27] focuses on the handling of expected exceptions and the integration of exception handling in the workflow management systems. In [2] the authors propose the use of "worklets", a repertoire of self-contained subprocesses and associated selection and exception handling rules to support the modelling, analysis and enactment of business processes. In [30] the authors consider a set of recovery policies both on tasks and regions of a workflow. They use an extended Petri Net approach to change the normal behavior when an expected but unusual situation or failure occurs. The recovery policies are set at design time.

5.3.2 Adaptation in Service-Oriented Architectures

To this end, a variety of methods for adaptation in Service-Based Applications has been proposed. Given high dynamism of such applications, the adaptation has become an important mechanism for managing continuous changes in the constituent services and their quality, in the context, and in the execution process. Accordingly, the adaptation approaches in service computing range from the self-healing functionalities to recover from application faults to quality degrade, to the customization of the application for the needs of the specific user, context, or even usage scenario of the services. Due to this diversity, the proposed solutions and their complexity differs substantially. Here we consider those approaches starting from simpler forms of adaptation, where the substitution of constituent services is exploited as the only adaptation mean, and moving to much more sophisticated types of adaptation, where the adaptation strategies and actions are reacher and where they are modelled and specified explicitly.

Adaptation by Dynamic Service Binding

A wide range of adaptation approaches relies on the ability to select and dynamically substitute services at run-time or at deployment time. In these approaches, the SBS model, i.e., service composition is defined in abstract way, while the candidate services are bind or re-bind when necessary. Service discovery is particularly relevant in this context, since the services are selected in such a way that the adaptation requirements are satisfied in the best possible way. In many cases the selection is driven not only by the categorization of the replaced service and/or by the necessity to optimize the quality-of-service characteristics of a system, but additional requirements that the replaced service has failed to satisfy. The adaptation goal is, therefore, to bind to a new service that is compliant with these additional requirements.

In [53, 52, 55] the authors present a framework, METEOR-S, for dynamically configuring and executing abstract workflows with a set of available services. The way the processes and requirements are specified relies on a set of specifications defining functional and non-functional constraints on the processes and involved services. In the presented approach the abstract workflow models are defined by the designer and represented in BPEL. Additionally to the workflow specification, the designer provides a set of specific requirements that constrain the functional and non-functional properties of the target process instance and the involved component services. In order to specify this information, the process model is equipped with the semantic specifications in OWL (OWL-S for service descriptions) that describe the domain specific knowledge, the functional and non-functional characteristics of the relevant aspects of the process model. The non-functional constraints refer to QoS properties, security or transactional aspects, and are transformed into integer linear programming constraints. Functional constraints define the statements over the data compatibility or control flow, and are represented in [52] using Semantic Web Rule Language (SWRL). At deployment-time, as well as at run-time when the service failures are detected and the reconfiguration is required, the proposed platform performs service discovery based on the requested service templates and their semantic descriptions, performs quantitative analysis over non-functional properties using linear programming solver, and qualitative analysis over functional properties using a dedicated SWRL reasoner. The run-time execution platform supports also data mediation, and run-time reconfiguration in case of service execution failure. The implementation adopted in [55] relies on a specific algorithm based on Markov Decision Process, which enables coordinated management of adaptation activities in case of distributed service and process constraints.

The approach presented in [9] targets the problem of maintaining dynamic service compositions, when the component services fail or become defective. As in previous approach, the composition is designed as a BPEL process. However, the run-time criterion for selecting the best possible service is different. First, the decision is made for each invocation of the component services. Second, the decision is driven by the only factor, which is service reputation. However, the approach adopts a proactive approach, where the processes proactively provide the reputation information about the usage of a service. If the service invocation was successful, the reputation is positive, while in case of failure the value degrades. This approach allows to improve the quality of selection. However, the system integrator has no way to control or alter such selection and adaptation process. Interestingly, this technique does not require extra description of the component services needed to drive adaptation strategy.

SCENE framework [17] offers a language for composition design that extends the standard BPEL language with rules used to guide the execution of binding and re-binding self-reconfiguration operations. A SCENE composition is enacted by a runtime platform composed by a BPEL engine executing the composition logic, an open source rule engine, Drools, responsible for running the rules associated to the composition, WS-Binder [43] that is in charge of executing dynamic binding and re-binding, and by a Negotiation component that can be used to automatically negotiate SLAs with component services when needed [42]. In a later work (see [13]), the SCENE framework has been extended through the integration of a module enabling the resolution of mismatches between the interfaces and protocols of invoked services. In the paper a set of possible mismatches is defined, together with a list of available adaptation strategies, that can be combined in scripts through a language. The adaptation script specifies the differences between the primary concrete service selected for binding, which is defined at design time, and the other available concrete services that can be candidate for dynamic binding.

The PAWS framework [4] proposes an interplay between design-time and run-time activities. Starting from an abstract process definition, in the design time a selection of candidate services is performed using a semantically enhanced registry, defining mapping information to be used for mediation at run time, and negotiating QoS levels with potentially participating services. At run time, concrete services are selected, based on QoS global constraints and QoS optimization techniques, and services are invoked through a mediation engine to semantically transform input and output messages. The run-time activities are managed by three modules: a Process Optimizer, a Self-healing module and a Mediation engine. The Process Optimizer is in charge of guaranteeing both local and global QoS, according to the constraints required by the user. The Self-healing module enables adaptation, performing semi-automatic actions in reaction to failures. The recovery could imply service reinvocation or substitution. If the recovery requires to substitute the running service, a new service is selected, among the candidates. Finally, the mediation engine, which is set up at design-time, redirects the invocations of the deployed process to the selected services.

Relatively different approach is proposed in [50]. The problem amounts to dynamic substitution of services that fail certain behavioral requirements and constraints. In this approach, the composite application is monitored at run-time, and if the violation of requirements is identified, the platform automatically extracts the additional constraints to the replacing services, and performs service discovery and selection based on those constraints. The requirements to be monitored are related to the behavior of the system or QoS parameters over a service or a composition, and are defined in event calculus in terms of event and fluents. The former correspond to operations performed by the application logic (message reception or emission, assignments, etc), while the latter characterize the state of the application (conditions that hold in some interval). At run-time these requirements are checked by the monitor against the actual executions of the system. When the violation is detected and the platform decides to replace the failed service, new services are discovered and the candidate is selected. The selection is based on the additional adaptation requirements extracted from the diagnostic information provided by the monitor. This information comprises the structural part regarding the categorization and functionality of the failed service and the behavioral part that defines the set of paths that the execution of the target service should respect. The behavioral part is obtained from the violated requirement and the violation synopsis generated by the monitor using predefined transformation rules. The discovery tool checks the behavioral specification of the candidate services expressed as state machines against the behavioral part of the query and selects the corresponding candidate.

QoS-Driven Adaptation of Service Compositions

In case of Web service based processes, the quality of the overall process strictly depends on the quality provided by Web services tied to the task. In this scenario, as defined in the perfective adaptation, it might happen that even if the process runs properly, the adaptation is required because of insufficient quality. As a consequence, the SBS should react in order to improve the quality of the service process. The goal is to select the best set of services available at run-time, taking into consideration process constraints, but also end-user preferences and the execution context. Normally, service selection and binding are used as a key mechanisms for the adaptation in these approaches.

As a consequence, Web service selection results in an optimization problem. The literature has provided two generations of solutions. First generation solutions implemented local approaches, which select Web services one at the time by associating the running abstract activity to the best candidate service which supports its execution. Local approaches can guarantee only local QoS constraints, i.e., candidate Web services are selected according to a desired characteristic, e.g., the price of a single Web service invocation is lower than a given threshold.

Second generation solutions proposed global approaches. The set of services that satisfy the process constraints and user preferences for the whole application are identified before executing the process. In this way, QoS constraints can predicate at a global level, i.e., constraints posing restrictions over the whole composed service execution can be introduced. The main issue for the fulfillment of global constraints is Web service performance variability. Indeed, the QoS of a Web service may evolve relatively frequently. If a business process has a long duration, the set of services identified by the optimization may change their QoS properties during the process execution or some services can become unavailable or others may emerge. In order to guarantee global constraints Web service selection and execution are interleaved: optimization is performed when the business process is instantiated and iterated during the process execution performing re-optimization at run-time. To reduce optimization/re-optimization complexity, a number of solution have been proposed that guarantee global constraints only for the critical path [57] (i.e., the path which corresponds to the highest execution time), or reduce loops to a single task [11], satisfying global constraints only statistically.

In WSCE framework [14] the authors address the problem of adaptive service composition states as follows: "given the specifications of a new service, create and execute a workflow that satisfies the functional and non-functional requirements of the service, while being able to continually adapt to dynamic changes in the environment". This problem requires not only to compose and to bind the services in a composition that satisfies the given requirements, but also to continuously monitor the execution and the environment and to dynamically modify the composition when the critical changes occur. The authors propose the two-staged approach, where first the abstract service composition (template) is defined based on the functional user requirements, and then the abstract composition is instantiated with the dynamic services based on the optimization of QoS metrics. The first stage relies on the composition goal specifications from the Semantics Web services domain, and the second stage relies on optimization of global QoS constraints. Furthermore, at run-time the two approaches may be interleaved. This happens when the platform cannot find a suitable instantiation of currently selected abstract process templates, and the templates should be regenerated from the same user requirements.

In [5], a new modeling approach to the Web service selection problem is introduced. This approach is particularly effective for large processes and when QoS constraints are severe. In the model, the Web service selection problem is formalized as a mixed integer linear programming problem, loops peeling is adopted in the optimization, and constraints posed by stateful Web services are considered. Moreover, negotiation techniques are exploited to identify a feasible solution of the problem, if one does not exist. Experimental results compare our method with other solutions proposed in the literature and demonstrate the effectiveness of our approach toward the identification of an optimal solution to the QoS constrained Web service selection problem.

In [11] the authors propose an implicit approach towards dynamic service composition based on multi-dimensional optimization of quality of service metrics. In the approach the composed process 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 provides 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 authors identify different sets of the relevant quality properties, such as price, duration, reputation, reliability, availability, and define the corresponding aggregation functions for each of them. The predefined goal of the approach is, therefore, at run-time optimize the values of these functions. Since this model is multi-dimensional, the weights should be provided in order to define the global criteria. This weights may be predefined, or set by the end user (as a set of preferences).

The approach of [25] particularly focuses on developing methods to provide the highest QoS. The authors use an extension of WSDL to express properties about the QoS behaviour of a system. The focus is on obtaining an adaptation of the system configuration through and adaptation of the observed QoS behaviour. The information gathered about the QoS behaviour provided is used to compare the different candidate configurations, using genetic algorithm to find the best one. Choosing the configuration with the highest QoS using genetic algorithms: genes are represented by variables concerning service selection and resource allocation. The QoS of each configuration is evaluated, and then a new configuration is generated through mutation.

Another example of run-time management of a SOA configuration can be found in [56]. The approach is based on representing a service configuration through a model, and then modifying this model as needed to adapt the configuration to changes in the environment and in the requirements of the users. This approach addresses the Web service context, particularly the case of highly dynamical environments. The system is described through a Petri Net, which represents the dependencies among the services in the configuration. In other words, the Petri Net represents the places where the services are mapped, and the arcs in the graph model the relationships among the places. This model can dynamically evolve, according to changes in the environment. In [56] the authors describe an algorithm which modifies the configuration (and its model) with the aim to provide the highest QoS. Many different parameters are used to evaluate the QoS, and many metrics are considered to measure each parameter. The QoS provided by a service is defined as a function of place and time, and the described algorithm looks for the best configuration.

While both local and global approaches have been applied, the need for further research toward more advanced optimization techniques, in particular for cycli processes is necessary. In addition, none of the previous approaches considers in the optimization the case of processes composed by stateful Web services, where more than one task must be performed by the same Web service.

Adaptation of Service Interfaces and Protocols

While standardization in Web services makes interoperability easier, adaptation still remains necessary. Adaptation, or in this case we speak about *mediation*, is an important functionality that should be offered to enable integration inside and across enterprise boundaries. We need to generate a service that mediates the interactions among two services with different signatures, interfaces. and protocols so that interoperability can be made effective. The need for adapters in Web services comes from two sources: (1) the heterogeneity at the higher levels of the interoperability stack (e.g., at business-level vs. infrastructural protocols), and (2) the high number and diversity of clients, each of which can support different interfaces and protocols. Such a mediation may be automatic or semi-automatic. Depending on the level of the service specification, this may amount to signature-based adaptation (syntactic properties of the exchanged messages), ontology-based adaptation (exchanged data represent different concepts), or behavior-based adaptation (differences in behavioral specification). In order to perform all these kinds of adaptation, the service descriptions should provide the corresponding models at the different levels of Web service stack.

Generation of adapters for behavioral mismatches is addresed in [10, 8, 21]. The approach requires that the participating service descriptions are equipped with the interaction protocol the service implements. Such a protocol may be defined, for instance, as an abstract BPEL process. While the approach presented in [10] aims at automated generation of an adapter that guarantees the non-locking interaction of the services, the approaches of [8, 21] transfer the problem to the system integrator. In [8] the authors propose taxonomy of different behavioral mismatches and a set of parametric behavioral patterns that may resolve the mismatch. The corresponding pattern is instantiated when the mismatch is detected and proposes it to the application integrator as a possible adaptation strategy. In [21] the authors propose an algebraic model of six transformation operators and the corresponding visual notation that permits, given a pair of required and provided interfaces, construct the necessary adaptation. Based on this construction, a mediation engine performs the necessary run-time actions for processing and managing the messages and service invocations.

An approach presented in [41] addresses wider range of mismatches (signature, merge/split, extra/missing message) providing, however, semi-automated support for adapter generation. By leveraging XML schema matching approaches, the proposed framework allows one to identify the inputs needed to cover the mismatches. For the behavioral mismatches the framework provides an automated generation of mismatch tree that show the possible deadlocks entailed by the mismatch. Based on these two components, the framework and the implementing tool provide the decision support for the engineers that allow for managing mismatches and defining the appropriate adapters.

The work in [37] address the problem of mediation in a different way. Instead of generating external adapters, the authors propose an aspect-oriented framework for aligning internal service implementation to a standardized external specification. In particular, the framework consists of a taxonomy of the different possible types of mismatch between external specification and service implementation (i.e., signature mismatch, ordering mismatch, etc.). The reasoning behind having a taxonomy of mismatches is because the authors argue that similar mismatches can be addressed with similar modifications to the service implementation. Then, for each mismatch, a template that embodies the AOP approach to adaptation is provided. Specifically, the template contains a set of pointcut-advice pairs that define where the adaptation logic is to be applied, and what this actions are. The approach relies on the assumption that the services are realized as BPEL processes, so also the templates are. Finally, the authors present a tool to support template instantiation and their execution together with the service implementation.

Work in [54] focus on Semantic Web services protocol mediation by providing a framework that allows interaction between two services despite the difference of the protocols they rely on. Mediation allows automatic adaptation of a service requester behavior while meeting the interface constraints of the service provider. This is done by abstracting from the existing interface descriptions before services interact and instantiating the right actions when contracting a service. To be possible, a framework managing these levels of abstraction was provided. It consists of defining abstract primitives used by services during their interaction that will be mapped to concrete primitives that represent the real actions in terms of messages exchange between the two communicating parties. Monitoring is however necessary for ensuring the correct use of the abstract primitives that must follow the constraints defined for the service in a low level description language. It could be stated in a state chart expression. An ontology of shared concepts (of the business domain for example) where each concept used in the protocols is defined is also necessary for understanding the semantics of the domain actions which in addition have to exploitable by machine.

Explicit Adaptation Specification Languages

The approaches to adaptation in services and service compositions considered in previous section define the adaptation activities implicitly. That is, the way the system is adapted is somehow predefined and is hard-coded in the managing infrastructure, being service selection and binidng or generation of adapters. As described in the taxonomy, a wide range of approaches comes with the capability to explicitly choose and describe the adaptation specification. Here we present such approaches for the adaptation in service compositions.

In SH-BPEL approach [40] the problem of providing self-healing capabilities to the service compositions in BPEL is presented. The authors aim at extending the standard failure recovery and management capabilities of BPEL with additional functionalities that are crucial in open and dynamic settings. Apart from service-level recovery strategies, such as retry or rebind, the authors identify and describe a wide range of process-specific activities, such as modifying the values of process variables, redoing a process task or an entire part of a process (scope), specifying and executing alternative paths in the process, going back to a "safe point" in the process execution, etc. The proposed solution defines the extension of the standard BPEL execution environment, namely SH-BPEL, which integrates and supports the necessary recovery facilities. In this solution, the original process specification is preprocessed and extended with the additional instructions and control points, which allow for performing the above actions. These additional recovery actions constitute the management API and can be invoked through a special process management interface that is made available at deployment-time. The underlying architecture provides the necessary tools for detecting critical events and engaging recovery actions invoked through this management interface. The existence of a well-defined management and recovery API and interface enables various ways to control and define the recovery strategies. First, these strategies and decisions may be pre-defined in the underlying process manager and customized through a corresponding programming interface. Second, the architecture allows for a collaborative environment, where a set of engines running different composed services participating to choreography perform coordinated recovery activities through the corresponding management interfaces. Finally, the platform allows for recovery both at the instance level, when only current process instance is adapted, and class level, when the whole process model is changed.

In [44] the SH-BPEL framework is extended with the methodology and a tool for learning the repair strategies of Web services to automatically select repair actions are proposed. The methodology is able to incrementally learn its knowledge of repairs, as faults are repaired. This learning technique and the strategy selection are based on a Bayesian classification of faults in permanent, intermittent and transient, followed by a comparative analysis between current fault features and previously classified faults features which suggests which repair strategy has to be applied. Therefore, this methodology includes the ability to learn autonomously both model parameters, which are useful to determine the fault type, and repair strategies, which are successful and proper for a particular fault.

In [7, 6] the authors address similar problem, that is, how to recover the application execution in case of unpredictable service failures in highly dynamic execution environments. The authors exploit run-time monitoring for timely detection of problematic situations. The monitoring instructions are defined as functional and non-functional assertions on the BPEL composition activities. In order to react to the detected violations, the authors propose three kinds of recovery strategies, namely retry, rebind, and restructure. The corresponding instructions for the retry and rebind actions are introduced directly in the composition specification, while the restructure instructions, defined as rewriting rules on the process flow graphs, are defined separately. In [6] the proposed approach is further refined and extended. In particular, the monitoring specification is defined using specific notation, Web Service Constraint Language (WSCoL), which provides expressive and extendable facilities for monitoring composition assertions over functional and non-functional properties of the system. In order to define recovery specifications, Web Service Recovery Language (WSReL) is proposed. The main ingredients of the language are the atomic recovery actions and the recovery specifications that declaratively join atomic actions in two complex procedures (steps) and alternatives. The set of actions include actions at service level (retry, rebind, ignore), at process level (substitute, halt, call), and management actions (notify, log). The supporting platform relies on a specific rule engine that manages recovery specifications and activates recovery, and on aspect-oriented techniques to introduce the recovery implementation at the level of the process engine.

In MASC approach [23] the authors propose policy-driven middleware for self-adaptation to accommodate to various business exceptions and faults. In particular, the policies define the customization actions that should be performed in order to react to certain exceptional situation or a fault. The policies are defined in the form of ECA-rules (Event-Condition-Action) that describe the triggering event (application fault, interaction operations, start/end of an application instance), the condition, under which the rule applies (restriction on the application state or history), and the actions to be applied. The policies are represented in a specific language that extends WS-Policy standard in order to deal with monitoring and adaptation activities. The language allows for defining not only adaptation but also monitoring rules and directives. The monitoring rules define the (relevant) information to be observed and map the undesirable situation or condition to a meaningful failure event that will be processed by the adaptation rules. The approach allows for the specification of two kinds of adaptation action: process-level actions and message-level actions. The former are used to alter business logic of a particular instance, while the latter are mainly used to deal with various low-level faults, such as invocation failure, SLA violation, etc. The possible process-level actions are add, remove, replace, change order of activities, suspend, delay, resume business process. Message-level actions include invocation retries, service substitution, concurrent invocation of several similar services, etc. The adaptation rules may be also assigned priorities to define the order of executions when several rules apply to an event. The adaptation process is supported by the specific platform that allows for monitoring relevant policy information and events and performs the necessary actions both at message-level (intercepting the messages) and process-level (altering the process engine).

A similar problem is addressed in the context of WS-Diamond project [18]. The project focuses on problem diagnosis in service compositions, as well as on the recovery actions for those problems.The service execution environments are extended to include the features to support the diagnostic/fault recovery process. Different types of repair strategies and implementic mechanisms are systematically studied through the project. They include, in particular, instance- and class-level actions, actions refering to the single service or composition-related actions, etc.

In mobile applications pervasiveness and ubiquitous availability are essential features; they exploit a wide range of diverse mobile services provided and advertised locally, making the user and application contexts a central concept in the development and provision of mobile service-based systems. In [48] the authors propose an approach that defines a simple policy-based framework and architecture for designing and providing mobile service-based and context-aware applications. In this approach, the context is represented using Resource Description Framework, and then reflected in definition of policies using the concepts of template (generic contextual aspect) and facts (concrete contextual situation). The context in these settings refers to the properties of the device, possible services and their characteristics, user preferences and settings, etc. The policies define the adaptation actions that should be fired when the system occurs in a certain context. Based on these concepts, the work also defines an iterative methodology for defining contextual information and adaptation policies. Starting from initially collected contextual knowledge, the designer specifies the adaptation policies and extracts intermediate contextual information. The process is iteratively repeated until no new rules can be identified or new contextual specifications can be obtained. The context models and policies are separated into modules and the module pipelines are identified. The process is supported with a visual notation, which shows the initial and intermediate contexts, the final result and the policy processing flow between context modules. The proposed methodology is supported with the agent-based run-time platform that incorporates the monitoring facilities for observing various contextual properties, the communication middleware for interacting with the services, the policy decision maker that filters the policies to be applied, and the policy enforcement point, where the adaptation actions are defined.

Model-based approaches, which are often used at design-time, can also be exploited to obtain automatic run-time adaptation of a system configuration. An example of this approach is presented in [31]. It is based on building a model of the system at design time, together with an initial configuration of its services. The model is object-oriented, and it includes high level policies that specify the desired behaviour of the system. They are composed by a left-hand-side and a right-and-side, which both are graphs of the same kind of the one representing the model. When a part of the system model meeting the left-hand-side of a rule is met, then it is substituted with the corresponding right-hand-side. These simple rules can be easily modify by users as needed, with the same tool allowing for the building of a model of the system. Moreover, the policies described at design-time are deployed on a structure made of services which enables the run-time adaptation of the system.

Adaptation by Explicit Variability Modeling

Explicit variability approaches allow one to define relevant application changes and the way the system should react to it, but to precisely identify a particular moment in the execution, where the change happens, and to represent all the relevant variants of behavior, applicable in such cases. In order to provide explicit definition of the execution location (variation point), where the variation takes place, the behavioral specification of the application is used. The variation point is equipped with a set of alternative behaviors (variants) that may be applied under certain conditions and in particular cases. The explicit variability approaches are widely used in business process modeling and software product line engineering, where variability models provide the basis for application customization, flexibility and re-use. In business processes variation often refers to the single tasks or sub-processes, while in SPL this may also correspond to components, their interfaces and implementations.

In the DySOA approach [49] the problem of reconfiguring the application in order to react to the critical changes in QoS metrics is addressed. As in many approaches, the adaptation actions correspond to selection and binding to a new service or to changes in the composition specification structure. The approach adopted in this work, however, relies on a completely different design method. The relevant adaptation concepts become first-class entities, and the variation of the application is modeled and represented explicitly. The approach is based on explicit modeling of different variants corresponding to the variation points, and on defining various constraints that drive the selection of one alternative or another. Variant defines the behavioral or functional alternative to be applied in the variation point (e.g., process fragment or a concrete component service). The actual code/specification of the variant is defined in its realization definition. The variation model may contain intrinsic variation constraints that restrict the selection of a particular variant, or extrinsic variation constraints define mutual dependencies between various choices potentially at different variation points. The proposed DySOA architecture provides run-time support for the application adaptation. The QoS metrics of the application are continuously monitored and evaluated. When the certain violations are detected, the reconfiguration unit takes care of analyzing and selecting possible variants in the corresponding points, such that all the variation constraints are met and, moreover, the QoS optimality is ensured.

The work in [29] contribution proposed to model the variability of a service-based system using principles and methods from the field of software product line engineering. These include the explicit modeling of commonalities and the variability of a software product line. The variability of a software product line is modeled explicitly by variation points and variants. The authors observe that an adaptive service-based system can have different types of variability, which can be found at the following layers of a service-based system:

- Business Process Layer: This layer includes the definition of business processes from the customer's perspective. The authors use BPMN (the Business Process Modeling Notation) to model the business processes. The

required flexibility in the sequence of activities of a business process is called Workflow Variability by the authors.
- Unit Service Layer: This layer includes the definition of service units from an architect's perspective, which are necessary for handling a business process. The dynamic composition of services is called Composition Variability and it is modeled using BPEL (The Business Process Execution Language).
- Service Interface Layer: This layer includes the definition of service interfaces which reflect the requirements of the service units. The interfaces are described by WSDL (the Web Service Description Language). If alternative interfaces are defined this is called Interface Variability.

For a description of Web services and their composition, the authors use WSDL and BPEL. The respective XML schemas thus are expanded to describe additional properties and composition possibilities of services. By determining the variability types and simple XML-based description of variability, the variability of an adaptive service-based system can be explicitly documented.

In the work of [16] the authors state that dealing with changes in service-oriented systems requires the following information: knowledge about the rationales for decisions; understanding the alternatives; traceability between stakeholders' goals and technical realization elements. In addition, they observe that very flexible and adaptable systems cannot be fully specified in advance and thus postulate that design methods and modeling techniques are needed in order to describe flexible and evolvable systems as far as possible in a declarative way. The approach proposes integrating goal modeling and variability modeling techniques. This is done by identifying variability aspects in i* goal models and by proposing an approach for mapping i* models to variability models. Those variability models in turn are then refined and employed to support monitoring and adaptation of service-oriented systems. The proposed approach focuses on the following two "directions" of changes:

1. Top-down stakeholder-driven changes: A new requirement affects existing services or service architecture. Maybe negotiations with 3rd-party service providers are needed and new SLAs are "signed". The new decisions have to be considered within the service architecture through reconfiguration.
2. Bottom-up monitoring-driven changes: Monitoring results demand changes within the current service configuration to fulfill SLA directives.

In order to deal with both kinds of changes, the authors state that the following facts need to be considered:

- common and specific (individual) goals of different stakeholders;
- knowledge about the current service configuration;
- information about alternative service configurations;
- traceability information to handle dependencies between goals, service types, services and service instances (see definitions below);

- kind of representation of the system at different levels to understand the assignment of stakeholders' needs down to the current and alternative system configurations.

The proposed approach allows modeling these different facts together with traceability and variability information to support system adaptation. In order to map between these facts, the authors propose integrating the metamodels of i* and the variability modeling language used. To map concrete i* models to variability models, the authors have identified six different types of variability (e.g., softgoal and instance variability) and show how these can be specified with the different i* modeling constructs and map to variability models. The mapping between i* models and variability models is supported by the DOPLER tool suite [1]. The Variability management engine (VME) of this tool is used to manage the variability model of available service instances, services, and goals together with the relevant traceability information. The Adaptor component of the tool ultimately performs the requested updates of the service-oriented system. The approach presents how a variability model can be derived from an i* goal model and how this variability - derived from the goal model - can be employed to support the monitoring as well as adaptation of service-based applications.

5.3.3 Comparison of the Adaptation Approaches

The synthetic summary of the adaptation approaches considered in this section is presented in Table 5.1 and 5.3. For the summary the following elements of the adaptation taxonomy are considered: usage of adaptation, subject and aspect of adaptation, location and methodology of the adaptation approach, the specification, the characterization of human involvement (realization of decision mechanisms and level of autonomy), and the specific features of the realization. Note that the approaches we consider here (as well as most of the adaptation approach) deal with temporal modifications with local effect, and therefore these dimensions are not considered here.

While the presented adaptation framework demonstrate good coverage and diversity of the adaptation problems and proposed solutions, there are several important considerations we would like to focus on.

Usage of Adaptation

Different approaches use adaptation for the purpose of recovery (correction), optimization (mainly of QoS properties), and customization. The customization, however, may be caused by different factors. This includes, in particular, contextual properties (i.e., business context or specific customers in BPM systems, operational context or specific application users), or the use of the services in different applications that entails mediation of service interactions and messages. However, the use of contextual factors is currently limited;

existing approaches do not focus on the specific role of various contextual factors in the adaptation process. Another important consideration is the lack of preventive adaptation approaches. Most of the solutions are reactive. those marked as pro-active (i.e., for QoS optimization or mediation) do not aim to prevent some problem, but construct the solution given the current situation without "looking ahead".

Subject of Adaptation

Most of the existing approaches focus on the adaptation of the application instance (i.e., composition or process instance) or even its part (i.e., constituent service). There are few approaches targeting adaptation of the SBS class or the adaptation mechanisms themselves. This makes the adaptation weaker and less flexible in the face of unexpected changes and failures.

Adaptation Strategy

As we already mentioned, there is a lack of pro-active approaches that try to predict and prevent future failures, QoS degrades, trends, etc. Another critical consideration is that the adaptation is usually forward. While such adaptation is easier to implement, it leads to accumulation of problems as the new problems may occur when adaptation is executed. Furthermore, the problem of distribution and coordination of adaptation activities is not considered in the existing frameworks.

Adaptation Specification

While several approaches that provide rich and expressive notations for the adaptation specification, most of the approaches hard code the adaptation activities, providing few or no freedom to its configuration. This is another factor that makes the adaptation framework not flexible and static. Note also that the explicit adaptation languages codify adaptation strategies usually at design time. Consequently, only a subset of possible situation is considered, which may lead to a situation, where the adaptation is harmful for the application [33].

Decision and Autonomy

The adaptation decision in different approaches made either statically (i.e., predefined at design time) or dynamically. The adaptation process in several cases involves also human actors. This latter factor is usually entailed by the inability of the framework to identify appropriate adaptation action (i.e., appropriate mediation protocol), and in some cases is motivated by the necessity to make a critical decision (i.e., which process activity to perform to customize a process, or to recover from the fault). Human involvement, however, should be further studied in order to enhance the capabilities of the adaptation solutions.

Table 5.1. Classification of Adaptation Approaches: Part 1

	Usage	Subject	Aspect	Loc.	Methodology	Specification	Decision	Autonomy	Realization
PROVOP [28]	customize	process instance	workflow	BPM	reactive, forward, centralized	explicit variability, flow instance actions	variants are defined at design time	autonomous	integrated into workflow system, automatic generation of BPEL
ADEPTflex [45]	customize	process instance	workflow	BPM	reactive, forward, centralized	ad-hoc specification of flow instance actions	dynamic, interactive	manual	support for change management
Process adaptation correctness [24, 47, 38]	customize	process instance	workflow	BPM	reactive, forward, centralized	ad-hoc specification of flow instance actions	dynamic, interactive	manual	correctness analysis of changes
Process Recovery [27, 30, 2]	correct	process instance	fault handling	BPM	reactive, forward, centralized	action-based, flow instance actions	static, automatic	autonomous	integrated into workflow system
METEOR-S [52, 53]	customize/ correct	const. services	functional constraints, QoS, security	SC	reactive, forward, centralized	implicit + declarative constraints	dynamic, automatic	autonomous	Integrated into execution framework, SWRL reasoner for constraints
Reputation-based maintenance [9]	correct	const. services	reputation	SC	pro-active, forward, centralized	implicit	dynamic, automatic	autonomous	integrated into BPEL engine, reputation management
SCENE [17, 42, 13]	customize/ correct	const. services	functional constraints, QoS	SC	reactive, forward, centralized	implicit + binding policies and scripts	static, automatic	autonomous	Extension of BPEL engine, support for negotiation and interface mediation

Table 5.2. Classification of Adaptation Approaches: Part 1a

PAWS [4]	optimize/ correct	const. services	QoS	SC	reactive, forward, centralized	implicit + QoS constraints	dynamic, automatic/ semi-automatic (recovery)	interactive	run-time mediation of messages, self-healing capabilities
Discovery framework [50]	correct	const. services	QoS or behavioral correctness	SC	reactive, forward, centralized	implicit + behavioral constraints	dynamic, automatic	autonomous	selection based on run-time behavioral diagnosis information
WSCE [14]	optimize	const. services / composition template	functional requirements, QoS	SC	reactive, forward, centralized	utility function, functional constraints (for composition template) and QoS constraints	dynamic, automatic	autonomous	run-time interlleaving of Semantic Web-based composition and QoS optimization
Flexible processes [5]	optimize	const. services	QoS	SC	pro-active, forward, centralized	utility function, QoS constraints	dynamic, automatic	autonomous	support for QoS negotiation
Composition replanning [11]	optimize	const. services	QoS	SC	pro-active, forward, centralized	multi-dimensional utility function	dynamic, automatic	autonomous	multi-dimensional QoS optimization with customizable weights
QoS maximization [25]	optimize	const. services	QoS	SC	pro-active, forward, centralized	goal-based	dynamic, automatic	autonomous	genetic algorithms for selection
Petri net-based configuration [56]	customize/ optimize	const. services	QoS	SC	reactive, forward, centralized	goal-based	dynamic, automatic	autonomous	Petri-net based composition model

Table 5.3. Classification of Adaptation Approaches: Part 2

	Usage	**Subject**	**Aspect**	**Loc.**	**Methodology**	**Specification**	**Decision**	**Autonomy**	**Realization**
BPEL adapters [10, 8, 21]	customize	service interaction protocol	compatibility	SC	pro-active, forward	implicit	dynamic, automatic/ semi-automatic	autonomous/ interactive	run-time engine for mediation of message exchanges
Semi-automated mediation [41]	customize	interface, data, protocol	compatibility	SC	pro-active, forward	implicit	dynamic, semi-automatic	interactive	generation of mismatch tree for decision support
Aspect-oriented mediation [37]	customize	interaction protocol	compatibility	SC	pro-active, forward	action-based, aspect-oriented	static, semi-automatic	interactive	AOP approach for template instantiation and execution
Mediation in Semantic Web [54]	customize	interaction protocol	compatibility	SC	pro-active, forward	implicit	dynamic, automatic	autonomous	Onotology-based reasoning for identifying mediation actions
SH-BPEL [40, 44]	correct	const. services, composition instance	failure recovery	SC	reactive, forward/ backward, centralized/ distributed	action-based, service and flow instance and class actions	static, automatic	autonomous	highly integrated with BPEL engine and BPEL processes, learning of recovery strategies [44]
Rule-based adaptation [7, 6]	correct	const. services, composition instance	failure recovery	SC	reactive, forward, centralized	action-based, service and flow instance actions	static, automatic	autonomous	instrumentation of BPEL code, rule engine [6]

Table 5.4. Classification of Adaptation Approaches: Part 2a

	Usage	Subject	Aspect	Loc.	Methodology	Specification	Decision	Autonomy	Realization
MASC [23]	correct	const. services, composition instance	failure recovery	SC	reactive, forward, centralized	action-based, service and flow instance actions	static, automatic	autonomous	policy-based middleware based on WS-Policy standard
WS-Diamond [18]	correct	const. services, composition instance	failure recovery	SC	reactive, forward, centralized	action-based, service and flow instance actions	dynamic, automatic	autonomous	failure diagnosis to drive adaptation
Policy-based adaptation [48]	customize	mobile application instance	contextual changes	SC/SI	reactive, forward, centralized	action-based (domain-specific policies)	static, automatic	autonomous	design-time framework for context policy modeling, run-time middleware
Model-based adaptation [31]	customize	service composition	behavioral reconfiguration	SC	reactive, forward, centralized	action-based	static, automatic	autonomous	use of graph rewriting policies
DySOA [49]	correct	composition instance	QoS degrade	SC	reactive, forward, centralized	explicit variability, flow actions	static, automatic	autonomous	integrated into execution platform
SPLE-based adaptation [29]	customize	composition instance	workflow	SC	reactive, forward, centralized	explicit variability	static, automatic	autonomous	extension of BPEL/ WSDL for variability
Goal/Variability modeling [16]	customize	composition instance	changing requirements	SC	reactive, forward, centralized	explicit variability, i*-based goal/ variability modeling	dynamic, automatic	autonomous	integrated into execution platform

5.4 Related Works on Adaptation in Software Systems

Due to the constant changes in information systems, the adaptability has been considered an important challenge in different types of information system disciplines. Here we discuss and demonstrate some of the most active research areas in this line, namely the adaptation in component based software engineering and in software product line engineering.

5.4.1 Adaptation in Component-Based Systems

The problem of building adaptive computing systems has gained dramatically more interest over recent years. The emergence of ubiquitous computing and the growing demand for autonomic computing are the main factors entailing this interest [39]. Ubiquitous computing aims to remove traditional boundaries for how, when, and where humans and computers interact. To do this, computer systems must adapt to its environment of computing platforms and communication networks. Autonomic computing refers to the ability of a system to manage and protect their own resources. Such systems require run-time adaptation in order to survive failures, network outages, and security attacks.

In [34] the authors identify the following groups of reasons for software system adaptation: corrective (remove faulty behavior), adaptive (response to changes affecting the context), extending (extend the system with new functionalities), and perfective (improve characteristics of an application). The authors also classify the adaptation into the following classes: architectural adaptation (affect the structure of the system), implementation adaptation (affect implementation of the components without changing the interface), interface adaptation (affect the interfaces of the components), geography adaptation (affect distribution of the components over the network). Orthogonally to this, adaptation approaches in [39] are classified into parameter adaptation, where the variables that determine the system behavior are affected, and the composition adaptation, where the structural parts of the system are changed.

The rapid growth in the area of adaptation in software engineering is explained by a set of technological reasons [39]. Separation of concerns, computational reflection, and component-based design provided programmers with the tools to construct adaptable system in a systematic way, while widespread use of middleware provided a place to locate and enable adaptive behavior. These technologies, combined in different ways, lead to the development of a wide range of application adaptation approaches and principles [39, 3].

Separation of concerns provides a way to separate development of the functionality and the crosscutting concerns (e.g., quality of service, security). This principle has become one of the cornerstone principle in software engineering, and has lead to a wide spread od aspect-oriented programming (AOP) approach [35]. AOP supports adaptation in several ways. First, many adaptations are relative to some crosscutting concern (e.g., quality-of-service) and therefore AOP may be used to define and implement this concern. Second, it permits delaying the modification of the system to run-time, making adaptation more flexible and dynamic.

Computational reflection refers to the ability of a program to reason about, and possibly alter, its own behavior. Reflection enables a system to reveal (selected) details of its implementation without compromising portability. It comprises two activities: introspection (enables an application to observe its own behavior) and intercession (enables a system or application to act on the observations and modify its own behavior). Together with AOP, it allows for observing and reasoning on the system behavior, enabling its run-time modification.

Component-based design comes with well-defined interfaces, providing a way to develop separately providers and consumers independently, and, therefore, promoting component re-use. We remark that this technology was further advanced by the service-oriented architecture providing even better decoupling, interoperability and re-use of the underlying services (components).

Middleware is a set of services that separate applications from operating systems and protocols. These services include high-level programming abstractions, different aspects (QoS, security, fault tolerance, persistence, transactionality), and specific functionalities. Since middleware provides an abstraction of many adaptation-related concerns, it serves as good place for implementing adaptation mechanisms.

The above technologies and principles has many similarities with the service-oriented architectures and technologies for the SBS development. Not surprisingly, a wide range of adaptation approaches for SBSs adopt similar concepts as the ones for component-based software systems, such as aspect-oriented approaches, use of middleware for realizing adaptation, etc.

5.4.2 Adaptation in Software Product Line Engineering

Software product line engineering (SPLE [15, 36]) has proven to be the paradigm for developing a diversity of similar software applications and software-intensive systems at low costs, in short time, and with high quality. Numerous reports document the significant achievements of introducing software product lines in industry [36].Key to SPLE is to define and realize the commonality and the variability of the product line and its applications. The commonalities comprise the artifacts and the properties that are shared by all product line applications. The variability defines how the various applications derived from the product line can vary. A prerequisite for managing software product line variability is the explicit documentation of the variability.

A Framework for Software Product Line Engineering

The SPLE framework depicted in Figure5.4 illustrates the two product line engineering processes: domain engineering and application engineering. The

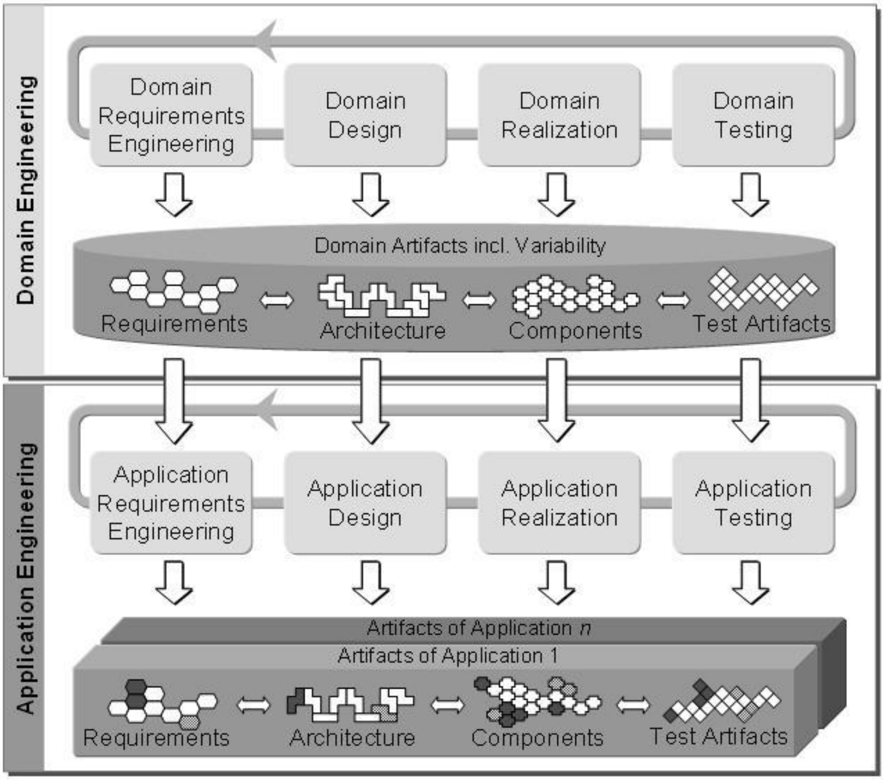


Fig. 5.4. SPLE Framework (Simplified Version of the One in [36])

framework has been developed in the context of European SPLE research projects ESAPS, CAFE, and FAMILIES [51]. The domain engineering process is responsible for defining the commonality and the variability of the applications of the product line [19]. Furthermore, the do-main artifacts are realized which implement the commonalities and provide the variability required to derive the set of intended applications. The domain artifacts constitute the product line platform and include, among others, requirements models (e.g., use case diagrams), architectural models (e.g., component or class diagrams) and test models. The application engineering process is responsible for deriving applications from the domain artifacts. Application engineering exploits the variability of the domain artifacts by binding (resolving) variability according to the requirements defined for the particular application.

By splitting the overall development process into do-main engineering and application engineering a separation of the two concerns building a robust product line platform and creating individual, customer or market specific applications is established.

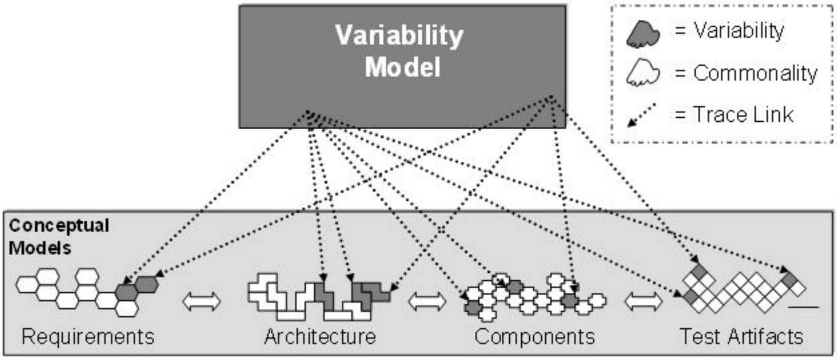


Fig. 5.5. Variability Model and Relationship to Existing Conceptual Models

Two Approaches for Modelling Variability

To model the variability of a product line, two principle types of approaches are proposed in the literature. One type of approaches proposes to integrate variability information into existing models. For example, extensions for UML models by defining stereotypes for product line variability are proposed (e.g., see [26]), or feature models are extended to facilitate the documentation of variability information (e.g., FORM [32], CBFM [20]). The other type of approaches proposes employing a dedicated variability model, i.e., those approaches argue that variability should not be integrated into existing models, but defined separately. Among, others, the Orthogonal Variability Model (OVM, [36]) has been proposed for documenting software product line variability in a dedicated model. In a dedicated variability model only the variability of the product line is documented (independent of its realization in the various product line artifacts). The variability elements (Figure5.5)in a dedicated variability model are, in addition, related to the elements in the traditional conceptual models which "realize" the variability defined by the variability model.

In a dedicated variability model, at least the following information is documented:

- Variation Point ("what does vary?"): This documents a variable item or a variable property of an item.
- Variant ("how does it vary?"): This documents the possible instances of a variation point.
- Variability Constraints: There can be constraints on variability, because product management decided, e.g., not to offer certain combinations of variants in an application or because the realization of one variant requires another variant to be present.

References

1. DOPLER: An Adaptable Tool Suite for Product Line Engineering, volume 2. IEEE Computer Society (2007)
2. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Facilitating flexibility and dynamic exception handling in workflows through worklets. In: Short Paper Proceedings at (CAiSE), Porto, Portugal. CEUR Workshop Proc., vol. 161 (2005)
3. Aksit, M., Choukair, Z.: Dynamic, Adaptive and Reconfigurable Systems Overview and Prospective Vision. In: ICDCS Workshops, p. 84 (2003)
4. Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., Plebani, P.: PAWS: A Framework for Executing Adaptive Web-Service Processes. IEEE Software 24(6), 39–46 (2007)
5. Ardagna, D., Pernici, B.: Adaptive Service Composition in Flexible Processes. IEEE Trans. Software Eng. 33(6), 369–384 (2007)
6. Baresi, L., Guinea, S., Pasquale, L.: Self-healing BPEL processes with Dynamo and the JBoss rule engine. In: ESSPE '07: International workshop on Engineering of software services for pervasive environments, pp. 11–20 (2007)
7. Baresi, L., Ghezzi, C., Guinea, S.: Towards Self-healing Service Compositions. In: First Conference on the PRInciples of Software Engineering (PRISE'04), pp. 11–20 (2004)
8. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing Adapters for Web Services Integration. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 415–429. Springer, Heidelberg (2005)
9. Bianculli, D., Jurca, R., Binder, W., Ghezzi, C., Faltings, B.V.: Automated Dynamic Maintenance of Composite Services Based on Service Reputation. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 449–455. Springer, Heidelberg (2007)
10. Brogi, A., Popescu, R.: Automated generation of BPEL adapters. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 27–39. Springer, Heidelberg (2006)
11. Canfora, G., di Penta, M., Esposito, R., Villani, M.L.: QoS-Aware Replanning of Composite Web Services. In: ICWS 2005 Proc., Orlando (2005)
12. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data Knowl. Eng. 24(3), 211–238 (1998)
13. Cavallaro, L., Di Nitto, E.: An Approach to Adapt Service Requests to Actual Service Interfaces. In: SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, New York, NY, USA, pp. 129–136. ACM Press (2008)
14. Chafle, G., Dasgupta, K., Kumar, A., Mittal, S., Srivastava, B.: Adaptation in Web Service Composition and Execution. In: International Conference on Web Services - ICWS, pp. 549–557 (2006)
15. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2001)
16. Clotet, R., Dhungana, D., Franch, X., Grünbacher, P., Lôpez, L., Marco, J., Seyff, N.: Dealing with Changes in Service-Oriented Computing Through Integrated Goal and Variability Modelling. In: Proceedings 2nd International Workshop on VaMoS (2008)

17. Colombo, M., Di Nitto, E., Mauri, M.: SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 191–202. Springer, Heidelberg (2006)
18. Console, L., Fugini, M.G.: The WS-Diamond Team:WS-DIAMOND: an approach to Web Services - DIAgnosability, MONitoring and Diagnosis. In: e-Challenges Conf. 2007, The Hague (Oct. 2007)
19. Coplien, J., Hoffman, D., Weiss, D.: Commonality and Variability in Software Engineering. IEEE Software 15(6), 37–45 (1998)
20. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing Cardinality-based Feature Models and Their Specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
21. Dumas, M., Spork, M., Wang, K.: Adapt or perish: Algebra and visual notation for service interface adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006)
22. Eder, J., Liebhart, W.: Workflow recovery. In: Proc. of IFCIS Int. Conf. on Cooperative Information Systems (CoopIS), Brussels, Belgium, pp. 124–134. IEEE Computer Society Press, Los Alamitos (1996)
23. Erradi, A., Maheshwari, P., Tosic, V.: Policy-driven middleware for self-adaptation of web services compositions. In: van Steen, M., Henning, M. (eds.) Middleware 2006. LNCS, vol. 4290, pp. 62–80. Springer, Heidelberg (2006)
24. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow Evolution. In: Thalheim, B. (ed.) ER 1996. LNCS, vol. 1157, Springer, Heidelberg (1996)
25. Gao, T., Ma, H., Yen, I.-L., Bastani, F., Tsai, W.T.: Toward QoS analysis of adaptive service-oriented architecture. In: Toward QoS analysis of adaptive service-oriented architecture (SOSE), pp. 219–226 (2005)
26. Gomaa, H., Barber, M.: Designing Software Product Lines With UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley, Reading (2004)
27. Hagen, C., Alonso, G.: Exception handling in workflow management systems. IEEE Trans. Software Eng. 26(10), 943–958 (2000)
28. Hallerbach, A., Bauer, T., Reichert, M.: Managing Process Variants in the Process Lifecycle. In: 10th Int'l Conf. on Enterprise Information Systems (2008)
29. Hallsteinsen, S.O., Stav, E., Solberg, A., Floch, J.: Using Product Line Techniques to Build Adaptive Systems. In: SPLC, pp. 141–150 (2006)
30. Hamadi, R., Benatallah, B.: Recovery nets: Towards self-adaptive workflow systems. In: Zhou, X., Su, S., Papazoglou, M.P., Orlowska, M.E., Jeffery, K. (eds.) WISE 2004. LNCS, vol. 3306, pp. 439–453. Springer, Heidelberg (2004)
31. Illner, S., Pohl, A., Krumm, H., Luck, I., Manka, D., Sparenberg, T.: Automated runtime management of embedded service systems based on design-time modeling and model transformation. In: 3rd IEEE International Conference on Industrial Informatics, pp. 134–139 (2005)
32. Kang, K.C., Kim, S., Lee, J., et al.: FORM: A Feature-oriented Reuse Method with Domain-specific Reference Architectures. Annals of Software Engineering 5, 143–168 (1998)
33. Kazhamiakin, R., Metzger, A., Pistore, M.: Towards correctness assurance in adaptive service-based applications. In: Mähönen, P., Pohl, K., Priol, T. (eds.) ServiceWave 2008. LNCS, vol. 5377, pp. 25–37. Springer, Heidelberg (2008)
34. Ketfi, A., Belkhatir, N., Cunin, P.-Y.: Dynamic Updating of Component-based Applications. In: SERP (2002)

35. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
36. Bockle, G., Pohl, K., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, New York (2005)
37. Kongdenfha, W., Saint-Paul, R., Benatallah, B., Casati, F.: An aspect-oriented framework for service adaptation. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 15–26. Springer, Heidelberg (2006)
38. Ly, L.T., Rinderle, S., Dadam, P.: Integration and Verification of Semantic Constraints in Adaptive Process Management Systems. Data Knowl. Eng. 64(1), 3–23 (2008)
39. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.C.: A Taxonomy of Compositional Adaptation. Technical report, Department of Computer Science and Engineering, Michigan State University (2004)
40. Modafferi, S., Mussi, E., Pernici, B.: Sh-bpel: a self-healing plug-in for ws-bpel engines. In: Proceedings of the 1st Workshop on Middleware for Service Oriented Computing, MW4SOC 2006, Melbourne, Australia, November 27 - December 01, 2006, pp. 48–53 (2006)
41. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, pp. 993–1002, New York, NY, USA, ACM (2007)
42. Di Nitto, E., Di Penta, M., Gambi, A., Ripa, G., Villani, M.L.: Negotiation of Service Level Agreements: An Architecture and a Search-Based Approach. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 295–306. Springer, Heidelberg (2007)
43. Di Penta, M., Esposito, R., Villani, M.L., Codato, R., Colombo, M., Di Nitto, E.: WS Binder: a Framework to Enable Dynamic Binding of Composite Web Services. In: SOSE '06: Proceedings of the 2006 international workshop on Service-oriented software engineering, New York, NY, USA, pp. 74–80. ACM (2006)
44. Pernici, B., Rosati, A.M.: Automatic Learning of Repair Strategies for Web Services. In: Proceedings of the Fifth European Conference on Web Services (ECOWS 2007), November 26 - 28, 2007, pp. 119–128 (2007)
45. Reichert, M., Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control. JIIS, 93 – 129 (1998)
46. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive process management with ADEPT2. In: Proc. of Int. Conf. on Data Engineering ICDE, pp. 1113–1114, Tokyo, Japan (2005)
47. Rinderle, S., Reichert, M.: A formal framework for adaptive access control models. J. Data Semantics 9, 82–112 (2007)
48. Rukzio, E., Siorpaes, S., Falke, O., Hussmann, H.: Policy based adaptive services for mobile commerce. In: WMCS '05: Proceedings of the Second IEEE International Workshop on Mobile Commerce and Services, pp. 183–192, Washington, DC, USA, IEEE Computer Society (2005)
49. Siljee, J., Bosloper, I., Nijhuis, J., Hammer, D.: DySOA: Making service systems self-adaptive. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 255–268. Springer, Heidelberg (2005)

50. Spanoudakis, G., Zisman, A., Kozlenkov, A.: A Service Discovery Framework for Service Centric Systems. In: Proceedings of Service Computing Conference, SCC (2005)
51. van der Linden, F.: Software Product Families in Europe: The Esaps & Café Projects. IEEE Software 19(4), 41–49 (2002)
52. Verma, K., Akkiraju, R., Goodwin, R., Doshi, P., Lee, J.: On accommodating inter service dependencies in web process flow composition. In: Proc. of Int. Semantic Web Services Symposium, AAAI spring symposium series, Palo Alto (CA) USA (2004)
53. Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z.: The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. Technical report (2005)
54. Williams, S.K., Battle, S.A., Cuadrado, J.E.: Protocol Mediation for Adaptation in Semantic Web Services. In: 2nd European Semantic Web Conference, pp. 635–649 (2006)
55. Wu, Y., Doshi, P.: Regret-Based Decentralized Adaptation of Web Processes with Coordination Constraint. In: Proceedings of Service Computing Conference, SCC (2007)
56. Xiong, P.C., Fan, Y.S., Zhou, M.C.: Petri net-based Approach to QoS-aware Configuration for WS. In: IEEE International Conference on Systems, Man and Cybernetics, pp. 1286–1291 (2007)
57. Zeng, L., Benatallah, B., Dumas, M., Kalagnamam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Trans. on Software Engineering 30(5) (2004)

6

Modeling and Negotiating Service Quality

Salima Benbernou[1], Ivona Brandic[2], Cinzia Cappiello[3], Manuel Carro[4], Marco Comuzzi[3], Attila Kertész[5], Kyriakos Kritikos[3], Michael Parkin[6], Barbara Pernici[3], and Pierluigi Plebani[3]

[1] Université Claude Bernard Lyon 1, France
[2] Technische Universität Wien, Vienna, Austria
[3] Politecnico di Milano, Italy
[4] Universidad Politécnica de Madrid, Spain
[5] MTA Computer & Automation Research Institute (MTA-SZTAKI), Budapest, Hungary
[6] Tilburg University, The Netherlands

Chapter Overview. In this chapter the research problems of specifying and negotiating QoS and its corresponding quality documents are analyzed. For this reason, this chapter is separated into two main sections, Section 6.1 and 6.2, with each dedicated to one of the two problems, i.e., QoS specification and negotiation, respectively. Each section has a similar structure: they first introduce the problem and then, in the remaining subsections, review related work. Finally, the chapter ends with Section 6.3, which identifies research gaps and presents potential research challenges in QoS modelling, specification and negotiation.

6.1 QoS Specification

The Quality of Service (QoS) of a service is a set of quality attributes that indicate the service's ability to satisfy a stated or implied requirement in an end-to-end fashion [55]. This set of quality attributes not only characterizes the service but also any entity used in the path between the service and its client. Such an entity may exist in any of the three possible service technology layers described in Chapter 1. Thus, different QoS attributes may be used to define the QoS of a service or service-based application in the *application*, *service*, and *infrastructure* layers.

In the literature two main research approaches can be identified for specifying QoS attributes: *QoS models* and *Quality Specification Formalisms* (QSFs). Section 6.1.1 analyzes which are the main QoS artifacts for services which include the QoS models and the QSFs. Several of these QoS models and QSFs have been proposed. Section 6.1.2 analyzes the general content of QoS models and reviews the most representative QoS models proposed. Section 6.1.3 reviews proposals for QSFs from academia and industry.

The original version of this chapter was revised: The affiliation of Barbara Pernici was corrected. The correction to this chapter is available at https://doi.org/10.1007/978-3-642-17599-2_10

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 157–208, 2010.

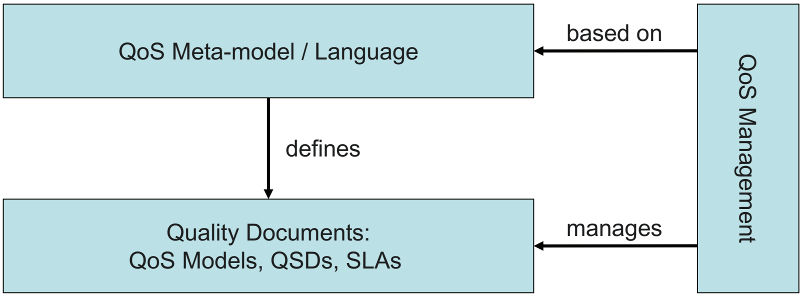


Fig. 6.1. QoS Artifacts

6.1.1 Main QoS Artifacts

In a Service Oriented Architecture (SOA), service providers need to characterize their services to define both the offered functionalities and the offered quality. At the same time, users not only express their requirements by listing the desired functionalities, but also define a minimum level of quality that the service must ensure. The main issue here is the subjectiveness 'quality': the quality of a service from the provider's perspective may be different than the quality experienced by the user. At the same time, the same quality level might be sufficient for a given user and not enough for another one. Thus, an effective quality of service definition must mediate between the intrinsic subjectiveness of quality definition and the objectiveness required whenever we have to compare different standpoints. From the taxonomy proposed by Sabata et al. [88], Figure 6.1 shows the artifacts required to define and manage QoS for services. These artifacts are distinguished between *Quality Documents*, *Quality Specification Formalisms*, and *QoS management*. In the following, we analyze the content and purpose of these artifacts.

QoS models, *Quality-Based Service Descriptions* (QSDs) [10] and *Service Level Agreements* (SLAs) are classified under the term *quality document*. Each of these documents focuses on a different aspect of QoS expression usually with a different granularity of information. A *QoS model* is a taxonomy or categorization of QoS attributes. QSDs express service quality capabilities or requirements in the form of a set of quality constraints that involve QoS attribute and metrics. QSDs are used during service advertisement and discovery, SLAs are produced after service negotiation and express the agreed service quality level between the service provider and requester defined again as a set of quality constraints. Moreover, they contain other information that is used for supporting the *service quality assurance* activity.

Both QSDs and SLAs use or reference QoS models in order to select those QoS attributes and metrics that will be used in their quality constraints. In other words, QoS models provide the concrete semantics of the quality terms that may be used in QSDs and SLAs [10]. Thus, QoS models constitute the basis for expressing QoS. For this reason, we regard them as one of the most significant artifacts in QoS models. Indeed, many QoS models have been proposed in the literature (the most representative are reviewed in Section 6.1.2). However, as QoS models are usually tailored for specific scenarios or application domains, their usage is quite limited. Indeed, no standard QoS model has been yet identified for services. Moreover, the mechanisms used for service discovery, selection, and negotiation and designed based on a specific service QoS model are not extensible and have to be re-designed and implemented each time this QoS model is changed or extended with new QoS aspects.

Knowledge of the QoS offered by a service, in combination with ability to request a certain level of QoS for a particular service, implies an unavoidable need for expressing that QoS. However, widely adopted service description languages (in particular WSDL) do not allow the inclusion of QoS properties with the functional properties. Before service providers and clients can make any decision based on quality of a service, both sides have to agree on a common *Quality Specification Formalism*, which can be used not only to describe basic QoS parameters and their possible combinations but also the provider or requested or finally agreed service quality level. For this reason, many research works have defined a QSF for specifying QoS attributes and other QoS concepts and their inter-relationships, stating the necessity to have a common way for expressing service QoS models. Apart from their ability to specify QoS models, all QSFs are also able to specify either QSDs or SLAs or both types of quality documents. By adopting a specific QSF, mechanisms for service discovery, selection, and negotiation can become more generic as they can be designed and implemented independently of a particular set of quality attributes.

Finally, *QoS management* includes mechanisms needed for supporting users during those service management activities which are encompassed under the term *service provisioning*, i.e., *service discovery* (matchmaking and selection), *service negotiation*, and *service quality assurance*. These mechanisms actually manage the quality documents exchanged by the service users, i.e., service providers and requesters, and thus support the life-cycle of those documents. However, as the quality documents themselves existentially depend on the service that they describe, their management mechanisms are tightly coupled with those of the described service [10]. Indeed, as it is analyzed in [10], QSDs are used and managed during service advertisement and discovery, while SLAs and their respective templates are used also during service quality assurance.

In the remainder of this chapter we will review the ability of QoS models and QSFs to define QoS for a service, using a different perspective for each of the two quality artifacts. In addition, we will review the ability of the QSFs to support a subset of the service provisioning activities, i.e., those of service discovery and negotiation. The interested reader is encouraged to read [10] for a more detailed review of the related work in service quality definition that spans the activities of service provisioning and Chapter 7 for a thorough analysis of related work in service quality assurance.

6.1.2 QoS Taxonomies

In a QoS model, each QoS attribute (i.e., quality characteristic) may belong to one or more QoS groups (quality dimensions). For example, *response time* belongs to the *performance* QoS group. Apart from the categorization of QoS attributes, the structure of QoS models may encompass some other information. Most QoS models describe *technical* QoS attributes, i.e., QoS attributes that characterize the service provisioning and are relevant regardless of the kind of service and its application domain. For instance, the service *availability* is considered an important quality aspect even if we are considering a travel-reservation service, rather than a bank account service. Few QoS models describe QoS attributes applicable to a particular application domain, i.e., *domain-dependent* QoS attributes. This is because it is impossible to enumerate all QoS attributes for all application domains. As a result, QoS models are often extensible to facilitate domain-dependent categorizations of QoS attributes.

Some QoS models describe if QoS attributes are *atomic* or *composite*. Composite QoS attributes can be computed by evaluating the values of other attributes. For instance, *response time* can be computed in terms of *execution time* and *latency*. QoS attributes may also be *measurable* or *unmeasurable*. Unmeasurable quality attributes represent static information which is qualitative in nature. For instance, the *flexibility* QoS attribute can take the following values {*inflexible*; *flexible*; *very flexible*} which denote the degree to which a specific service functions correctly in the presence of invalid, incomplete or conflicting inputs. Measurable QoS attributes are measured with the use of one or more QoS metrics. The latter are concepts which encompass all measurement details of a QoS attribute, such as measurement units, schedules, directives, formulas, and values. Very few QoS models can describe the dependencies between QoS attributes. An example of such dependency which is qualitative in nature is that *availability* and *reliability* have “a positive correlation”, i.e, an increase of the one’s value causes an increase to the other’s value.

In the literature, several research approaches propose a list of quality attributes classified in different ways. For instance, similarly to the ISO 9126 standard which provides a QoS model for software product quality, Sommerville [96] identifies three main categories of non-functional requirements — process requirement, product requirements and external requirements — to cover the whole software life-cycle.

The majority of proposed QoS models for services only describe technical quality attributes. In the following paragraphs, the most representative QoS models are reviewed based on their *extensiveness*, *level of detail*, and *number of referenced service levels*.

Liu et al. [63]

The research described in [63] discusses the need for an extensible QoS model that not only contains general but also domain-specific QoS criteria. It follows that QoS must be presented to users according to their preferences and users should express accurately their preferences with the same QoS model without resorting to complex coding of user-profiles. It also suggests that QoS computation must be fair and open for providers and requesters. Then it proposes a QoS model which is not very extensive as it contains a small number of QoS groups or categories.

Ran [83]

The QoS model in [83] identifies a set of technical quality attributes, grouped in four main categories (*runtime*, *transaction*, *configuration management*, and *security*), that are considered relevant for describing a Web service: i.e, *scalability*, *capacity*, *performance*, *reliability*, *availability*, *flexibility*, *exception handling*, *integrity*, *regulatory*, *supported standard*, *stability*, *cost*, *completeness.* This QoS model is very extensive but it defines only QoS attributes that are relevant at the service level.

Chung et al. [25]

The QoS model proposed in [25] contains 161 attributes and some of them are presented using the NFR (Non-Functional Requirements) Framework proposed by the same authors. These attributes refer to the *accuracy*, *security*, and *performance* requirement categories. Thus, this QoS model is not particularly extensive but it has a great level of detail.

Oasis WSQM [100]

Oasis proposes in its Web Service Quality Model (WSQM) [100] a QoS model with three interdependent components:

1. Quality Factors: the different attributes, dimensions, and measures of the quality.
2. Quality Associates: the organizations or people related to inspection, loading, provision, and use of Web services. These associates may be providers, developers, stakeholders, etc.
3. Quality Activities: the actions which can be taken by the Quality associates related to ensuring the stability of the web service, such as (outstandingly) contracting, but also search, delegation, monitoring, etc.

The overall WSQM is informally defined through the composition of several *quality dimensions*, namely the *business value quality*, the *service level measurement quality*, the *interoperability quality*, the *business processing quality*, the *manageability quality* and the *security quality*. Each quality dimension is decomposed into a set of *quality sub-factors* and associated quality contracts, quality associates, and (in some cases) standards are identified.

Table 6.1. Some Service Level Measurement Quality Factors for WSQM

Factor	**Definition**
Availability	$1 - \frac{Down\ time}{Unit\ time}$
Successability	$\frac{Number\ of\ response\ messages}{Number\ of\ request\ messages}$
Accessability	$\frac{Number\ of\ acknowledgements\ received}{Number\ of\ request\ messages}$
Max. Throughput	$\frac{\max Completed\ requests}{Unit\ time}$

Business Value Quality This measures the value that a Web service usage brings to the business itself. It is dependent on the type of business, and is decomposed into the following sub-factors: *business suitability*, *business effect*, and *business recognition level*. Suggestions of how to calculate these sub-factors and how to compose them together, are given in the WSQM.

Service Level Measurement Quality This deals with the user's perception of the Web service and is divided into *performance* (i.e., *response time*, *maximum throughput*), *stability* (i.e., *availability*, *successability* and *accessibility*). Numerical formulas are given to calculate these indicators (see Table 6.1).

Interoperability Quality This measures the degree of compatibility of a Web service with established standards, such as SOAP, WSDL, and UDDI.

Business Processing Quality This factor is a measure of the swiftness and accuracy with which a business process is executed. It is divided into three sub-factors: *reliability of messaging* (which identifies when unreliable communications happen and when certain properties of the messaging system must be guaranteed), *transactionality* (i.e., ACID properties) and *collaborability* (i.e., the ability to include execution of distributed Web services within a business process).

Manageability Quality This factor measures the quality of the Web service from the viewpoint of the maintainer or developer. It includes subfactors such as *introspectability*, *controlability*, and *notifiability* of the Web service and of the platform, measured by both taking into account the possibility that these sub-factors can be achieved and whether they are actually achieved. These sub-factors are, clearly, more closely related with other classic notions of software quality.

Security Quality This final factor deals with the ability to prevent or avoid unauthorized access to the system and to protect the integrity of data. *Security* is subclassified into several sub-factors including (*data confidentiality*, *data integrity*, *authentication*, *access control*, *non-repudiation*, *accessibility*, *audit trail* and *privacy*), and is defined at two different levels: the *transport* and the *message* level (e.g., use of TLS or IPSEC to ensure user authentication at the transport level, or SOAP messages signed with

XML-DISG at the message level). Several security profiles are defined depending on which quality factors are guaranteed by a Web service.

WSQM is a broad model, tackling QoS from the point of view of several parties. In addition, it defines QoS attributes at all possible service levels. However, as a result of the breadth of the approach, there is a lack of well-defined indicators for the objective comparison of quality levels.

Data Quality

In case where the final product of a service is data, previous work in information quality provides a rich set of approaches for identifying and classifying quality attributes. *Data quality* can be measured along the following dimensions that contain various QoS attributes [85, 98]:

- Data views: relevance, granularity and level of detail.
- Data values: accuracy, consistency, currency and completeness.
- Data presentation: format and ease of interpretation.
- General dimensions: privacy, security, trust and ownership.

6.1.3 Formalisms for Modeling and Specifying QoS Characteristics

In the presence of multiple services with overlapping or identical functionality, service requesters need objective QoS criteria to distinguish one service from another. It is argued in [63] that it is not practical to come up with a standard QoS model that can be used for all services in all domains. This is because QoS is a broad concept and can contain a number of context-dependent non-functional properties such as privacy, reputation and usability. Moreover, when evaluating QoS of Web services, domain specific criteria must be taken into consideration. For example, in the domain of phone service provisioning, the penalty rate for early termination of a contract and compensation for non-service, offered in the service level agreement are important QoS criteria in that domain. Therefore, an extensible QoS model must be proposed that includes both the generic and domain specific criteria. In addition, new domain specific criteria should be added and used to evaluate the QoS of Web services without changing the underlying computation (i.e., matchmaking and ranking) model.

Last but not least, the semantics of QoS attributes/concepts must be described in order to ensure that both Web service provider and consumer can ‘talk’ about the same QoS attribute/concept. Sometimes, generic and domain-dependent QoS attributes with the same name like “availability” may have different meanings to the parties that describe them (i.e., the network level of the hosting system or application that implements the service) or they may be computed differently. As a result, it is important to describe QoS attributes/-concepts not only syntactically but also semantically in order to have a better discovery (matchmaking) process with high precision and recall.

The above problems can be solved with the introduction of QSFs for specifying formal and extensible QoS models. The specified QoS models are able to provide the concrete semantics of all quality terms, i.e., quality attributes and metrics, used to specify the quality constraints in QSDs and SLAs. Moreover, the majority of QSFs define all appropriate concepts that are needed in order to fully construct QSDs, SLAs, or both types of quality documents. Thus, QSFs constitute the cornerstone of quality documents and QoS specifications.

The main approaches proposed in the literature can be distinguished between: a) *pure QoS meta-models*, b) *QoS Languages*, and c) *QoS ontologies. Pure QoS meta-models* use a model-based approach to represent the characteristics of quality attributes or dimensions and to relate them to services and to their use by interested parties. Several models have been proposed as a basis for QoS metamodeling, including UML and Object-Role Model (ORM). The characteristic of these approaches is that they provide a high level, semi-formal description of quality, focusing on syntactic aspects and leaving the semantic aspects informally defined.

QoS languages are either text-based or use XML in order to describe QoS models, QSDs, or SLAs. In the second case, the XML Schema is used to provide rules on how the language constructs can be combined and structured. QoS languages are usually designed using a specific QoS meta-model which is either abstract or is specified informally in an XML Schema. The characteristic of the QoS languages is that they provide a high-level, semi-formal (XML-based) or informal (text-based) description of quality, focusing more on syntactic aspects.

Ontologies provide a formal, syntactic and semantic description model of concepts, properties and relationships between concepts. They give meaning to concepts like QoS attributes, QoS offers, domains, services so they are human-understandable, machine-interpretable and provide the means for interoperability. Moreover, they are extensible as new concepts, properties or relationships can be added to an ontology. In addition, Semantic Web techniques can be used for reasoning about concepts or for resolving or mapping between ontologies. These techniques can lead to syntactic and semantic matching of ontological concepts and enforcement of class and property constraints (e.g., type checking or cardinality constraints). Therefore, by providing semantic description of concepts and by supporting reasoning mechanisms, ontologies cater for better discovery process with high precision and recall. Last but not least, ontologies can help specialized agents/brokers in performing complex reasoning tasks like Web service discovery or mediation. For these reasons, many ontology-based approaches have been proposed and implemented for specifying QoS and there is a trend in using ontologies for specifying QoS in the recent years. However, most of the ontology-based approaches focus only on the support of QSDs, while only one — [74] — goes one step further and provides a semantic language for Web service agreements.

Based on the above categorization of QSFs, the following subsections review the related work in each category according to the criteria defined in [55]. These criteria include:

- An extensible and formal semantic QoS model.
- Standards compliance.
- Syntactical separation of QoS-based and functional parts of service specification.
- Support refinement of QoS specifications and their constructs.
- Allow both provider and requester QoS specification.
- Allow a fine-grained QoS specification.
- Extensible and formal QoS metrics model.
- Allow classes of service specification.
- Use in frameworks or tools.

Pure QoS Meta-models

Non-functional Service Properties Based on the Object-Role Modeling (ORM) [80]

O'Sullivan et al. [80] propose a formal description of non-functional service properties based on the Object-Role Modeling (ORM). Using this approach, the authors define foundational concepts as: service provider, temporal model, locative model, service availability, obligations, price, payments, discounts, penalties, rights, language, trust, quality, security. Concerning service quality, Figure 6.2 shows the definition of a QoS dimension and the relationships with the other concepts introduced in their work.

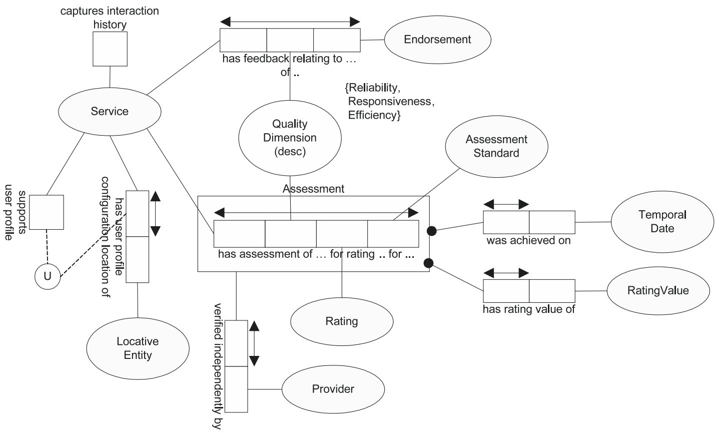

captures interaction history
Endorsement
has feedback relating to … of ..
{Reliability, Responsiveness, Efficiency}
Service
Quality Dimension (desc)
Assessment Standard
supports user profile
has user profile configuration location of
Assessment
Temporal Date
was achieved on
has assessment of … for rating .. for ...
U
RatingValue
has rating value of
Rating
Locative Entity
verified independently by
Provider

Fig. 6.2. Service Quality

QoS Modeling Language [40]

QML (QoS Modeling Language) [40] is another research effort for the specification of a QoS meta-model. It was designed according to some basic principles for the support of QoS specification. It contains the following constructs:

- Contract type: definition of a QoS dimension that includes definitions for the metrics of this QoS dimension.
- Contract: gives particular values/constraints to the fields of a contract type. This is where the idea of contract inheritance is implemented.
- Profile: one service is associated with many (QoS) profiles. Each profile consists of one list of contracts/requirements.

Each contract may describe constraints for a QoS dimension either for the whole service or just for one service operation. But for every QoS dimension, at most one constraint will be valid for one operation of the functional interface of the service. One (QoS) Service Profile *P* is matched with one client profile Q if all contracts of P conform to all the contracts of *Q*. Contract conformance is translated into constraint conformance. Considering its expressivity, QML conforms to many of the requirements set in [55] that were previously referenced in this section and was initially designed based on an abstract meta-model it is now specified in UML. However, the major drawback with QML is that there is no implemented framework supporting it.

OMG Metamodel to Define Quality of Web Services [101]

In [101] the OMG group proposes a UML profile (i.e., a definition of entities and their relations) encompassing generic QoS concepts, which reflect non-functional characteristics that can be uniformly represented. This model is intended to represent the concepts contained in a *metamodel* (an abstract description) that defines what QoS is and how it can be used. This metamodel (which is in itself described using UML) is decomposed into:

QoS characteristics This sub-metamodel includes the *names* (constructors) of the non-functional characteristics/attributes in the QoS model (e.g., latency, throughput); the *dimensions* in which each characteristic is measured (e.g.: reliability can be measured in MTBF, time to repair, etc), as well as the direction of order in the domain, its units, associated statistics, etc.; the possibility of grouping several characteristics (e.g., the performance *category*); the description of the values that quantifiable/measurable QoS characteristics can take, and others.

QoS constraints This makes it possible to express the limits the values that some QoS characteristics can take, and also to specify whether these limits are offered (guaranteed by a provider) or requested (needed by a client). UML classes make it possible to represent a QoS contract which links offered and requested QoS levels between two participants, including the degree to which the constraints are to be satisfied (e.g., *Guaranteed* or *Best-Efforts*) and their extent (e.g., *end-to-end*). Note that constraints can be composed of other constraints.

QoS Level This defines the different modes in which a service can function. Depending on, e.g., available resources, a different execution level can be jumped to if continuing the execution in the current one is not possible. This part of the meta-model defines the abstract classes to represent levels, transitions between them, and when those transitions have to take place.

The metamodel is complemented with a profile that extends it in a constrained way, with a catalog (i.e., a QoS model) of some categories which group characteristics/attributes (e.g., performance characteristics, security characteristics), expressed through UML models. This proposal does not provide any guidance of how to calculate QoS for a given service. However, it does provide a structured representation of the different traits needed to state or calculate the QoS of a service and the relationships between these traits.

QoS Languages

UDDI Approaches [84]

The UDDI[7] Web service standard is dedicated to the description and discovery of Web services. However, UDDI is based on the *tModel* concept which leads to purely syntactic queries. In addition, there is no QoS description of offers or demands in the UDDI description model.

In [84], an extension to UDDI is proposed. A new data structure type — called *qualityInformation* — is added to the UDDI model that represents description of QoS information about a particular Wb Service. This proposed data structure uses the *businessService* and *bindingTemplate* data structure types to provide the binding information for a particular service. The qualityInformation data structure type also refers to tModels as references to QoS taxonomies which also need to be defined in the extended UDDI registry. These taxonomies define the new terminologies or concepts about the proposed QoS information, which do not exist in existing UDDI registries.

The main disadvantage of this approach is that there is no description of the contents of the *qualityInformation* data structure type and its referenced tModels. In addition, it relies on the UDDI technology and UDDI's tModel so can be used only for syntactic matchmaking of QoS terms and specifications.

Modeling Web Service Reputation [67]

In [67], an architecture and model for Web service reputation is presented. It is proposed that for successful description of QoS, three challenges must be dealt with: a) definition of a QoS conceptual model for Web service attribute; b) semantics to QoS service attributes must be added; and c) reputations should consider time and history of endorsements and ratings.

[7] http://www.uddi.org/pubs/uddi_v3.htm

Based on the above requirements and challenges, a conceptual model of Web service reputation is proposed which is used for the calculation of a Web service reputation and is influenced by the following factors: a) relative weights given to QoS service attributes by the requesting user; b) QoS attribute aggregation algorithm for each QoS attribute; c) the set of endorsers of the service and the list of trusted endorsers of the user; d) the history of the service; and e) smoothing the rating so older ratings matter less than newer ratings.

The suggested conceptual model includes a model of QoS attributes. In this model, for each attribute the following aspects are defined: a) its type and allowed values; b) the domains it belongs to, along with a weight of this attribute in relation to the enclosing domain and user preferences; c) the characteristic function from attribute values to ratings; and d) the time characteristics of the values of this attribute.

The main disadvantages of this work are the following. First, the reputation of a Web service is calculated and not its QoS (we believe reputation should be considered as another QoS attribute). Second, there is no explicit clarification of how the reputation of a Web service is calculated. Third, concepts like QoS constraints, QoS offers and demands are not modeled. Fourth, the QoS metrics conceptual model is limited [103].

WSDL Extension for Constraint Specification [29]

In [29], there is an extension to WSDL that allows the construction of constraint specifications for Web services. By using these constructs, a service provider can specify meaningful attributes for characterizing a Web service and its operations. Attribute and inter-attribute constraints can be explicitly defined. In addition, a constraint satisfaction processor was developed for matching the requirements given in a service request against the constraints of registered services in the service discovery process. This processor and some additional components are integrated with the IBM's UDDI registry to form a constraint-based broker. The drawbacks of this approach are the following: a) the QoS-related language constructs/concepts are not rich enough as they do not cover every possible aspect of QoS; b) the semantics of QoS metrics and of other entities are missing; and c) the constraint specification language is not rich enough as it only allows unary attribute constraints and simple inter-attribute constraints of the form "if A then B". This language also does not include linear and non-linear attribute functions.

QRL[27]

QRL [27] is a simple but powerful text-based language used mostly to describe complex constraints between QoS metrics and to analyze how the assessment of QoS metrics will take place based on their values or range of values. Its main highlights are: a) both QoS offers and demands contain not only constraints about their capabilities but also requirement constraints on the capabilities of the other party, and it follows a symmetric approach of QoS-based Web service description for both providers and requesters; b) the QoS demand has a specific part that realizes the specification of the utility function of a QoS metric so the selection part of the Web service discovery process is easier to implement; and c) it relies on the powerful OPL[8] language for describing mathematical or constraint programming problems.

Its main drawbacks are: a) the language is not XML based, and its full specification is not given; b) it is not particularly expressive; c) QoS parameters and their metrics are described in a single common text-based catalog which is not easily maintained; and d) it is transformed to OPL so relies on the OPL language and its specific capabilities.

HP Language to Specify Service Level Agreements [89]

HP Labs have devised a language to specify SLAs in a flexible and extensible way [89]. Their proposal focuses on the ability to express QoS constraints and takes into account they may be seen from several points of view:

- **When** should a service check for SLA conformance? (e.g., after every invocation or as average of n invocations).
- **Which** inputs are necessary for checking SLA conformance?
- **Where** is the conformance monitored? (e.g., at the provider or at the client).
- **What** factors are monitored and how are they measured?

Within these points of view, this proposal is somewhat better suited for time-related constraints, than for other interesting kinds of constraints. Table 6.2 shows an abstract syntax for this SLA language. *Clause* contains the exact information regarding the expected performance. *measuredItem* defines what is being measured (e.g., messages, operations, ports, etc.); *evalWhen* specifies when it has to be measured; *evalOn* specifies the range of data on which the evaluation takes place (e.g., delivery time of a message, or average over some time span); *evalFunc* is the function which is applied to the data to obtain the final QoS evaluation; finally, *evalAction* is the operation to be executed upon measurement (which could take, for example, corrective actions).
The concrete syntax used in the specification is specified through an XML schema.

IBM WSLA [51]

IBM provides a generic SLA framework [51] on top of which different specific QoS models and SLAs can be built. As such, this framework is rich, trying to provide basic blocks to specify metrics, constraints, SLA parameters, etc. The framework also makes it possible to delegate part of the SLA enforcement to third parties and aims at achieving seamless integration with state-of-the-art

[8] `http://www.ilog.com/products/oplstudio/`

Table 6.2. Abstract Scheme of the SLA Specification Proposed by HP Labs

```
SLA            = Dateconstraint Parties SLO*
Dateconstraint = Startdate Enddate Nextevaldate
SLO            = Daytimeconstraint Clause*
Dateconstraint = Day* Time*
Clause         = MeasuredItem EvalWhen EvalOn EvalFunc EvalAction
MeasuredItem   = Item*
Item           = MeasuredAt ConstructType ConstructRef
```

e-Commerce systems. Similarly to [89], specifications following the language of the WSLA framework state what the SLA parameters are, to which service they are related, how they are computed, and which metrics they are using. Unlike other proposals, the aim of the framework is to make this as flexible and customizable as possible.

The SLA definitions are contained in a document which extends the WSDL document(s) corresponding to the service(s) being monitored. These SLA definitions can be applied either to separate operations, or refer to the Web service as a whole – even to compositions of Web services — and cover negotiation, deployment, SLA measurement and reporting, necessary corrective actions and termination.

IBM's WSLA framework has an associated language, WSLA, extensively specified both in syntax and runtime semantics in [64]. WSLA documents have three main sections: the **Parties** section, identifying all the parties taking part in a SLA; the **Service Description**, which specifies the characteristics of the service and its observable parameters; and the **Obligations**, which defines guarantees and constraints on SLA parameters.

The language itself is XML-based and covers all the concepts in the framework, including, for example, the possibility of defining metrics, where the party containing the source of the data for the metric can be expressed, the function to be applied to give the final result (and its units) can be written, and the metric can be composed of other metrics.

The WSLA monitoring services are automatically configured to enforce an SLA upon receipt. The SLA management life-cycle of WSLA consists of five stages:

- Negotiation & Creation: in this stage an agreement between the provider and the consumer of a service is arranged and signed. An SLA document is generated.
- SLA Deployment: The SLA document of the previous stage is validated and distributed to the involved components and parties.
- Measurement and Reporting: In this stage the SLA parameters are computed by retrieving resource metrics from the managed resources and the measured SLA parameters are compared against the guarantees defined in the SLA.

- Corrective Management Actions: If an SLO has been violated, corrective actions are carried out. These actions can be the opening of a trouble ticket or the automatic communication with the management system to solve potential performance problems. Before all actions regarding the managed system are executed, the Business Entity of the service provider is consulted to verify if the proposed actions are allowable.
- SLA Termination: The parties of an SLA can negotiate the termination the same way the establishment is done. Alternatively, an expiration date can be specified in the SLA.

WS-Policy [8]

IBM's WS-Policy [8] is an XML-based language and W3C recommendation which aims at describing models by expressing different types of policies, through *policy assertions*, and how they should be combined. The language design is not made concrete at the level of individual policy assertions, but it focuses more on the combination of several, possibly nested, policies to generate more complex policies. Among the combination patterns in the language we can find patterns for:

- Policy intersection.
- The requirement that a least one out of a non-empty collection of policies is enforced.
- The requirement for all policies in a non-empty collection of policies to be enforced.

This language permits referring to policies using an URI, and has also the notion of "normal form" of policies (since, due to the combination patters above, several ways of writing the same combination are possible). In order to have shorter policy expressions, additional attributes are also defined (for example, to express that a policy is optional). A series of XML transformation operators are defined so that compactly expressed policies can be transformed into a *normal form*, which can then be used to, for example, compare different policies.

WS-Agreement [112]

The work by the Grid Resource Allocation Agreement Protocol (GRAAP) Working Group [2] of the Open Grid Forum (OGF) has led to the development of WS-Agreement [112], the specification of a simple generic language and protocol to establish agreements between two parties. It defines a language and a protocol to represent the services of providers, create agreements based on offers and monitor agreement compliance at run-time. The agreement structure is composed of several distinct parts: *Name*, *Context* and *Terms of Agreement*. The latter is also divided in *service description terms* and *guarantee terms*. Service descriptions terms mainly describe the functionality to be delivered under the agreement. The guarantee terms denote the assurance on service quality for each item mentioned in the service description terms section of the WS-Agreement. In grid resource management such assurances may be denoted as a parameter (constant) or bounds (min/max) on the availability of part or the whole of the resource. In the WS-Agreement, such assurances are referred to as *Service Level Objectives* (SLOs); in a domain specific to computation services provision, they are usually expressed as values. Each SLO may refer to one or more business values, called *Business Value Lists* (BVLs). This list expresses different value aspects of an SLO: *importance* (relative importance of meeting an objective), *reward* (reward to be assessed for meeting an objective), *penalty* (the penalty to be assessed for not meeting an objective), and *preference* (a list of fine-granularity business values for different alternatives, where each alternative refers to a Service Description Term and its associated utility). An SLO can also have a *Qualifying Condition*, which is an optional condition that must be met (when specified) for a guarantee to be enforced (e.g., on external factors such as time of the day or defining conditions to be met by service consumers).

Web Service Offerings Language (WSOL) [104, 105]

WSOL [104, 105] (Web Service Offerings Language) is a WSDL-compatible language to express the QoS that a given service can offer as well as QoS constraints. Functional, non-functional constraints, access rights, as well as management statements (management responsibility, prices, monetary penalties) and different reusability constructs can be expressed within WSOL in a homogenous way that is non-intrusive to the WSLD description, using QoS constraints. A QoS constraint contains the specification of what QoS metrics are monitored, as well as when and by what entity, and usually describes QoS guarantees.

QoS constraints can be grouped in classes of services, termed *service offerings* (also possible within the meta-model of [101]) which bundle together related QoS constraints; different classes of QoS can be separately applied to a single Web service. One advantage in so is that changes in the environment conditions (due to, e.g., network problems) can be compensated for by renegotiating the QoS with the same service which was being accessed, without the need to start another search and composition phase — unless, of course, the alternative service offering is not satisfactory.

The available types of constraints are defined in an XML schema, and these may usually refer to arithmetic and boolean operations. However, metrics and measurement units are assumed to be defined in an external ontology. Interestingly enough, WSOL makes it possible to define, besides post-conditions of the Web service operations, constraints that have to be checked some time (to be defined) after an operation takes place, periodically, etc.

The (dynamic) relationship between service offerings is not represented in WSOL. It is, rather, represented in a specific XML format the triple

$$\langle ServOff_1, ContrState, ServOff_2\rangle$$

where $ServOff_1$ and $ServOff_2$ are the initial and replacement service offerings and *ContrState* are the constraint and statements which were not satisfied by $ServOff_1$.

The shortcomings of this proposal include: a) the lack of integration between constraint dimensions; b) the need to improve the specification of relationships between service offerings to support both easier and more flexible specification and dynamic adaptation; c) there is no specification of the QoS demand of the consumer; and d) the ontologies for metrics have, to the best of our knowledge, not yet been developed.

QoS Ontologies

Ontologies for QoS [103]

[103] describes the specification of constraints between QoS metrics. In this work, five ontologies must be developed from which the base ontology is the metrics ontology. The authors describe the structure and involved elements in four out of the five ontologies. However, the description is limited to the requirements for the metrics ontologies and does not propose any specific ontology. In addition, the requirements specified are incomplete according to the requirements from [55]. For example, the 'metric' class consists only of five attributes while other important attributes/properties are missing. Another example states that metrics should be related to each other. However, the authors do not describe the types of relationships that can be made between QoS metrics.

OWL-S [99]

The OWL-S [99] ontology is a semantic approach for the description of Web services. It has many advantages in respect with the other Web service description standards but it does not describe QoS offers or demands. It only contains an attribute used for rating a Web service. However, as it is an ontological approach, it can be extended in order to describe QoS offers or demands.

In [117], the DAML-S (OWL-S) Web service description language is extended to include a QoS specification ontology called DAML-QoS. This is achieved by the following: a) a *ServiceProfile* element is associated to many QoS profiles (service offerings); b) external ontologies in DAML for metrics and units are referenced or developed; and c) the existence of a *BasicQoSProfile* containing all the basic metrics and ability to inherit/extend this type of profile to provide constraints and/or include custom-made metrics. DAML-QoS is supported by an implemented QoS-based Web service discovery framework.

The deficiencies of this research are: a) the proposed ontology is limited, does not capture many aspects of QoS description and it is not accompanied by a QoS model, e.g., describing domain-independent QoS attributes and metrics; and b) the QoS metrics values are restricted to have the set of natural numbers as their range for better reasoning support. This approach, however, leads to imprecision and errors up to one half of measurement unit. Moreover, this flaw is actually the result of a misuse of the ontology language's (i.e., OWL's) cardinality constraints.

Relations among QoS Attributes [67, 68]

The work analyzed in [68] is a continuation of the work in [67]. The requirements of the work in [67] have been translated into an expressive ontology language. This work's main highlight is the formalization of relationships between QoS attributes. When a QoS attribute depends on another then either value can influence the value of the other with a specified impact or through changing the values of another attribute. A framework using the ontology to support dynamic Web services selection is also outlined.

The main drawback of the proposed ontology is the lack of a metric description model. In addition, this ontology lacks both an openly available implementation and links to a semantic description Web service language like OWL-S.

Mapping Requirements from Higher Layers onto the Underlying Network [102]

The research described in [102] analyzes what must be enclosed into the QoS information for a Web service request or advertisement with the help of a QoS ontology. Important elements of this ontology are *QoSInfo* and *QoS-Definition*. *QoSInfo* describes standard or user-defined *serverQoSMetrics* and *transportQoSPriorities* and the values they will take. It also references protocols used by a Web service for security and transaction support. The *QoS-Definition* element describes QoS information (*QoSInfo*) either for the whole service or for a particular operation of the service. Additionally, it includes information about protocols supporting service management and QoS monitoring and about the trusted third-parties that will participate in these protocols. It ends with the price for the usage of this service supporting the QoS offer. One Web service advertisement is related to many service offers (*QoS-Definition*) while one service request enquires one particular service offer. One important feature of this research effort is that it supports the mapping of QoS requirements from higher layers onto the underlying network in terms of the Internet model. This mapping is achieved by the help of proxies (residing at the provider and consumer) and by the existence of a QoS-aware network. QoS network parameters are given as guidelines to QoS-aware routers while the client proxy calculates the network performance by taking into account the server performance information provided by the server proxy. This research work comes with three main deficiencies. First of all, there is not a complete and accurate description of QoS constraints as QoS constraints are just equality constraints. Secondly, metrics ontologies are not developed, but are just referenced. Finally, this work is not supported by Web service discovery framework.

QoSOnt and Upper Level Ontologies [31, 106, 53]

QoSOnt [31] is another carefully designed ontology for semantic QoS-based Web service description. Its main features are: a) an ability to measure QoS attributes by many metrics; b) the application of a QoS metric to either whole Web service or a single operation; c) an approach to unit conversion using SWRL rules; and d) a direct connection to OWL-S. This is a good approach but not complete. In addition, it is not accompanied by a formal Web service discovery framework.

The work in [106] proposes an upper-level ontology that uses the main features of the ontologies produced by the work of [68, 31]. In addition, a mid-level ontology has been designed for domain-independent QoS properties (i.e., a QoS model). The proposed ontology is rich and is also connected to OWL-S. However, it lacks information on how QoS constraints are specified and it is not publicly available. In addition, it is also not supported by a Web service discovery framework.

OWL-Q [53] is an upper ontology carefully designed on specific requirements [55]. The ontology is composed of many facets, each capturing a particular aspect of QoS-based Web service description. It is publicly available available[9], directly connected to OWL-S and is supported by a QoS-based Web service discovery framework. A mid-level ontology has also been designed for domain-dependent QoS metrics. In addition, this ontology is now enriched with SWRL rules in order to support unit transformation, statistical metric value derivation, semantic property inference and matching of QoS metrics [54].

Enriched WS-Agreement [74]

The work in [74] semantically enriches the WS-Agreement language in order to develop a semantic framework for matching agreement specifications of providers and requester automatically. The WS-Agreement language is extended in important areas such as the *SLO* and *QualifyingCondition* with the addition of the *expression*, *predicate*, *parameter*, and *value* tags as defined in the WSLA specification [51]. In addition, four ontologies are used to provide a commonality of terms between agreement parties and to provide rich domain knowledge to the search engine so that it may achieve the best possible match results: 1) an OWL ontology for representing the WS-Agreement schema; 2) a OWL-based QoS ontology encompassing QoS concepts used in guarantees; 3) a third OWL ontology representing domain knowledge; 4) the OWL Time

[9] http://www.csd.uoc.gr/~kritikos/OWL-Q.zip

ontology [47] for representing temporal constructs such as *endTime*, *interval*, *dayOfWeek*, and *seconds*. Moreover, this approach uses SWRL rules in order to: a) transform one *SLO* to another one that is semantically similar but syntactically heterogeneous with respect to the first one; b) to compare *SLO*s according to the semantics of a domain specific predicate; c) to derive new domain knowledge by e.g., producing a new *SLO* from one or more other *SLOs*; d) to enable user assertions over subjective personal preferences. Last but not least, it must be stated that this extended language is connected to WSDL-S[10] and is supported by a complete semantic QoS-based Web service discovery framework. This work has the following deficiencies: a) QoS metrics are not modeled at all; b) *SLO*s of guarantees are expressed in terms on unary constraints (i.e., containing just one QoS concept); c) Although timing conditions are expressed in *guarantee*s, this does not happen with the whole *alternative*.

WSMO-QoS [108]

WSMO-QoS [108] is an upper level ontology complementary to the WSMO[11] semantic language for functional Web service description. It is also directly connected to WSMO. Besides this upper-level ontology, a vocabulary (i.e., a QoS model) of general domain-independent QoS attributes has also been developed. WSMO-QoS is a very rich ontology capturing many aspects of QoS metric description. It includes and allows many metric value types (including linguistic, numeric and boolean), dynamic calculation of metrics values, the attachment of units to metrics, unit transformation functions, the expression of the tendency of the metric's value from the user's perspective and grouping of QoS attributes. This ontology is supported by a QoS-aware selection framework of Web services. The main deficiencies of this ontology are the following: a) there is a one-to-one mapping of QoS attributes and metrics, which is incorrect; b) there is no measurement modeling, i.e., functions and measurement directives of metrics are not expressed; c) only equality constraints on metrics are allowed, which is quite restrictive; and d) it is not yet publicly available.

onQoS-QL [43]

The onQoS-QL [43] ontology is rich, and covers all aspects of QoS-based Web service description in almost the same way as OWL-Q. It is also supported by a semantic QoS-based Web service discovery framework. Its main highlights are: a) the use of scales for restricting the metric value types; b) the use of unary and binary predicates for constructing metric constraints; c) metric constraints have both retrieval and ranking semantics; d) many important types of measurement processes are supported. The main drawbacks of this

[10] www.w3.org/Submission/WSDL-S/

[11] www.wsmo.org

work are: a) no connection to a functional Web service description language; b) measurement process is external and not specifically defined with the use of metric functions and measurement directives; c) only unary and binary metric comparison predicates are used for expressing QoS constraints.

6.1.4 Trust and Security QoS Models and Formalisms

The growing number of services accessible in open distributed systems necessarily calls for security enforcement. Examples of such systems include the World Wide Web, grid computing systems, distributed intelligence systems and ad-hoc networks used by the military [45][86]. The security requirements for these systems can include guarantees of confidentiality, integrity, authentication, authorization, availability, auditing and accountability [79, 7]. A significant issue for service-based applications is that if they are publicly accessible, such as over the Internet, there is no way of knowing in advance the users that will be accessing the service. Ensuring the users of a Web service can gain access to it where no relationship exists between the user and the service provider is a challenging task which requires a flexible approach to access control through *trust* mechanisms.

A number of surveys relating to trust have been published: [45] provides an overview of trust in Internet, [7] discusses trust in computer science and semantic web, [86] examines trust management. [7] organizes trust research in four areas: 1) policy-based trust, using policies to establish trust, focused on managing and exchanging credentials and enforcing access polices; 2) using reputation to establish trust where past interactions or performance for an entity are combined to assess its future behaviour; 3) general models of trust, where trust models are useful for analyzing human and agent-based trust decisions; and 4) trust in information resources in relation to Web resources and the reliability of Web sites.

Our study focusses on the first area. Traditional access control models rely on knowing requester identities in advance [36]. However, service-based applications typically have large and dynamic requester populations meaning that requesters' identities are seldom known in advance. Most existing Web applications deal with strangers by requiring them to first register an identity at the Web site. Such approaches do not fit into the Web service philosophy of dynamically choosing services at run-time. As a result, traditional assumptions for establishing and enforcing access control regulations no longer hold. Trust negotiation is an access control model that addresses this issue by avoiding the use of requester identities in access control policies [114, 14]. Instead, access is granted based on the trust established in a negotiation between the service requester and the provider. In this negotiation — called a trust negotiation — the requester and the provider exchange credentials. Credentials are signed assertions describing attributes of the owner. Examples of credentials include membership documents, credit cards, and passports. The attributes of these credentials are then used to determine if access should be granted.

For instance, a requester may be given access to resources of a company by disclosing a credential proving she is an employee of that company. This example shows that the requester identity is not always needed to determine access. Credentials are typically implemented as certificates.

To summarize, the goal of a trust negotiation is to find a sequence of credentials $(C1, ..., Ck, R)$, where R is the resource to which access was originally requested, such that when credential Ci is disclosed, its policy has been satisfied by credentials disclosed earlier in the sequence or to determine that no such credential disclosure sequence exists (more details about trust negotiation will be discussed in Section 6.2.3). In recent years, trust negotiation has been proposed as a novel authorization solution and as an automated technique for the establishment of trust and the enforcement of need-to-know between entities in open-system environments, in which resources are shared across organizational boundaries.

6.2 QoS Negotiation

This section reviews research concerning automated the negotiation of a service's QoS. We tailor the discussion of QoS negotiation in SBAs around: 1) application level quality negotiation; 2) QoS negotiation in grid services; and 3) mechanisms for trust and security negotiation.

In current practice the QoS of a Web service is usually statically defined. Policies or, more generally, quality documents are attached to Web services when they are deployed. These documents are retrieved and analyzed once a service is requested. In Web service selection, quality documents are matched against the quality requirements expressed by applications or users requesting a service, whereas, in service composition, information about quality of a Web service can be used to make a decision on whether to consider or not a Web service for executing a process task. Such an evaluation is made on the ability of a service offer to satisfy local and global quality constraints that users can express on tasks or the whole process, respectively.

In theory, the above mentioned approach represents a *take-it-or-leave-it* or *one-shot* negotiation of Web service QoS. In other words, the service requestor is forced to accept the QoS offer made by the provider and there is no opportunity for setting the QoS profile of a service at runtime on the basis of the providers or requestors preferences, cost model or willingness to pay. However, Web services are usually provided in a loosely coupled and highly variable environment. On one hand, Web services can be offered with multiple QoS profiles and, on the other hand, the offered QoS may vary according to current conditions, such as network load or the number of requests served in a given time instant. The interests of both service providers and requestors on the QoS of a Web service may also be different. For instance, the costs sustained by a Web service provider to increase the availability of its services may not be balanced by a comparable increase of the service requestors' benefits in obtaining a service with higher availability. The on-the-fly setting of QoS profiles on the basis of the service oriented architecture users' needs can be instantiated through the adoption of automated negotiation frameworks in the context of SBAs.

This section reviews relevant literature in the field of QoS negotiation in SBAs and is structured as follows. Section 6.2.1 revises research on application level Web service QoS negotiation, classifying contributions on the basis of the QoS model adopted for negotiation, the supported negotiation protocols, and the chosen architectural style adopted for the implementation of the negotiation infrastructure. Adopting similar classification criteria, Section 6.2.2 discusses the issue of service QoS negotiation in Grid computing, where negotiation is mostly employed as an admission control mechanism for resource reservation. In a loosely-coupled environment, service requestors and providers are not likely to know each other in advanced, since business relationships can be created on-the-fly. This is why we consider establishing trust as a paramount issue in SBAs and why we review relevant work in the field of trust and security negotiation in Section 6.2.3.

6.2.1 QoS Negotiation in Web Services and Semantic Web Services

The objective of this section is to revise relevant literature on QoS negotiation in a Web service context. In particular, the literature on Web service QoS negotiation is constituted by several isolated contributions. The heterogeneity of contributions in this field can be ascribed to different motives. First, negotiation is not a *native* research issue of Web services, since it has been studied since 50 years by microeconomics and, more recently, by the literature on multi-agent systems. Moreover, QoS negotiation represents a tool for improving the management of architectures using Web services. In particular, typical issues in loosely coupled environments management, such as service discovery and selection, composition, and monitoring, raise different issues concerning the negotiation of QoS. Finally, QoS negotiation can be implemented according to different paradigms, such as in broker-based architectures and multi-agent systems. Each implementation paradigm introduces specific issues that must be dealt with while tackling the Web service QoS negotiation problem.

In order to provide a common background for classifying research on Web service QoS negotiation, we start from understanding the nature of the negotiation problem in the Web service context. As underlined in the previous sections, QoS of a Web service is usually defined by multiple attributes, which span from performance related to domain specific QoS dimensions. Hence, the negotiation of Web service QoS can be usually intended as a multi-attribute negotiation problem [50]. For what concerns negotiation participants, negotiation can be either one-to-one or multiparty. Specifically, the participants involved in Web service QoS negotiation are the service requestor, who requires a service with a certain level of quality, and one or more service providers, which provide services with variable quality.

More specifically, our literature classification is grounded on the three basic elements that define an automated negotiation problem [50, 34], that is:

- **Negotiation Object.** It defines the features of the *object* that is under negotiation. While, for instance, in a planning problem agents may negotiate which actions need to be taken in the future, in the Web service context the object of negotiation is always QoS. Therefore, as already underlined, negotiation is always multi-attribute because of the multi-attribute nature of Web service QoS.
- **Negotiation Protocol.** It is constituted by the set of rules that define what is allowed in a negotiation and how negotiation participants can exchange messages.
- **Negotiation Strategy.** It defines the decision models of negotiation participants. A decision model defines how negotiators generate new offers, when they accept offers, or when to withdraw from negotiation.

Starting from the framework above, we propose three dimensions for classifying the research contributions in the field of Web service QoS negotiation research. First, we classify research contributions according to the features of the quality description on which negotiation is performed. Secondly, we focus on the negotiation protocols that can be instantiated by the proposed solutions. Finally, we classify contributions according to the architectural paradigm chosen for their implementation. A summary of the classification made in this section is reported in Table 6.5.

The first classification is based on the nature of the negotiation object, i.e., how to define the QoS dimensions on which negotiation is performed. In particular, we make a distinction between approaches to negotiation that rely on a *fixed* set of QoS dimensions and other approaches that adopt *extensible* ways to define QoS. When QoS dimensions are fixed, their number, types, and metrics cannot be modified according to the domain in which QoS negotiation occurs. QoS dimensions, in this case, are usually performance-related, since they are usually independent from the application domain. Conversely, extensible QoS usually rely on the definition of a model, either declarative or ontological, which can be used to define and use ad-hoc QoS dimensions.

The research in [22, 113], for instance, presents an architecture for executing QoS negotiations between service requestors and providers in an Internet applications outsourcing scenario. The negotiation considers a fixed set of QoS dimensions, i.e., the throughput, the availability, and the latency of service provisioning.

Declarative extensible QoS models are considered in [26, 30, 44]. QoS dimensions can be defined in terms of name, metric, unit of measure, and evaluation method. An ontological model for QoS description is proposed in [57], where a QoS ontology is required to validate the content of policies that define Web service SLAs.

Table 6.3. Fixed vs. Extensible Negotiable QoS Definition: Advantages and Drawbacks

	Fixed set of QoS dimensions	**Extensible QoS model**
Advantages	Real world use cases specification. Rigorous definition of QoS metrics and evaluation methods.	Ease in including domain specific QoS. Facilitated QoS lifecycle management. In tune with the open world perspective of SOA.
Drawbacks	Domain specific QoS constrained to chosen use cases. No QoS lifecycle management.	Lack of real-world use cases.

Most of the approaches presented in the literature rely on extensible QoS models, that can be either declarative or ontological. In particular, declarative QoS models [44, 71] include QoS definitions into policies that are usually attached to published services. Ontological models [57] organize the elements that define a negotiable QoS dimension, such as name, metric, and monitoring methods, in an ontology, usually expressed in OWL-S.

It should be noted the extensible QoS models reviewed so far do not allow the dynamic definition of QoS dimensions while negotiation is executing. In other words, relevant QoS dimensions for a given domain should be defined prior to the negotiation architecture deployment, since it is not possible for negotiating participants to define new QoS dimensions on-the-fly.

Table 6.3 summarizes the principal benefits and drawbacks of the two alternatives concerning QoS models. The main benefit of considering a fixed set of QoS is that is provides the opportunity to adopt real-world use cases with a rigorous definition of relevant QoS characteristics. Conversely, extensible QoS models trade-off the adoption of real world use cases in favor of flexible and generalizable QoS descriptions that are easier to be managed, i.e., updated, modified and revised.

Concerning the negotiation protocol, we discern between approaches that support one-to-one (1:1) negotiations between the service requestor and a single service provider and one-to-many (1:N) negotiations between the service requestor and *N* service providers.

On the one hand, 1:1 negotiation applies to the case of automated SLA establishment [26, 44]. In this case, the service requestor has already chosen a service but, since such service can be provided with variable quality, the QoS of the Web service can be automatically negotiated at runtime.

On the other hand, 1:N negotiation applies to the issues of Web service discovery and selection [41], when the service requestor must choose among a set of functionally equivalent Web services that can be distinguished only by their variable QoS profile. A utility-based approach to QoS-based Web service selection is proposed in [70]. In particular, although no actual negotiation algorithms are provided, the authors propose a service selection method which maximizes the utility of the Web service consumer while guaranteeing costs constraints.

In case of service compositions, current solutions for Web service QoS automated negotiation rely on the coordination of a set of bilateral negotiations between the service requestor and the services involved in the composition [30, 22]. In particular, [22] proposes two different methods for dealing with such a coordination. The *negotiate-all-then-enact* approach involves a round of bilateral negotiation before enacting the service compositions. The objective is for the service requestor to obtain agreements on QoS that satisfy his or her global QoS requirements. The second approach is the *step-by-step-negotiate-and-enact*, in which the QoS of a service is negotiated just before it is invoked. This second case increases the complexity of obtaining QoS agreements which satisfy the service requestor's global QoS requirements.

Semantic-based negotiation mechanisms and protocols have often been inspired by the literature produced by the Artificial Intelligence (AI) and agent communities. This is because in these fields the goal is to design appropriate models with automated and tractable negotiation mechanisms such that autonomous agents can deal with. [56] provides a survey of approaches for multi-attribute negotiation in AI. Faratin et al. [35] presents a formal account of a negotiating agent's reasoning component to generate the initial offer, to evaluate the incoming proposal, and to generate the counter proposal. Regarding the protocols, the FIPA Communicative Act Library Specification [38] is considered as a foundational approach for defining specialized negotiation protocols. An example architecture based on this specification is discussed by Chhetri et al. [23].

Focusing on the semantic Web community, Chiu et al. [24] discuss how ontology can be helpful for supporting this negotiation. In particular, the authors highlight how shared and agreed ontologies provide common definitions of the terms to be used in the subsequent negotiation process. In addition, they propose a methodology to enhance the completeness of issues in requirements elicitation. Lamparter et al. [58] introduce a model for specifying policies for automatic negotiation of Web services. In this case, the starting point is the upper ontology DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [66]. On this basis, this work proposed a policy ontology that also includes the user preferences specifications and an additional ontology for expressing the contracts.

Regarding the use of ontologies for specifying the agreement among parties, Oldham et al. [75] present reasoning methods for the components of an agreement which must be compatible for quality matches. This approach is based on WS-Agreement and takes advantage of Semantic Web methods to achieve rich and accurate matches. Along the same lines, Uszok et al. [107] have developed KAoS policy ontology that allows for the specification, management, conflict resolution and enforcement of policies within the specific contexts established by complex organizational structures.

We argue that providing a review of negotiation strategies proposed in the literature is out of scope of this chapter since the analysis of negotiation strategies belongs to different fields of inquiry, such as agent-based computing and microeconomics. Moreover, we want to stress that the reviewed approaches to Web service QoS negotiation are not innovative from the point of view of negotiation strategies, since they rely on well known families of strategies, such as concession-based [26], learning-based [22], or search-based strategies [30].

We end our discussion with a classification of the architectural styles adopted for the implementation of the architecture that supports the design and the execution of negotiations. In particular, we have identified two main approaches for implementing QoS negotiations, i.e., *broker-based* and *multi-agent* architectures.

Broker-based architectures imply the existence of a third party, i.e., a broker, which executes QoS negotiation on behalf of service requestors and providers [26, 70]. The negotiation strategy of the participants is made known to the broker by means of policies. Once having read the policies, the broker is able to simulate the whole negotiation process. The need for including a broker to execute the negotiation was introduced in [65]. The architecture of a broker for supporting Web service QoS negotiation is described in [26]. On the requestor side, negotiation can be either automated, by means of policies, or manually executed by human actors. [44] and [71] propose frameworks for QoS negotiation based on policy matching. Although the description of an architecture is out of scope in the policy framework definition, the authors hypothesize the existence of a broker which execute QoS negotiation and policy matching.

Multi-agent architectures exploit Multi-Agent Systems (MAS) to execute Web service QoS negotiations. Specifically, negotiation is executed by agents that negotiate on behalf of external actors, that is, Web service providers and requestors. The underlying multi-agent system is usually built according to the FIPA specification [37]. Negotiation agreements are then translated into XML documents which can be easily managed by the Web service architecture.

[22, 113] propose a multi-agent based negotiation system for coordinated negotiations in service compositions. A coordinator manages the message exchange among service providers and requestors' negotiating agents built from service offers and requirements, respectively. A multi-agent negotiation system that supports multiple protocols is proposed in [30]. A rule-based system implements different negotiation protocols, such as auctions and 1:1 negotiations, while message exchange is managed with an event-based approach. Negotiating agents implement strategies for generating or accepting offer and withdrawing from negotiation on the behalf of their owners, i.e., service requestors and providers. Finally, the Collaborative Agreement for Automatic Trading (CAAT) is a multi-agent negotiation system built on top of the FIPA specification for managing negotiations in Web service choreography [73]. CAAT is described in terms of the agents' interaction protocol, which is based on

Table 6.4. Broker-Based vs. Multi-agent QoS Negotiation Architectures: Advantages and Drawbacks

	Broker-based	**Multi-agent**
Advantages	Reduced negotiation communication overhead. No need to create new agents/services for negotiating.	Customizable negotiation strategies embedded in agents. Increased flexibility in implementing complex negotiation protocols. Strategies not disclosed to third parties.
Drawbacks	Trust and security (information disclosure to third party). Need for complex policy model definitions. Lack of scalability (the broker may become a bottleneck).	Communication overhead (message exchange among agents). Need to integrate agent-based platforms into Web service architectures. Often a coordinator of negotiating agents is required. Effort required for building and managing negotiating agents.

an ontology of concepts, and supports bilateral negotiation and moderated negotiations, i.e., bilateral negotiation moderated by an external third party.

Table 6.4 summarizes the principal benefits and drawbacks of the two alternatives for Web service QoS negotiation implementation. The main benefit of broker-based solutions is to reduce the negotiation communication overhead, since negotiation strategies and offers can be automatically generated by the broker. At the same time, the need to declare strategies, e.g., behavioral and pricing models, to a third party introduces issues of privacy and security that will be further analyzed in the sections to come. Agent-based solutions have higher communication overhead and need to rely on an underlying multi-agent platform in order to be implemented. The main advantages of agent-based solutions are customization and flexibility, since service requestors and providers can create ad hoc agents that reflect their personal negotiation strategies.

6.2.2 Negotiation Protocols in Grid Computing

The shift of emphasis in Grid computing towards a service-oriented paradigm and the trends in application service delivery to move away from tightly-coupled systems towards loosely-coupled, dynamically bound systems, has led to the adoption of Service Level Agreements (SLAs) as the standard concept through which Grid schedulers can allocate resources and allow coordinated resource management. In the context of Grid and Web services, the current understanding is that an SLA is an electronic contract, which is expected to be negotiated almost fully-automatically, as such, must be machine readable and understandable. As a result, there has been a significant amount of recent research on various topics related to SLAs.

Table 6.5. Comparison of Approaches to Web Service QoS Negotiation

	QoS Model		Neg. protocols			Architecture	
	Fixed QoS	Ext. QoS	1:1	1:N	others	Broker-based	Agent-Based
Chhetri et al. [22]	√				√ (WS composition)		√
Yan et al. [113]	√				√ (WS composition)		√
Comuzzi and Pernici [26]		√	√			√	
Di Nitto et al. [30]		√	√	√			√
Gimpel et al. [30]		√	Not specified			√ policy matching	
Lamparter et al. [57]		√ ontology	Not specified			√ policy matching	
Mukhi and Plebani [71]		√	Not specified			√ policy matching	
Menasce and Dubi [70]		√		√ Web service selection and composition		√	

Service Negotiation and Acquisition Protocol (SNAP) [28]

The first promising negotiation protocol in the field of Grid Computing was the Service Negotiation and Acquisition Protocol (SNAP) [28]. The negotiation *objects* are tasks (QoS user requests) and resources (its characteristics). This work proposes three different types of SLAs:

1. *Task service level agreements* (TSLAs) in which one negotiates for the performance of an activity or task. It characterizes a task in terms of its service steps and resource requirements.
2. *Resource service level agreements* (RSLAs) in which one negotiates for the right to consume a resource. An RSLA can be negotiated without specifying for which activity the resource will be used. These two SLAs can be regarded as negotiation *protocols*.

3. *Binding service level agreements* (BSLAs) in which one negotiates for the application of a resource to a task. The BSLA associates a task, defined either by its TSLA or some other unique identifier, with the RSLA.

By combining these agreements in different ways, a high variety of resource management approaches can be represented, including: batch submission, resource brokering, co-allocation and co-scheduling. These variations define different negotiation *strategies*. The authors have also shown an SLA usage scenario for resource brokering, which they call the 'community scheduler' scenario. They define a community scheduler as an entity similar to a broker (see previous section) that acts as a mediator between the user community and the Grid resources. Activities are submitted to the scheduler rather than to the end resource, and the activities are scheduled to resources in a way that optimizes the use of the community's resources. A Grid environment may contain many resources, all presenting a RSLA interface as well as a TSLA interface. Optimizing the use of resources across the community by the scheduler is only possible if the scheduler has some control over the resources used by the community. Hence the scheduler negotiates guarantees via RSLAs with a pool of underlying resources, and exploits those capabilities via TSLAs and BSLAs. This set of agreements abstracts away the impact of other community schedulers as well as any local workloads, assuming the resource managers enforce SLA guarantees at the resources. Community scheduler services present a TSLA interface to users. Thus a user can submit a task to the scheduler by negotiating a TSLA, and the scheduler then turns to a resource by binding this TSLA against one of the existing RSLAs. The scheduler may also offer an RSLA interface. This would allow applications to co-schedule activities across communities, or combine scheduled resources with additional non-community resources. The various SLAs offered by the community scheduler result in a very flexible resource management environment. Users in this environment can interact with community and resource-level schedulers as appropriate for their goals and privileges.

In the area of Grid Computing SLA use is of great importance in the area of resource management. The following use cases, gathered in a technical report as [93], describe the need for such agreements:

- Agreement on resource usage: for a presentation with live demonstration of an application the necessary compute resources to run the application have to be available at the time of the presentation. In a normal cluster environment where the nodes of the cluster are used under a space-sharing policy, the probability of finding a free slot that matches the requirements of a user immediately is low, thus his job usually will be queued and executed later. In order to have the resources required at the time of the presentation the presenter needs to reserve the resources in advance to be sure that they can be used for the demonstration at the time foreseen. This reservation can be expressed as a Quality of Service and an SLA may be issued where the reservation is fixed.

- Agreement on multiple QoS parameters: in an environment containing several compute clusters operated in different administrative domains, SLAs might be used for co-allocation or workflow-based resource allocation. A typical use-case is the co-allocation of multiple compute resources along with the network links between these resources with a dedicated QoS to run a distributed parallel application. The user specifies his request and a resource broker starts the negotiation with the local scheduling systems of the compute resources and with the network resource management system in order to find a suitable time-slot, where all required resources are available at the same time. Once a common time-slot is identified, the broker requires the reservation of the individual resources. This reservation can also be expressed as a Quality of Service and an SLA may be issued where the reservation is fixed. Another use-case is a workflow spanning several resources. The only difference to the use-case described before is the type of temporal dependencies: While for the distributed parallel application the resources must be reserved for the same time, for the workflow use-case the resources are needed in a given sequence. Thus, a scheduler needs to negotiate the reservations such that one workflow component can be executed on the required resource after the preceding component is completed.
- Grid Scheduler interoperation: as there is no single orchestrating service or Grid scheduler in a Grid spanning countries and administrative domains, we have to deal with multiple instances of independent Grid schedulers. Using resources from different domains requires co-ordination across multiple sites. Two approaches can support such co-ordination: a) direct negotiation with respective local scheduling systems; and b) negotiation with the respective local broker. The first requires local policies allowing a remote broker to negotiate with local schedulers, which is in general unrealistic. In the second case, there is one access point to the local resources, which then negotiates on behalf of the initiation broker. As the second approach also has a better scalability than the first one, the OGF Grid Scheduling Architecture Research Group (GSA-RG) [3] decided to consider this approach for the definition of a Grid Scheduling Architecture. For the communication between the different Grid schedulers, a language and protocol to create SLAs was selected to achieve the necessary interoperability, while at the same time resulting in SLAs at the end of the negotiation process that can be combined by the initiating scheduler into one single agreement with his client.

The work presented in [91] provides a deeper investigation of SLA usage for computational job scheduling in Grids. Computational jobs, submitted to high-performance computing resources for execution, are associated with an SLA. This SLA is negotiated between a client (e.g., a user or a resource broker) and a provider (the owner of a resource with its own local scheduler) and contains information about the level of service agreed between the two parties, such as acceptable job start and end times. Users negotiate and agree an SLA with a resource broker. Brokers negotiate and agree an SLA with users; these SLAs may be mapped to one or more SLAs, which are negotiated and agreed with local resources and their schedulers. Finally, local schedulers need to schedule the work associated with an SLA they have agreed to (the constraints associated with such an SLA, agreed by a resource, may be stored locally in the resource, in some kind of a resource record). It is also noted that a single SLA between a user and a broker may translate to multiple SLAs between the broker and different local resources to serve the user's request (for example, this could be the case when the SLA between a user and a broker refers to a workflow application with several tasks that are executed on different resources). In such case, the user may want to set constraints for the workflow as a whole and the broker may have to translate it to specific SLAs for individual tasks. To indicate the possible differences between these two types of SLA, the terms meta-SLA and sub-SLA are used. Furthermore, this SLA-based view for job submission, may still allow the submission of jobs that are not associated with an SLA. However, no guarantee of their completion time would be offered in this case.

To introduce SLA usage to job scheduling the following challenges need to be solved:

- SLA vocabulary: the vision of SLA-based scheduling assumes that the SLAs themselves are machine readable and understandable. This implies that any agreement between parties for a specified level of service needs to be expressed in a commonly understood (and legally binding) language.
- SLA negotiation: it is envisaged that SLAs may be negotiated between machines and users or only between machines. In this negotiation some commonly agreed protocol needs to be followed. This protocol needs to take into account the unreliable nature of the distributed systems and networks used for the negotiation and should abide by appropriate legal requirements. In addition, during negotiation, machines should be able to reason about whether an offer is acceptable and possibly they should be allowed to issue counter-offers.
- Scheduling: currently, most scheduling of jobs on high-performance compute resources uses priority queues (with the possible addition of backfilling techniques). The use of SLAs would require the development of a new set of algorithms for efficient scheduling, which would be based on satisfying the terms agreed in the SLA. The existence of these efficient scheduling algorithms would be of paramount importance to estimate capacity and reason on the possible acceptance (by a local resource) of a new request to form an SLA.
- Constitutional Aspects: in the context of any SLA based provision, sooner or later the need for dispute resolution will arise. In addition, users may also be interested in the reliability of specific brokers; for example, how likely is that a broker will honour an SLA, even if breaking the SLA would

require the broker to pay a penalty? This issue of modeling reputation may also be related to the approaches followed for pricing and/or penalties when agreeing SLAs.

These requirements and the previously introduced use cases reveal the relevant properties of SLA usage in Grids. The negotiation *objects* are the static resource characteristics (e.g., CPU, memory, disk), the resource capabilities (e.g., licenses, reservation, price, software) and the user requirement properties (e.g., time, cost constraints). The negotiation *protocol* and *implementation* depend on the actual solution, such as the WS-Agreement and the WSLA languages. The negotiation *strategies* usually depend on the actual protocol but solutions using the same protocol may differ. SLA-based Grid resource management relies heavily on the concept of renegotiation of existing agreements. Renegotiation generally means reconsideration of the quality and the level of service, such as processor, memory or bandwidth requirements, resource reservations, and so on. Renegotiation requires user involvement during the job execution. Therefore, the most important issue is to reduce the amount of user interaction. In [90] researchers provided an interesting solution to achieve this goal within the WS-Agreement framework. The basic idea is to regard guarantee terms as functions to increase the flexibility of the agreement. They introduced a list of universal variables (i.e., current wall clock time, network bandwidth) and a list of predefined functions (i.e., min/max bound, list average, Gaussian function). As a result the Service Level Indicatiors/Objectives in the SLAs are described not as constants but as functions of universal variables. In this way, the terms of the WS-Agreement are no longer constants or independent range-values. Such an agreement has an infinitely large number of outcomes, therefore it is able to describe the entire guarantee terms of an SLA. This way of SLA usage provides a richer term set for Grid applications and reduces the need for renegotiation.

QoWL [20]

[20] presents an approach for high level Grid workflow specification that considers a comprehensive set of QoS requirements. Besides performance related QoS, it includes economical, legal and security aspects. For instance, for security or legal reasons the user may express the location affinity regarding Grid resources on which certain workflow tasks may be executed. The QoS-aware workflow system provides support for the whole workflow life cycle from specification to execution. Workflow is specified graphically, in an intuitive manner, based on a standard visual modeling language. A set of QoS-aware service-oriented components is provided for workflow planning to support automatic constraint-based service negotiation and workflow optimization. For reducing the complexity of workflow planning, a QoS-aware workflow reduction technique is introduced.

Amadeus [21]

[21] presents the Amadeus framework, a holistic service-oriented environment for QoS-aware Grid workflows. Amadeus considers user requirements, in terms of QoS constraints, during workflow specification, planning, and execution. Within the Amadeus environment workflows and the associated QoS constraints are specified at a high level using an intuitive graphical notation. A set of QoS-aware service-oriented components is provided for workflow planning to support automatic constraint-based service negotiation and workflow optimization. The Amadeus framework uses static and dynamic planning strategies for workflow execution in accordance with user-specified requirements.

End-to-End QoS for Compute-Intensive Medical Simulation Services [12]

In [12], a Grid infrastructure is presented that addresses the issues in offering end-to-end QoS in the form of explicit timeliness guarantees for a compute-intensive medical simulation service. Within the GEMSS project [1], parallel applications installed on clusters or other HPC hardware may be exposed as QoS-aware Grid services for which clients may dynamically negotiate QoS constraints with respect to response time and price using SLAs. The infrastructure uses standard Web services technology and relies on a reservation based approach to QoS coupled with application specific performance models.

6.2.3 QoS Negotiation in Security

In this section two specific aspects of QoS negotiation in security are discussed: 1) trust negotiation; and 2) privacy negotiation.

Features of Trust Negotiation System

As described earlier, the goal of trust negotiation is to allow ‘strangers’ to access sensitive data in open environments without violating the data’s access policy. In trust negotiation, two parties establish trust through iterative disclosure of credentials and requests for credentials to verify properties of the negotiating parties. To carry out a trust negotiation, parties usually use strategies implemented by an algorithm. Most of the research in this area focuses on protecting resources and credentials and assumes that policies can be freely disclosed. Research in trust negotiation has focused on a number of important issues, including languages for expressing resource access policies such as Trust-X, X-TNL, Trustbuilder, Cassandra (e.g., [9, 13, 16, 62]), protocols and strategies for conducting trust negotiations (e.g., [115, 4, 52, 46]), and logics for reasoning about the outcomes of these negotiations (e.g., [81, 111]). The initial results presented in this work have been shown to be viable as access control solutions for real-world systems through a series of implementations (such as those presented in [15, 4, 110]) that demonstrate the utility and practicality of these theoretical advances.

Based on the current work, we can state that a trust negotiation system may include four components:

- Trust negotiation policy module.
- Credential manager module.
- Strategies negotiation module.
- Trust negotiation policy management module.

We now define the requirements for each of these components. We propose some dimensions to evaluate the current relevant negotiation models. However, these dimensions are not limited and can be augmented due to the increasing requirements in this area.

Trust Negotiation Policy Language Requirements

A trust negotiation policy language is a set of syntactic constructs, including credentials and policies and the way they encode access control needs [15, 14]. The following discusses the characteristics of the policy language.

Policy specification. Trust negotiation policies specify which credentials — and other resources — to disclose at a given state of the trust negotiation, and the conditions to disclose them. The specification can be formal (using logic, algebra, etc.) or semi-formal.

Specification of the combination. The language may be expressive enough to contain operators combining credentials characterizing a given subject.

Authentication of multiple identities. Under this approach to automate trust creation, each party can have many identities with each identifying a particular credential issuer. For instance, if today my purse includes my drivers license, my credit card and my library card I may be asked to prove that I possess one or more of these identities at any time to establish a level trust based on the credentials issuers.

Sensitive Policy protection. The protection can be handled at the policy level or system level in the sense that by analyzing policies' content, outsiders might infer sensitive information about parties.

Trust negotiation policy lifecycle management. The trust negotiation policy life-cycle management may include details about how to update trust negotiation policies in a consistent manner and how to cope with dynamic policy evolution, that is, changes to a policy during a trust negotiation.

Credential Manager Requirements

Type of disclosure. Context-aware applications should reveal credentials only in the correct context. This disclosure needs some actions to be specified that will satisfy what those conditions are.

Automation. The credential exchange can be automatic (no user intervention is needed), semi-automated (where some aspects are automated and others need user intervention) or manual.

Credential chain discovery. The credentials used in a negotiation can be available locally or discovered at run-time.

Ownership. Some systems use various security protocols during negotiation, where a remote credential is received asking for verification to prove the ownership. Some other systems integrate negotiation frameworks with the existing tools and systems.

Strategies Negotiation Requirements

Dynamicity

In some domains, such as e-Health, not all entities are willing to negotiate credentials or disclose access policies directly to strangers regardless of negotiation strategies and instead prefer to negotiate and disclose sensitive information only to strangers within what we refer to as a *circle of trust*. The requirement here is to describe how locally-trusted intermediary parties can *delegate* whilst protecting credentials and access policies [5].

Privacy Protection Mechanisms

Privacy is one of the major concerns of users when exchanging information through the Web and we believe that trust negotiation systems must effectively address privacy issues in order to be widely applicable. Some research investigates privacy in the context of trust negotiations, e.g., [39, 17, 97, 78] propose a set of privacy-preserving features for inclusion in any trust negotiation system and [92] proposes the identification of privacy vulnerabilities in trust negotiation.

Automation

The exchange of attribute credentials is a means to establish mutual trust between strangers wishing to share resources or conduct business transactions. Automating trust negotiation to regulate the exchange of sensitive information during this process is therefore important [49, 109].

Bilateral Trust Creation

It is important not only for a service provider to trust the clients requesting its services but also for clients to trust the services that they choose to interact with.

Scalability

How much trust creation can be automated in a highly scalable manner? Can it be truly ubiquitous?

Analyzing Existing Trust Negotiation Work for Web Services and Grid Computing

In this section we survey and analyze several approaches based on the requirements presented earlier.

Web Services

Existing efforts in the area of trust negotiation have not been standardized and do not fit into any authorization standard such as the eXtensible Access Control Markup Language (XACML). In [69], it is investigated how XACML can fit into trust authorization management systems by exploring existing concepts, and where necessary, extending them to accomplish the goal. The authors propose the XACML Trust Authorization Framework (XTAF), which is a loosely-coupled architecture with a trust component that protects authorization information (policies and credentials) layered in such way that it integrates seamlessly into any XACML compliant authorization engine with minimal effort. They show how the XACML policy language can be used to support bilateral exchange of policies and credentials, and protect unauthorized access to services. They also introduce a Trust Authorization Service Handler (TASH) to handle trust and privacy of authorization information. This supports runtime bilateral authorization operations between two or more parties.

In [95], authors propose a model-driven approach to trust negotiation in Web services. The framework, called Trust-Serv, features a trust negotiation policy model based on state machines. Importantly, this model is supported by both abstractions and tools that permit life-cycle management of trust negotiation policies, an important trait in the dynamic environments that characterize Web services. In particular, TrustServ provides a set of change operations to modify policies, and migration strategies that permit ongoing negotiations to be migrated to new policies without being disrupted.

In [94] the WS-ABAC model is proposed to use attributes associated with subject, object and environment, and service parameters for access control measures in Web services environment. The WS-ABAC model extends the traditional RBAC model to gain many advantages from its attributes-based capability. In this model, authorization decisions are based on the attributes of the related entities. The ATN mechanism is used to provide the needed attributes and addresses the disclosure issue of the sensitive attributes when the attribute conditions are not satisfied. So it can protect user's privacy. Only when the user does not give the needed attributes for authorization decision, the access request is rejected.

Re-designing and re-standardizing existing protocols to make authorization decisions meet the needs of large-scale open systems is time consuming. To address this problem, in [59, 79, 60], authors propose a system called Traust, a stand-alone authorization service that allows for the adoption of trust negotiation in a modular, incremental, and grassroots manner, providing access to a wide range of resources without requiring widespread software or protocol upgrades. It uses the current prototypes Trust-X or TrustBuilder to allow clients to establish bilateral trust.

In [77], the authors investigate the problem of trust in Semantic Web services. They include trust policies in WSMO, together with the information disclosure policies of the requester, using the Peertrust language [42]. Peertrust provides the means to perform trust negotiation and delegation. As the matchmaker must have access to both the requestor's and provider's policies, in order to match the requestor's functional requirements and trust information the authors propose a distributed registry and matchmaker architecture that allows the service providers to keep their policies private, thus not forcing them to disclose sensitive information.

In [32], security-by-contract is proposed as a novel negotiation framework where services, credentials and behavioral constraints on the disclosure of privileges are bundled together, and clients and servers have a hierarchy of preferences among the different bundles. While the protocol supports arbitrary negotiation strategies, two concrete strategies (one for the client and one for the service provider) make it possible to successfully complete a negotiation when dealing with a co-operative partner and to resist attacks by malicious agents to "vacuum-clean" the preference policy of the honest participant.

The Web Services Trust Language (WS-Trust) uses the secure messaging mechanisms of WS-Security to define additional primitives and extensions for the issuance, exchange and validation of security tokens. WS-Trust also enables the issuance and dissemination of credentials within different trust domains.

The goal of WS-Trust [72] is to enable applications to construct trusted [SOAP] message exchanges. In order to secure a communication between two parties, the two parties must exchange security credentials (either directly or indirectly). However, each party needs to determine if they can "trust" the asserted credentials of the other party. This specification defines extensions to WS-Security for issuing and exchanging security tokens and ways to establish and access the presence of trust relationships. Using these extensions, applications can engage in secure communication designed to work with the general Web services framework, including WSDL service descriptions, UDDI businessServices and bindingTemplates and SOAP messages.

Grid Computing

In [61], a novel trust negotiation framework is proposed, TOWER, which provides distributed trust chain construction and management with the goal of enhancing Grid security infrastructure. The approach leverages attribute-based credentials to support flexible delegation, and dynamically constructs trust chains. The TRust chAin based Negotiation Strategy (TRANS) is proposed to establish trust relationships on-the-fly by gradually disclosing credentials according to various access control policies. It is implemented in the CROWN Grid. In [48], ROST is presented, an original scheme for Remote and hOt Service deployment with Trustworthiness. By dynamically updating runtime environment configurations, ROST avoids restarting the runtime system during deployment. In addition, trust negotiation is included in ROST, which greatly increases the flexibility and security of the CROWN Grid.

In [6], several classes of trust and their use in Grids are analyzed: service provision, resource access, delegation, certification and context trust. Current technologies for managing trust have been also discussed. The concept of Virtual Organizations is central to Grids. The authors have enriched the classical VO life-cycle with trust management activities. Trust values and trust policies are created before starting the VO identification phase. In the VO Identification phase, trust information such as reputation could be taken into account when selecting potential VO candidates. The VO formation phase includes all activities related to trust negotiation. During VO operation, trust values are computed and distributed among the VO participants. In VO dissolution, trust information such as credentials and access rights are revoked to avoid misuse of the information.

An efficient method of providing hidden credentials by reusing randomness in a way that does not compromise the security of the system is proposed in [19]. The number of elliptic curve operations required depends only on the number of credentials relevant to a transaction and is constant over a change in policy size or complexity. A monotonic secret splitting scheme is also proposed, where the relevant shares and the corresponding boolean expression are only revealed when relevant pairs of shares are discovered.

[87] introduces the adaptive trust negotiation and access control (ATNAC) framework that addresses the problem of access control in open systems by providing protecting from adversaries who may want to misuse, exhaust or deny service to resources. ATNAC provides a federated security context that allows Grid participants to communicate their security appraisal and make judgments based on collective wisdom and the level of trust between them.

Comparing the Approaches

To assess the current state-of-the art, this section maps the requirements introduced in Section 6.2.3 to three tables to provide a higher-level view of the detailed discussions provided in Section 6.2.3. Each of the approaches have their own drawbacks and advantages and none of the proposals are complete, though current approaches address significant subsets of relevant requirements.

Comparing Policy Languages

Table 6.6 compares the approaches surveyed with respect to the policy language requirements. The comparison is based on the following dimensions: the nature of the policy specification, whether the combination on credentials can be supported by the policy language or not, the existence of multiple authentication, if the protection of sensitive data is provided in the policy level and, finally, if the management of the policy life-cycle is supported.

In the approaches the combination of credentials is considered important and there is a way to express it in the specification language in almost all existing frameworks. However, the authentication of multiple identities is missing in almost all the approaches surveyed.

Also, life-cycle management of policies — i.e., the creation, evolution, and management of policies — is an often overlooked part of policy model design as policies are rarely set in stone when first defined. Such aspect is totally absent in all the current frameworks except Trust-Serv.

There are no existing approaches addressing the trust negotiation cross all the layers of the service based systems with respect to the variety of information discussed earlier. However, all the surveyed approaches address the requirement trust negotiation at one level.

Comparing Credential Management

Table 6.7 compares the approaches surveyed with respect to the credential management requirements. The comparison is based on the following dimensions: the types of disclosure, the automation when tackling the credentials, the discovery of the credential chain and, finally, the ownership of the credential.

The table shows that in the research into service trust negotiation, the trend is towards performing actions while disclosing credentials in an automatic way. Mechanisms for credentials discovery during the negotiation are rare. However, run-time systems should include tools for chain discovery to retrieve at run-time credentials that are not locally cached.

Most of current systems try to integrate the negotiation framework with the exiting tools and systems in order to maximize the control of the security while the data are exchanged.

No existing system addresses how to obtain credentials, assuming that the entity disclosing credentials has its own method to obtain and cache the credentials locally.

Comparing Systems and Strategies

Table 6.8 compares the approaches surveyed with respect to the strategy of negotiation requirements. The comparison is based on the following dimensions: the *dynamicity*, the degree of automation, the presence of the mechanism of privacy protection during the negotiation, the scalability and, finally, the existence of bilateral trust establishment.

The table shows that there is a trend in the strategies to support dynamic and automatic negotiations. Most credentials are not disclosed locally during run-time but can be found dynamically or through combining credentials.

Table 6.6. Comparing Policy Specification Language

Approaches	Trust-Serv	Traust	WS-ABAC	XTAF	PeerTrust	WS-Trust
Specification	Formal	Semi	Formal	semi-formal	Semi	Syntactic
Combination	Yes	Yes	No	Yes	Yes	Yes
Multiple-authentication	No	No	No	No	No	Yes
Sensitive-policy-protection	No	Yes	No	Yes	Yes	Yes
Management-evolution	Yes	No	No	NO	No	No

Table 6.7. Comparing Credential Manager

Approaches	Trust-Serv	Traust	WS-ABAC	XTAF	PeerTrust	WS-Trust
Type-disclosure	actions	Actions	Actions	context+actions	Actions	Actions
Automation	Auto	Auto	Semi	semi	Auto	Semi
Chain discovery	Yes	No	No	No	Yes	Yes
Ownership	No	Integ(Tokens)	Integ(RBAC)	Integ(XACML)	No	INte(Tokens)

Table 6.8. Comparing Strategies

Approaches	Trust-Serv	Traust	WS-ABAC	XTAF	PeerTrust	WS-Trust
Dynamicity	Dynamic	Run-time	No	No	Yes	No
Automatic	Auto	Semi		semi-auto	Auto	semi
Privacy protection mechanism	No	Yes	No	Yes	Yes	Yes
scalability	Yes	Yes	Yes	No	Yes	Yes
bilateral	No	Yes	No	No	Yes	Yes

We found current systems are increasingly aware of protecting sensitive data exchanged during the negotiation with the integrated mechanism taking care of this. Furthermore, bilateral trust creation is considered in the current systems as it is critical that a service provider and customer trust each other.

Scalability is an important evaluation criterion in distributed systems such as service-based applications (SBAs). However, in the presented approaches, the complexity and the consistency of the credential discovery mechanisms are not discussed, which are very important factors in distributed systems. Moreover, the existing approaches are only research paradigms, no standardization is pointed out.

Privacy Negotiation in SBAs

Privacy violation is a serious and pressing problem for Internet users that requires an immediate solution.

Records of negotiation attempts should confidential and private and their creators may not want to the details of these record revealed. In [116], a privacy preserving negotiation learning scheme is introduced, which incorporates secure multiparty computation techniques into negotiation learning algorithms to allow negotiation parties to securely complete the learning process on a union of distributed data sets.

[33] identifies the risks of privacy invasion during the setup of interactive multimedia applications, and introduce three schemes to solve the problem of protecting the user's privacy, with varying degree of complexities. The first approach is based on the use of a trusted third party, a common approach used in Public Switched Telephony Networks (PSTNs). The second approach is based on a trust relationship between communicating parties, and the third approach is uses primitives from the field of secure distributed computation.

[82] has presented the necessity of negotiation about privacy principles in the relationship between a service provider and customer. Modeling the user's individual utility maximization can take into account the multi-dimensionality of privacy; the service provider may wish to reduce the negotiation space in a way that suits the given business scenario. Two new elements were proposed that follow the structure of the current P3P 1.1 grouping mechanisms and allow software-supported negotiations in e-Commerce.

E-commerce systems are increasingly concerned with data protection. They follow a property-based approach to privacy which leads to privacy negotiation and bargaining upon the base of the data subjects' consent. After considering the technological and regulative strategies of protecting consumer privacy, [76] discusses the shortcomings of that approach and claims that, as long as a general privacy culture has not yet evolved in the (web) society, it might conflict with the notion of data protection as a fundamental right.

Due to the automation of infrastructures, both users and services have many agents acting on their behalf. In pervasive systems one of the most problematic concerns arises over the user's right to maintain their privacy.

[18] is focused on instruments to enable and maintain privacy through a subtle fusion of different privacy enabling techniques. The authors present a conceptual privacy enabling architecture of infrastructural pervasive middleware that uses trust management, privacy negotiation, and identity management during the inter-entity communication life-cycle.

In order to take into account the privacy concerns of the individuals, organizations (e.g., Web services) provide privacy policies as promises describing how they will handle the personal data of the individual. However, privacy policies do not convince potential individuals to disclose their personal data, do not guarantee the protection of personal information, and do not specify the way to handle the dynamic environment of the policies. [11] introduces a framework based on an agreement as a solution to these problems. It proposes a *privacy agreement* model that spells out a set of requirements related to the requestor's privacy rights in terms of how the service provider must handle privacy information. It defines two levels in the agreement: 1) the policy level; and 2) the negotiation level. A formal privacy model is described in the policy level to provide upon it a reasoning mechanism for the evolution. The framework supports life-cycle management during the negotiation of the agreement and, hence, the privacy evolution is handled in this level.

6.3 General Observations

6.3.1 QoS Specification Observations

Our survey has uncovered the lack of a well established and standard QoS model for services. In addition, most of the approaches do not offer a rich, extensible, and formal QoS model that includes an extensive categorization of QoS attributes in all service levels. As a result, QSDs and SLAs are populated using many different incompatible QoS models and which lack the richness needed to specify the QoS of many types of services.

Apart from the lack of a standard service QoS model, there is also a lack of well-established and standard QoS meta-models or languages that could be used to specify QSDs and SLAs. The QoS meta-models surveyed either do not have the appropriate formality or richness (or both) to specify quality documents. As a result, QSDs and SLAs are described by many different formalisms of languages or meta-models, all of which are not rich enough.

The above inefficiencies in specifying quality documents limit the fulfillment of the vision of automated and precise QoS-based service matchmaking and selection and QoS-aware service composition and the automation and support of all other activities related to service provisioning.

Hence, we argue that research should be carried out in the development of a standard and rich QoS model that provides an extensive categorization of QoS attributes in all service levels. Moreover, this QoS model should be extensible so as to allow the addition of new quality dimensions when it is needed (e.g., for a new application domain). Last but not least, this standard QoS model should be semantically enriched (i.e., formal) in order to be machine-processable and machine-interpretable.

Such a comprehensive QoS model for services requires a suitable formal QoS language or meta-model to be used in complex service-based applications, in which services can be invoked and composed with variable QoS profiles. Such a language should be capable of expressing QoS capabilities and SLAs by using functions, operators and comparison predicates on QoS metrics and attributes. It should also allow the description of composition rules for every possible combination of composition constructs and QoS metrics. Moreover, it should allow the description of different QoS offerings for the same functional offering of a service; i.e., it should be able to describe classes of service.

6.3.2 QoS Negotiation Observations

We identify two main streams for short-term research on service QoS negotiation. First, we underline the issue of automated SLA creation in service compositions. The review shows that most of the current work in this field concerns the negotiation between a service consumer and a service provider or the set of providers of functionally equivalent services. Proposals for managing complex one to many negotiations with services involved in the same service composition are still at their infancy and need further development. Second, research efforts should be devoted to the analysis of innovative negotiation strategies explicitly tailored to the requirements of service-based applications. Currently, the participants of QoS negotiations in service-based applications adopt state-of-the-art strategies, drawn from research on agent-based computing. We argue that more efficient and flexible solutions to the negotiation problem become feasible when negotiation strategies take into account the features of negotiation objects and protocols in service-based applications.

QoS negotiation can also be used to extend the capabilities of service composition tools. QoS agreement gives of course best results when global optima have been achieved. This is difficult, because complex QoS expressions can give rise to optimization functions which are difficult or impossible to treat mathematically. Approximations and heuristics, such as the ones achieved with genetic algorithms, simulated annealing or planning strategies, appear to be the only feasible resort at this moment. Additionally, if the architecture allows for dynamic re-negotiation (within the same service, for which services should offer different QoS classes, or selecting another service), the cost of negotiation, e.g., planning, equation solving, etc., should be evaluated against the expected gain achieved with the new service.

In Grid computing, we have observed the QoS models and SLAs used for resource provisioning in current Grids are similar to approaches used in service-based applications. Some of the models and implemented tools are even used in both fields (an example is WS-Agreement). Concerning the solutions available for Grid computing, we can state that the adopted and proof-of-concept implementations are still premature. We found promising theoretical approaches that fit SLA usage to current Grid systems, but there is no common mechanism for SLA advertisement and discovery. WS-Agreement seems to be a good candidate, but existing solutions using this form still cannot interoperate. Regarding SLA negotiation, WS-Agreement still cannot deliver the solution: in most cases it can only be used for a simple offer-accept interaction. The future directions should identify a commonly accepted approach for service advertisement and discovery. Recent experiences both in the service-oriented and Grid communities should be taken into account in order to arrive at a widely accepted and interoperable solution.

References

1. The GEMSS project: Grid-enabled medical simulation services, EU IST project, ist-2001-37153, `http://www.gemss.de/`
2. OGF grid resource allocation agreement protocol working group website: `https://forge.gridforum.org/sf/projects/graap-wg`
3. OGF grid scheduling architecture research group website: `https://forge.gridforum.org/sf/projects/gsa-rg`
4. Interactive Access Control for Web Services. Kluwer (2004)
5. Ajayi, O., Sinnott, R., Stell, A.: Dynamic trust negotiation for flexible e-health collaborations. In: Proceedings of the 15th ACM Mardi Gras conference (MG), pp. 1–7. ACM Press, New York (2008)
6. Arenas, A., Wilson, M., Matthews, B.: On trust management in grids. In: Proceedings of the 1st international conference on Autonomic computing and communication systems (Autonomics'07), pp. 1–7. ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2007)
7. Artz, D., Gil, Y.: A survey of trust in computer science and the semantic web. Web Semant. 5(2), 58–71 (2007)
8. S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, H. Prafullchandra, C. von Riegen, D. Roth, J. Schlimmer, C. Sharp, J. Shewchuk, A. Vedamuthu, C.Ü. Yal D. Orchard. Web Services Policy Framework (WS-Policy). IBM (March 2006)
9. Becker, M., Sewell, P.: Cassandra: Distributed access control policies with tunable expressiveness. In: POLICY '04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA, p. 159. IEEE Computer Society (2004)
10. Benbernou, S., Brandic, I., Cappiello, C., Carro, M., Comuzzi, M., Kertész, A., Kritikos, K., Parkin, M., Pernici, B., Plebani, P.: A Survey on Service Quality Description. ACM Computing Surveys, submitted (2009)
11. Benbernou, S., Meziane, H., Li, Y.H., Hacid, M.-S.: A privacy agreement model for web services. In: IEEE SCC, pp. 196–203. IEEE Computer Society Press (2007)

12. Benkner, S., Engelbrecht, G., Middleton, S.E., Brandic, I., Schmidt, R.: End-to-End QoS support for a medical grid service infrastructure. New Generation Computing, Special Issue on Life Science Grid Computing (2007)
13. Bertino, E., Ferrari, E., Squicciarini, A.: X-TNL: An XML-based language for trust negotiations. In: POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, page 81, Washington, DC, USA, IEEE Computer Society (2003)
14. Bertino, E., Ferrari, E., Squicciarini, A.: Trust negotiations: concepts, systems, and languages. Computing in Science and Engineering 6(4), 27–34 (2004)
15. Bertino, E., Ferrari, E., Squicciarini, A.: Trust negotiations: Concepts, systems, and languages. Computing in Science and Engineering 06(4), 27–34 (2004)
16. Bertino, E., Ferrari, E., Squicciarini, A.C.: Trust-X: A peer-to-peer framework for trust establishment. IEEE Transactions on Knowledge and Data Engineering, TKDE 16(7), 827–842 (2004)
17. Bhargav-Spantzel, A., Squicciarini, A.C., Bertino, E.: Trust negotiation in identity management. IEEE Security and Privacy 5(2), 55–63 (2007)
18. Blazic, A.J., Dolinar, K., Porekar, J.: Enabling privacy in pervasive computing using fusion of privacy negotiation, identity management and trust management techniques. In: First International Conference on the Digital Society (ICDS 2007), Guadeloupe, French Caribbean, 2-6 January 2007, Springer, Heidelberg (2007)
19. Bradshaw, R.W., Holt, J.E., Seamons, K.E.: Concealing complex policies with hidden credentials. In: Proceedings of the 11th ACM conference on Computer and communications security CCS '04, Washingtion, DC, USA, pp. 146–157. ACM Press (2004)
20. Brandic, I., Pllana, S., Benkner, S.: An approach for the high-level specification of QoS-aware grid workflows considering location affinity. Scientific Programming Journal 14(3-4), 231–250 (2006)
21. Brandic, I., Pllana, S., Benkner, S.: Specification, planning, and execution of QoS-aware grid workflows within the Amadeus environment. Concurrency and Computation: Practice and Experience 20(4), 331–345 (2008)
22. Chhetri, M.B., Lin, J., Goh, S., Yan, J., Zhang, J.Y., Kowalczyk, R.: A coordinated architecture for the Agent-based Service Level agreement Negotiation of Web service composition. In: Proc. 2006 Australian Software Engineering Conference, ASWEC'06 (2006)
23. Chhetri, M.B., Lin, J., Goh, S., Zhang, J.Y., Kowalczyk, R., Yan, J.: A coordinated architecture for the agent-based service level agreement negotiation ofweb service composition. In: ASWEC '06: Proceedings of the Australian Software Engineering Conference, Washington, DC, USA, pp. 90–99. IEEE Computer Society (2006)
24. Chiu, D.K.W., Cheung, S.C.: Patrick C.K. Hung, and Ho fung Leung. Facilitating e-negotiation processes with semantic web technologies. In: HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 1, Washington, DC, USA, p. 36.1. IEEE Computer Society (2005)
25. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht (2000)
26. Comuzzi, M., Pernici, B.: An architecture for flexible Web service QoS negotiation. In: Proceedings of the 9th IEEE Enterprise Computing Conference, Enschede, The Netherlands (2005)

27. Cortés, A.R., Martín-Díaz, O., Toro, A.D., Toro, M.: Improving the Automatic Procurement of Web Services Using Constraint Programming. Int. J. Cooperative Inf. Syst. 14(4), 439–468 (2005)
28. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 153–183. Springer, Heidelberg (2002)
29. Degwekar, S., Su, S.Y.W., Lam, H.: Constraint specification and processing in web services publication and discovery. In: ICWS, pp. 210–217. IEEE Computer Society (2004)
30. Di Nitto, E., Di Penta, M., Gambi, A., Ripa, G., Villani, M.L.: Negotiation of Service Level Agreements: An Architecture and a Search-Based Approach. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 295–306. Springer, Heidelberg (2007)
31. Dobson, G., Lock, R., Sommerville, I.: QoSOnt: a QoS ontology for service-centric systems. In: EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, Porto, Portugal, pp. 80–87. IEEE Computer Society (2005)
32. Dragoni, N., Massacci, F.: Security-by-contract for web services. In: Proceedings of the 2007 ACM workshop on Secure web services (SWS '07), New York, NY, USA, pp. 90–98. ACM (2007)
33. El-Khatib, K., von Bochmann, G.: Protecting the privacy of user's qos preferences for multimedia applications. In: Proceedings of the 2nd ACM international workshop on Wireless multimedia networking and performance modeling (WMuNeP '06), New York, NY, USA, pp. 35–42. ACM (2006)
34. Faratin, P., Sierra, C., Jennings, N.R.: Negotiation decision functions for autonomous agents. Int. Journal of Robotics and Autonomous Systems 23(3-4), 159–182 (1998)
35. Faratin, P., Sierra, C., Jennings, N.R.: Negotiation decision functions for autonomous agents. Int. Journal of Robotics and Autonomous Systems 24(3-4), 159–182 (1998)
36. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed nist standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (2001)
37. FIPA. FIPA standard status specifications, http://www.fipa.org/repository/standardspecs.html
38. Document Title Fipa. FIPA Communicative Act Library Specification (2003)
39. Frikken, K., Atallah, M., Li, J.: Hidden access control policies with hidden credentials. In: Proceedings of the 2004 ACM workshop on Privacy in the electronic society (WPES '04), New York, NY, USA, pp. 27–28. ACM (2004)
40. Frølund, S., Koistinen, J.: Quality of services specification in distributed object systems design. In: COOTS'98: Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems (COOTS), 5(4), pp. 179–202 (1998)
41. Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A.: Contemporary Web Service Discovery Mechanisms. Journal of Web Engineering 5(3), 265–290 (2006)
42. Gavriloaie, R., Nejdl, W., Olmedilla, D., Seamons, K., Winslett, M.: No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web (2004)

43. Giallonardo, E., Zimeo, E.: More semantics in QoS matching. In: International Conference on Service-Oriented Computing and Applications, Newport Beach, CA, USA, pp. 163–171. IEEE Computer Society (2007)
44. Gimpel, H., Ludwig, H., Dan, A., Kearney, B.: PANDA: Specifying policies for automated negotiations of service contracts. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 287–302. Springer, Heidelberg (2003)
45. Grandison, T., Sloman, M.: A Survey of Trust in Internet Applications. IEEE Communications Surveys and Tutorials 3(4), http://www.comsoc.org/livepubs/surveys/public/2000/dec/index.html
46. He, Y., Zhu, M.: A complete and efficient strategy based on Petri Nets in automated trust negotiation. In: Proceedings of the 2nd international conference on Scalable information systems (InfoScale), pp. 1–7, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2007)
47. Hobbs, J.R., Pan, F.: An ontology of time for the semantic web. ACM Trans. Asian Lang. Inf. Process. 3(1), 66–85 (2004)
48. Huai, J., Sun, H., Hu, C., Zhu, Y., Liu, Y., Li, J.: Rost: Remote and hot service deployment with trustworthiness in crown grid. Future Generation Computer Systems 23(6), 825–835 (2007)
49. Irwin, K., Yu, T.: Preventing attribute information leakage in automated trust negotiation. In: Proceedings of the 12th ACM conference on Computer and communications security (CCS '05), New York, NY, USA, pp. 36–45. ACM (2005)
50. Jennings, N.R., Faratin, P., Lomuscio, A.R., Parsons, S., Wooldridge, M.J., Sierra, C.: Automated negotiation: Prospects, methods and challenges. Group Decision and Negotiation 10(2), 199–215 (2001)
51. Keller, A., Ludwig, H.: The WSLA framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management 11(1), 57–81 (2003)
52. Koshutanski, H., Massacci, F.: Interactive credential negotiation for stateful business processes. In: iTrust, pp. 256–272 (2005)
53. Kritikos, K., Plexousakis, D.: Semantic qos metric matching. In: ECOWS '06: Proceedings of the European Conference on Web Services, Zurich, Switzerland, pp. 265–274. IEEE Computer Society (2006)
54. Kritikos, K., Plexousakis, D.: Semantic QoS-based web service discovery algorithms. In: ECOWS '07: Proceedings of the Fifth European Conference on Web Services, Halle, Germany, pp. 181–190. IEEE Computer Society (2007)
55. Kritikos, K., Plexousakis, D.: Requirements for QoS-based Web Service Description and Discovery. IEEE Transactions on Services Computing, accepted (2009)
56. Lai, G., Li, C., Sycara, K., Giampapa, J.A.: Literature review on multi-attribute negotiations. Technical Report CMU-RI-TR-04-66, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (December 2004)
57. Lamparter, S., Luckner, S., Mutschelr, S.: Formal specification of Web service contracts for automated contracting and monitoring. In: Proceedings of the 40th Hawaii International Conference on System Sciences, Honolulu, Hawaii, pp. 63–73 (2007)

58. Lamparter, S., Luckner, S., Mutschler, S.: Formal specification of web service contracts for automated contracting and monitoring. In: HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences, Washington, DC, USA, IEEE Computer Society (2007)
59. Lee, A., Winslett, M., Basney, J., Welch, V.: Traust: a trust negotiation-based authorization service for open systems. In: SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies, New York, ACM (2006)
60. Lee, A.J., Winslett, M., Basney, J., Welch, V.: The Traust authorization service. ACM Trans. Inf. Syst. Secur. 11(1), 1–33 (2008)
61. Li, J., Huai, J., Xu, J., Zhu, Y., Xue, W.: Tower: Practical trust negotiation framework for grids. In: Second IEEE International Conference on e-Science and Grid Computing (e-Science'06), 0:26 (2006)
62. Li, N., Mitchell, J.: Rt: A role-based trust-management framework. In: The Third DARPA Information Survivability Conference and Exposition (DISCEX III) (April 2003)
63. Liu, Y., Ngu, A.H.H., Zeng, L.: QoS computation and policing in dynamic web service selection. In: Feldman, S.I., Uretsky, M., Najork, M., Wills, C.E. (eds.) WWW (Alternate Track Papers & Posters), pp. 66–73. ACM Press, New York (2004)
64. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreement (WSLA) Language Specification. Technical report, IBM Corporation (2003)
65. Mani, A., Nagarajan, A.: Understanding quality of service for web services (2002), http://www-128.ibm.com/developerworks/library/ws-quality.html
66. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L.: Wonderweb deliverable d17. the wonderweb library of foundational ontologies and the dolce ontology
67. Maximilien, E.M., Singh, M.P.: Conceptual model of web service reputation. SIGMOD Rec. 31(4), 36–41 (2002)
68. Maximilien, E.M., Singh, M.P.: A framework and ontology for dynamic web services selection. IEEE Internet Computing 8(5), 84–93 (2004)
69. Mbanaso, U.M., Cooper, G.S., Chadwick, D.W., Proctor, S.: Privacy preserving trust authorization framework using XACML. In: 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks(WoWMoM'06), pp. 673–678. IEEE (2006)
70. Menascé, D., Dubey, V.: Utility-based QoS brokering in service oriented architectures. In: Proceedings of the 2007 International Conference on Web services (2007)
71. Mukhi, N.K., Plebani, P.: Supporting policy-driven behaviors in Web services: experiences and issues. In: Proceedings of the 2nd International Conference on Service Oriented Computing, New York, NY (2004)
72. Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., Granqvist, H.: WS-Trust specification. Technical report. OASIS Working Draft (2007), http://www.ibm.com/developerworks/webservices/library/specification/ws-trust
73. Ncho, A., Aimeur, E.: Building a multi-agent system for automated negotiation in Web service applications. In: Proc. of AAMAS'04 (2004)
74. Oldham, N., Verma, K., Sheth, A., Hakimpour, F.: Semantic WS-agreement partner selection. In: WWW '06: Proceedings of the 15th International con-

ference on World Wide Web, Edinburgh, Scotland, pp. 697–706. ACM Press, New York (2006)

75. Oldham, N., Verma, K., Sheth, A., Hakimpour, F.: Semantic WS-agreement partner selection. In: WWW '06: Proceedings of the 15th international conference on World Wide Web, New York, NY, USA, pp. 697–706. ACM (2006)
76. Oliver-Lalana, A.D.: Consent as a threat. A critical approach to privacy negotiation in e-commerce practices. In: Katsikas, S.K., López, J., Pernul, G. (eds.) TrustBus 2004. LNCS, vol. 3184, pp. 110–119. Springer, Heidelberg (2004)
77. Olmedilla, D., Lara, R., Polleres, A., Lausen, H.: Trust Negotiation for Semantic Web Services. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 81–95. Springer, Heidelberg (2005)
78. Olson, L.E., Rosulek, M.J., Winslett, M.: Harvesting credentials in trust negotiation as an honest-but-curious adversary. In: Proceedings of the 2007 ACM workshop on Privacy in electronic society (WPES '07), New York, NY, USA, pp. 64–67. ACM (2007)
79. Olson, L., Winslett, M., Tonti, G., Seeley, N., Uszok, A., Bradshaw, J.: Trust negotiation as an authorization service forweb services. In: ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops, Washington, DC, USA, IEEE Computer Society (2006)
80. O'Sullivan, J., Edmond, D., ter Hofstede, A.H.M.: Formal description of nonfunctional service properties. Technical report, Queensland University of Technology (2005)
81. Bonatti, P., Samarati, P.: Regulating service access and information release on the web. In: CCS '00: Proceedings of the 7th ACM conference on Computer and communications security, New York, NY, pp. 134–143. ACM (2000)
82. Preibusch, S.: Implementing privacy negotiations in E-commerce. In: Zhou, X., Li, J., Shen, H.T., Kitsuregawa, M., Zhang, Y. (eds.) APWeb 2006. LNCS, vol. 3841, pp. 604–615. Springer, Heidelberg (2006)
83. Ran, S.: A model for web services discovery with QoS. SIGecom Exch. 4(1), 1–10 (2003)
84. Ran, S.: A model for web services discovery with qos. SIGecom Exch. 4(1), 1–10 (2003)
85. Redman, T.C.: Data Quality for the Information Age (Foreword By - Blanton Godfrey, A.). Artech House, Inc., Norwood (1997)
86. Ruohomaa, S., Kutvonen, L.: Trust management survey. In: Herrmann, P., Issarny, V., Shiu, S.C.K. (eds.) iTrust 2005. LNCS, vol. 3477, pp. 77–92. Springer, Heidelberg (2005)
87. Ryutov, T., Zhou, L., Neuman, C., Leithead, T., Seamons, K.E.: Adaptive trust negotiation and access control. In: Proceedings of the tenth ACM symposium on Access control models and technologies (SACMAT '05), New York, NY, USA, pp. 139–146. ACM (2005)
88. Sabata, B., Chatterjee, S., Davis, M., Sydir, J.J., Lawrence, T.F.: Taxonomy for QoS Specifications. In: Third International Workshop on Object-Oriented Real-Time Dependable Systems, 1997. Proceedings, 5-7 Feb. 1997, pp. 100–107 (1997)
89. Sahai, A., Durante, A., Machiraju, V.: Towards Automated SLA Management for Web Services. Technical Report HPL-2001-310, HP Laboratories, Palo Alto, CA (July 2002)

90. Sakellariou, R., Yarmolenko, V.: On the flexibility of WS-Agreement for job submission. In: Proceedings of the 3rd International Workshop on Middleware for Grid Computing, MGC'05 (2005)
91. Sakellariou, R., Yarmolenko, V.: Job Scheduling on the Grid: Towards SLA-Based Scheduling. In: High Performance Computing and Grids in Action (March 2008)
92. Seamons, K.E., Winslett, M., Yu, T., Yu, L., Jarvis, R.: Protecting privacy during on-line trust negotiation. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 129–143. Springer, Heidelberg (2003)
93. Seidel, J., Wäldrich, O., Ziegler, W., Wieder, P., Yahyapour, R.: a survey. Using SLA for resource management and scheduling. Technical report, TR-0096, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence (August. 2007)
94. Shen, H., Hong, F.: An attribute-based access control model for web services. In: Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2006), pp. 74–79 (2006)
95. Skogsrud, H., Benatallah, B., Casati, F.: Trust-Serv: Model-driven lifecycle management of trust negotiation policies for web services. In: Proc. 13th World Wide Web Conf. (May 2004)
96. Sommerville, I.: Software Engineering, 4th edn. Addison-Wesley, Reading (1992)
97. Squicciarini, A., Bertino, E., Ferrari, E., Paci, F., Thuraisingham, B.: PP-trust-X: A system for privacy preserving trust negotiations. ACM Transactions on Information and System Security (TISSEC) 10(3), 12 (2007)
98. Strong, D.M., Lee, Y.W., Wang, R.Y.: 10 potholes in the road to information quality. Computer 30(8), 38–46 (1997)
99. Sycara, K., et al.: OWL-S 1.0 Release. OWL-S Coalition (2003), http://www.daml.org/services/owl-s/1.0/
100. The OASIS Group. Quality model for web services. Technical report, The Oasis Group (September 2005)
101. The OMG Group. UML^{TM} Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. Technical Report ptc/2005-05-02, The OMG Group (May 2005)
102. Tian, M., Gramm, A., Nabulsi, M., Ritter, H., Schiller, J., Voigt, T.: Qos integration in web services. In: Gesellschaft fur Informatik DWS 2003, Doktorandenworkshop Technologien und Anwendungen von XML (October 2003)
103. Tosic, V., Esfandiari, B., Pagurek, B., Patel, K.: On requirements for ontologies in management of web services. In: Bussler, C.J., McIlraith, S.A., Orlowska, M.E., Pernici, B., Yang, J. (eds.) CAiSE 2002 and WES 2002. LNCS, vol. 2512, pp. 237–247. Springer, Heidelberg (2002)
104. Tosic, V., Pagurek, B., Patel, K.: WSOL - A language for the formal specification of classes of service for web services. In: Zhang, L.-J. (ed.) ICWS, June 2003, pp. 375–381. CSREA Press (2003)
105. Tosic, V., Patel, K., Pagurek, B.: WSOL - web service offerings language. In: Bussler, C.J., McIlraith, S.A., Orlowska, M.E., Pernici, B., Yang, J. (eds.) CAiSE 2002 and WES 2002. LNCS, vol. 2512, pp. 57–67. Springer, Heidelberg (2002)

106. Tsesmetzis, D.T., Roussaki, I.G., Papaioannou, I.V., Anagnostou, M.E.: Qos awareness support in web-service semantics. In: AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services, Guadeloupe, French Caribbean, pp. 128–134. IEEE Computer Society (2006)
107. Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S., Lott, J.: Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In: POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA, IEEE Computer Society (2003)
108. Wang, X., Vitvar, T., Kerrigan, M., Toma, I.: A qoS-aware selection model for semantic web services. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 390–401. Springer, Heidelberg (2006)
109. Winsborough, W.H., Li, N.: Safety in automated trust negotiation. ACM Transactions on Information and System Security (TISSEC) 9(3), 352–390 (2006)
110. Winslett, M., Yu, T., Seamons, K.E., Hess, A., Jacobson, J., Jarvis, R., Smith, B., Yu, L.: The TrustBuilder architecture for trust negotiation. IEEE Internet Computing 6(6), 30–37 (2002)
111. Winslett, M., Zhang, C.C., Bonatti, P.A.: Peeraccess: a logic for distributed authorization. In: CCS '05: Proceedings of the 12th ACM conference on Computer and communications security, pp. 168–179, New York, NY, USA. ACM (2005)
112. WS-AGREEMENT. WS-Agreement Framework (September 2003), https://forge.gridforum.org/projects/graap-wg
113. Yan, J., Zhang, J.Y., Chhetri, M.B., Lin, J., Goh, S., Kowalczyk, R.: Towards autonomous service level agreement negotiation for adaptvive service composition. In: Proc. 10th Int. Conf. on Computer Supported Cooperative Work in Design (2006)
114. Yu, T., Winslett, M.: Policy migration for sensitive credentials in trust negotiation. In: Proceedings of the 2003 ACM workshop on Privacy in the electronic society (WPES '03), New York, NY, USA, pp. 9–20. ACM (2003)
115. Yu, T., Winslett, M., Seamons, K.E.: Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. ACM Trans. Inf. Syst. Secur. 6(1), 1–42 (2003)
116. Zhang, S., Makedon, F.: Privacy preserving learning in negotiation. In: Proceedings of the 2005 ACM symposium on Applied computing (SAC '05), New York, NY, USA, pp. 821–825. ACM (2005)
117. Zhou, C., Chia, L.-T., Lee, B.-S.: Daml-qos ontology for web services. In: ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04), San Diego, CA, USA, pp. 472–479. IEEE Computer Society (2004)

7

Analytical Quality Assurance

Andreas Metzger[1], Salima Benbernou[2], Manuel Carro[3], Maha Driss[4], Gabor Kecskemeti[5], Raman Kazhamiakin[6], Kyriakos Krytikos[7], Andrea Mocci[8], Elisabetta Di Nitto[8], Branimir Wetzstein[9], and Fabrizio Silvestri[10]

[1] Universität Duisburg-Essen, Germany
[2] Université Claude Bernard, Lyon 1, France
[3] Universidad Politécnica de Madrid, Spain
[4] Institut National de Recherche en Informatique et Automatique (INRIA)
MTA Computer & Automation Research Institute (MTA-SZTAKI), Budapest, Hungary
[5] Fondazione Bruno Kessler (FBK), Trento, Italy
[6] University of Crete, Greece
[7] Politecnico di Milano, Italy
[8] University of Stuttgart, Germany
[9] Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy

Chapter Overview. As we described in Chapter 1, Services are often provisioned within short-term, volatile and highly dynamic (business) processes. These processes are designed in an abstract manner and when instantiated can involve service providers not known of during the design time of the service-based application. Thus, different from traditional software systems, service-based applications require the composition and coordination of services within highly distributed environments, cutting across the administrative boundaries of various organizations.

This chapter provides a review of quality contracts, or more generally, those parts of Service Level Agreements (SLAs) which deal with statements about the services quality levels on which the service requestor and the providers have reached an agreement. Aspects of the contracts, such as the identification of parties, legal obligations and penalties for contract violation, are not covered by this chapter.

7.1 Motivation

To guarantee the desired end-to-end quality of those service-based applications, contracts between the service providers and the service requestors (also known as service consumers) on quality aspects of services must be established [30]. In general, a contract is a formal agreement between two or more parties to create mutual business relations or legal obligations. Contracts can have different parts, such as the definition of business partners, the specifica-

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 209–270, 2010.

tion of functional obligations, and quality, price, and penalties related with the object of the agreement.

Based on the general life-cycle of electronic contracts [109] [151], three main activities relevant for quality contracts within service-based applications can be identified:

- *Quality definition*: In electronic contracting, the contract definition activity concerns the establishment of a model or language for the definition of contract terms, which is understood and shared by the contracting parties. This model or language then is used to instantiate an actual contract (e.g., a SLA) that reflects the domain dependent interests of providers and consumers.
- *Quality negotiation*: Establishment of an electronic contract concerns the set of tasks required for defining an actual contract (e.g., SLA) based on the model or language for the definition of contract terms (see above). This may involve the selection of the service provider (the contract partner) among a set of potential providers, the negotiation of the contract terms between the selected provider and the service consumer, and the agreement to the contract terms.
- *Quality assurance*: Contract enactment in electronic contracting concerns tasks for assuring the satisfaction of the contracts. In the case of quality contracts, this implies assuring that the quality levels negotiated and agreed between the service provider and the service requestor are met. In the dynamic setting of service-based applications, quality assurance can reveal that there is a deviation from the expected quality, thus necessitating an adaptation of the application. How such an adaptation can be achieved is described in Chapter 5.

This chapter addresses the final activity in electronic quality contracting, namely quality assurance (the first two activitues are addressed in Chapter 6). To achieve the desired quality of a service-based application (i.e, for contract enactment), two complementary kinds of techniques and methods can be employed: *constructive* and *analytical* quality assurance techniques and methods. The goal of constructive quality assurance techniques and methods is to prevent the introduction of faults (or defects) while the artifacts are created. Examples for such techniques include code generation (model-driven development), development guidelines, as well as templates. The goal of analytical quality assurance techniques and methods is to uncover faults in the artifacts after they have been created. Examples for analytical quality assurance techniques are reviews and inspections, formal correctness proofs, testing, as well as monitoring.

This chapter will cover the state of the art in analytical quality assurance techniques and methods for service-based applications by providing a comprehensive literature review in analytical quality assurance for service-based applications. Furthermore, this chapter will identify gaps in the state of the art in order to identify open research issues that should be addressed in future research.

The literature review covers three major classes of approaches for analytical quality assurance in service-based applications: (i) *Testing*, the goal of which is to (systematically) *execute* services or service-based applications with predefined inputs in order to uncover failures, (ii) *Monitoring*, which observes services or service-based applications as well as their context during their *current* execution, (iii) *Static Analysis*, the aim of which is to systematically *examine* (without execution) an artifact (e.g., a service specification) in order to determine certain properties or to ascertain that some predefined properties are met.

The literature review has been carried out by members of the S-Cube network of excellence, to whom we extend our gratitude.

7.2 Review Methodology

The review of relevant research in the fields of quality asurance in service-based applications has followed a systematic approach. Both general conference and journals on services and software engineering have been covered, and in addition, special research area publication sites have also been covered. Specifically, we reviewed work, starting from year 2000, in the following publication forums:

- **Conference Proceedings**: Int. Conf. on Software Engineering (ICSE), Int. Conf. on Web Services (ICWS), Int. Services Computing Conf. (SCC), European Conf. on Web Services (ECOWS), Int. Enterprise Distributed Object Computing Conf. (EDOC), World Wide Web Conf. (WWW), Int. Conf. on Service Oriented Computing (ICSOC), Int. Conf. on Business Process Management (BPM), Int. Conference on Software Engineering (ICSE), International Symposium on Software Testing and Analysis (ISSTA), International Symposium on Software Testing and Analysis (ISSTA)
- **Academic Journals and Magazines**: IEEE TSE, ACM TOSEM
- **Digital libraries**: ACM Digital library on a keyword basis (e.g., SOA, SBS, SOA and monitoring, verification, analysis), SCOPUS, INSPEC

With respect to monitoring in SOA, the presented survey is based on and extends a previous work in [56]. In that survey the authors concentrated on run-time monitoring of Web services and service compositions. The reviewed work presents both the research approaches towards monitoring of Web services and Web service-based compositions. The presented survey follows this approach and extends the list of relevant work with recent advances in this area.

Given the very wide field of analysis, we have chosen to narrow the selection of the survey in this chapter to these papers which are more related with non-functional properties such as performance prediction, structural complexity of service compositions, etc.

Other contributions for the literature review have been derived from the expertise of the different partners that contributed to this chapter. These include relevant work published in domain specific conferences and workshops proceedings or in specialized academic journals, all of which fall outside the aforementioned list of academic publications systematically reviewed.

7.3 Fundamentals

In order to structure the survey results, we sub-divide the analytical quality assurance techniques and methods into the three major classes: *testing*, *monitoring* and *static analysis*. These classes have been proposed in the software quality assurance literature (e.g., see [97, 93, 103, 55]) and have been used in a recent overview of quality assurance approaches for service-based applications (cf. [12]).

Figure 7.1 provides an overview of these classes and their sub-classes which are explained in the following sub-sections.

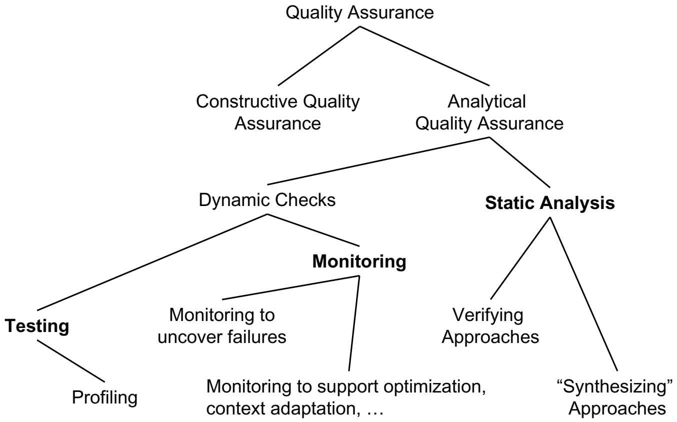

Quality Assurance
Constructive Quality Assurance
Analytical Quality Assurance
Dynamic Checks
Static Analysis
Monitoring
Testing
Monitoring to uncover failures
Verifying Approaches
Profiling
Monitoring to support optimization, context adaptation, …
"Synthesizing" Approaches

Fig. 7.1. Overview of Quality Assurance Approaches

Testing

The goal of testing is to (systematically) *execute* services or service-based applications[10] in order to uncover failures (cf. [97, 93, 103, 55]).

[10] In the remainder of this part of the chapter, we use the term service-based application as a synonym for service composition.

During testing, the service or service-based application which is tested is fed with concrete inputs and the produced outputs are observed. The observed outputs can deviate from the expected outputs with respect to functionality as well as quality of service (e.g., performance or availability). When the observed outputs deviate from the expected outputs, a failure of the service or the service-based application is uncovered.

Failures can be caused by faults (or defects) of the test object. Examples for faults are a wrong exit condition for a loop in the software code that implements a service, or a wrong order of the service invocations in a BPEL specification. Finding such faults typically is not part of the testing activities but is the aim of *debugging* (e.g., cf. [93, 55]).

A special case of testing is profiling. During profiling, a service or a service-based application can be systematically executed in order to determine specific properties. As an example, during profiling the execution times of individual services in a service composition could be measured for 'typical' or 'extreme' inputs in order to identify optimization potentials.

Testing cannot guarantee the absence of faults, because it is infeasible (except for trivial cases) to test all potential concrete inputs of a service or service-based application. As a consequence, a sub-set of all potential inputs has to be determined for testing (e.g., cf. [103]). The quality of the tests strongly depends on how well this sub-set has been chosen. Ideally this sub-set should include concrete inputs that are representative for all potential inputs (even those which are not tested) and it should include inputs that – with high probability – uncover failures. However, in cases where choosing such an ideal sub-set typically is infeasible, it is important to employ other quality assurance techniques and methods which complement testing.

Monitoring

Monitoring observes services or service-based applications during their *current* execution, i.e. during their actual use or operation (cf., [37]). In addition, the context of a service or a service-based application can be monitored. This context can include other systems, the execution platform (hardware, operating systems, etc.) and the physical environment (e.g., sensors or actuators).

Monitoring can address different goals. As an example, monitoring techniques can be employed to support the optimization of a service-based application during run-time. Further, monitoring can be used to enable the context-driven run-time adaptation of a service-based application. Also, monitoring may be used to uncover failures during the current execution of a service or service-based application.

In contrast to testing and static analysis, which aim at providing more or less general statements about services or service-based applications, monitoring always provides statements about their current execution (i.e., about current execution traces). Thereby, monitoring can uncover failures which have escaped testing, because the concrete input that lead to the current execution trace might have never been tested.[11] Also, monitoring can uncover faults which have escaped static analysis, because static analysis might have abstracted from a concrete aspect which was relevant during the current execution. Monitoring therefore provides a complementary measure to ensure the quality of a service-based application and thus "can be used to provide additional defense against catastrophic failure" [37].

Static Analysis

The aim of static analysis (e.g., see [44, 55]) is to systematically *examine* an artifact in order to determine certain properties or to ascertain whether some predefined properties are met. In the remainder of this chapter, we refer to the first kind of approaches as *synthesis* approaches and to the latter kind of approaches as *verification* approaches. Analysis can be applied at several stages in the development cycle, and therefore examples for artifacts which can be subject to analysis include requirement documents, design specifications, interface descriptions, and code.

Examples of static analysis include formal techniques and methods, such as data flow analysis, model checking, execution in abstract domains, symbolic execution, type checking, and correctness proofs, which are all usually characterized because they compute properties that are in many cases approximations of the more concrete properties, but which, in this case, are safe, in the sense that the lack of accuracy must not lead to an error for the intended use of the analysis. Informal approaches, such as reviews, walkthroughs, and inspections, are as well examples of static analysis.

In contrast to testing (or monitoring), where individual executions of the services or service-based applications are examined, analysis can examine classes of executions [55]. Thus, analysis can lead to more universal statements about the properties of the artifacts than testing (or monitoring).

In order to achieve these more universal statements, static analysis – unlike testing or monitoring – does not execute the artifacts which are being examined, since termination (which is theoretically ensured when the system has a finite state space) is usually a necessary condition for a successful analysis. However, systems may have a state space so large (or infinite) as to make traversing it unfeasible. In those cases static analysis resorts to working with *safe approximations* of the actual system semantics, which makes the system actually under analysis effectively finite, but different from the initial one.

Those approximations can be very sophisticated and take the form of, e.g., relations between inputs and outputs which approximate the system behavior in the domain of the analysis. When these approximations capture the properties of interest faithfully enough, then the results, even if not as accurate as they could be, are useful – and correct. Yet, as approximations might abstract

[11] As explained above, only a sub-set of all potential inputs can be tested.

away from some relevant concrete details, aspects might be overlooked [55] or simply not be captured faithfully enough. Thus static analysis can complement the other classes of quality assurance techniques and methods but typically will not be enough, if used in isolation, in order to give a complete picture of the whereabouts of the execution of a computational system.

7.4 Classification Framework

The papers which are surveyed in this section are categorized in order to understand common concepts and thus identify potential gaps and overlaps (see Section 7.7.3). A number of "dimensions" are used for this categorization and constitute our classification framework.

During the process of surveying the papers and based on discussions amongst S-Cube reseaechers, this framework has been continuously evolved in order to cover all relevant dimensions.

The "dimensions" of this classification framework are described in the following sub-sections.

Major Class of Quality Assurance Technique or Method (*Class*)

As has been described in Section 7.3, we will distinguish between three major classes of quality assurance techniques and methods for service-based applications: testing, monitoring and static analysis. For an individual technique it can well be possible that it will be classified to fall into more than one of these major classes.

Quality Characteristics Which Are Addressed (*Quality*)

Different kinds of quality characteristics – also known as QoS dimensions, quality attributes or quality parameters – can be addressed by the quality assurance techniques and methods.

An important quality characteristics is functional *correctness*, i.e. assuring that the functionality expected from the service or service-based application is met.[12]

Other quality characteristics that are relevant for service-based applications include performance, privacy, security or availability.

[12] Often, a distinction between functional and non-functional (quality) characteristics is made. Following the ISO/IEC 9126 standard, we subsume "functionality" under "quality".

Moment during the Life-Cycle (*Life-Cycle*)

This dimension classifies the techniques and methods according to the moment during the life-cycle at which they can be / should be employed. For the purpose of this chapter, we distinguish between two major phases in the life-cycle: *design* (before the application is being deployed) and *operation* (while and after the system has been deployed).

More detailed software life-cycle models for service-based applications are discussed in Section NNN.

Research Discipline (*Discipline*)

This dimensions states which research disciplines has proposed the discussed solution (this can typically be identified by the affiliation of the authors or the conference / journal in which the paper has been published).

Those disciplines include (but are not limited to) Business Process Management (*BPM*), Service-oriented Computing (*SOC*), Software Engineering (*SE*) and Grid Computing (*Grid*).

Relevant Layer of the Service-Based Application (*Layer*)

A check can involve artifacts on different layers of a service-based application. These layers are Business Process Management (*BPM*), Service Composition and Coordination (*SCC*) and Service Infrastructure (*SI*).

Artifact That Is Checked (*Artifact*)

This dimension classifies the technique or method according to the artifact that is checked (i.e., analyzed, tested or monitored).

The entity which is checked, can include – besides others – an atomic service (which does not invoke other services), a composed/aggregated service (or a service-based application), a service registry, or the context of a service or a service-based application. As an example, many monitoring approaches (see Section 7.6) observe changes in the context of the system in order to produce monitoring results and to enable the adaptation of the service-based application.

Artifact against Which the Entity Is Checked (*Reference*)

In order to check an entity, a reference artifact is needed against which the check is performed. As an example, when performing the test of an atomic service, the service implementation could be tested against the service interface.

Examples for such reference artifacts are service interfaces, service specifications, or Service-level Agreements (SLAs).

Level of Formalization (*Formality*)

The following levels of formalization of a quality assurance technique and method can be distinguished: *formal*, *semi-formal*, or *non-formal* techniques and methods.

The results of a formal techniques or methods are achieved by means of mathematical methods. This requires that the input artifacts to such formal techniques or methods have a formal representation. Examples for formal techniques include model checking, correctness proofs or symbolic execution.

A semi-formal technique or method rests on a language that contain formal as well as informal elements. Thus those artifacts are not completely amenable to mathematical methods.

Examples for non-formal techniques are reviews or inspections. Although the process for inspections is rigorously defined, the steps during inspection do not rely on mathematical precision.

Degree of Automation (*Automation*)

Quality assurance techniques and methods can consist of individual steps. An individual step can be amenable to automation (i.e., it can be executed by a software tool) or it cannot be automated, because it requires a human to perform a creative task.

The degree of automation of a technique or method can thus range from *fully automated*, over *partially automated* to *not automated*.

In a fully automated technique or method no manual steps have to be performed. During the application of a partially automated technique or method, not all the steps are performed by tools and thus some steps require "user" intervention.

Means of Validation (*Validation*)

This dimension aims at classifying how the proposed technique or method has been (empirically) validated.

The following "levels" of empirical evaluation, taken from [156], are used for that purpose:

- *Controlled Experiment:* All of the following exist: Random assignment of treatments to subjects. Large sample size. Hypotheses formulated. Independent variable selected. Random sampling.
- *Quasi Experiment:* One or more of points in Controlled Experiment are missing.
- *Case Study:* All of the following exist: Research question stated. Propositions stated. Unit(s) of analysis stated. Logic linking the data to propositions stated. Criteria for interpreting the findings provided. Performed in a 'real world' situation.

- *Exploratory Case Study:* One or more of points in Case Study are missing.
- *Experience Report:* All of the following exist: Retrospective. No propositions (generally). Does not necessarily answer how or why. Often includes lessons learned.
- *Meta-Analysis:* Study incorporates results from previous similar studies in the analysis.
- *Example Application:* Authors describing an application and provide an example to assist in the description, but the example is "used to validate" or "evaluate" as far as the authors suggest.
- *Survey:* Structured or unstructured questions given to participants.
- *Discussion:* Provided some qualitative, textual, opinion-oriented evaluation. E.g., compare and contrast, oral discussion of advantages and disadvantages.

**

The following sections 7.5 - 7.7 comprise the results of the literature survey for the three major classes of analytical quality assurance techniques and methods for service-based applications. At the end of the description of the survyed quality assurance techniques, the classsification results (see Section 7.4) are summarized and compared in two tables. The first table (Table-A) contains the *Class*, *Quality*, *Life-Cycle*, *Discipline* and *Layer* dimension of the classification framework. The second table (Table-B) shows the *Artifact*, *Reference*, *Formality*, *Automation* and *Validation* dimension.

7.5 Testing

7.5.1 Test Case Generation

An Approach for Specification-based Test Case Generation for Web Services [60]

Specification-based testing of Web services is important for consumers and brokers as Web services source code is usually unavailable to them and those stakeholders only have access to the descriptions or specifications of the Web services. Hanna and Munro in [60] thus describe a method for specification-based testing of Web services using WSDL and XML Schema data types.

Extending WSDL to Facilitate Web Services Testing [139]

WSDL files can be seen as a specification for testing Web services. A WSDL specification contains the number of inputs and outputs, the type and order of inputs and outputs and how Web services should be invoked. However, this information is not sufficient for testing. Thus, in [139] Tsai et al. propose four kinds of extensions for WSDL files: (1) input-output dependency, which can be generated by dependence analysis inside a Web service; (2) invocation sequence, which provides tracing information among Web services; (3) hierarchical functional description, which can improve functional and regression testing and enable automation; (4) concurrent sequence specifications, to capture Calling sequences, concurrent behaviors and timing aspects.

Coyote: An XML-Based Framework for Web Services Testing [143]

Web services are distributed and often only WSDL specifications are available. The authors of [143] thus propose an XML-based object-oriented testing framework called Coyote. The framework is composed of test master and test engine. The test master specifies test scenarios and test cases, performs the analysis and converts WSDL specifications into test scenarios. The test engine interacts with the Web service under test and provides tracing information.

WSDL-Based Automatic Test Case Generation for Web Services Testing [8]

Services are published, bound, invoked and integrated at runtime and only have a programmable interface. So automation of the testing process without user interaction is essential. In [8] Bai et al. thus propose a technique for generation test cases automatically based on WSDL specifications. They present their technique for individual and combination operations of atomic services. The technique consists of four steps: First the test data is generated from WSDL message definitions. Then test operations are generated based on the analysis of the parameters in the WSDL file. The third step is the generation of operation flows to test a sequence of operations by operation dependency analysis. At the end the test specification is build as test cases encoded in XML files.

Swiss Cheese Test Case Generation for Web Services Testing [136, 140]

Web services are based on UDDI which is not responsible for the quality of services. So the trustworthiness or vulnerability of Web services is a problem for the users of those services. Traditional dependability techniques must be redesigned to handle the dynamic features of Web services. To this end, in [136] the authors present the 'Swiss Cheese' test case generation approach. The OWL-S specification of a Web service is converted to scenarios. After that boolean expressions are extracted. With this boolean expressions a K-map is formed and Hamming distances and boundary counts are computed. Finally a Swiss Cheese map is created in which each cell at least belongs to a test case. Positive and negative test cases are generated. At the end the test cases can be ranked.

Ontology-Based Test Case Generation for Testing Web Services [152]

To test automatically constructed Web services, Wang et al. present a model-based approach with automatically generated test cases based on the OWL-S

Web services process specification. The OWL-S description is first transformed to a Petri-Net model to provide a formal representation of the structure and behavior of the service under test. Then, test cases are generated based on the Petri-Net model. A prototype implementation, called TCGen4WS, has been implemented.

Automated Functional Conformance Test Generation for Semantic Web Services [106]

In [106], the authors introduce an approach to generate functional conformance tests for Semantic Web services which are defined using the Inputs, Outputs, Preconditions, Effects (IOPEs) paradigm. For each Web service, the approach produces testing goals which are refinements of the Web service preconditions using a set of fault models. A planner component accepts these testing goals, along with an initial state of the world and the Web service definitions to generate a sequence of Web service invocations as a test case.

Generating Test Cases for Web Services Using Extended Finite State Machine [78]

As observed in [78], Web services are distributed applications with numerous aspects of runtime behavior that are different from typical applications. To this end, the authors introduce a new approach to testing Web services based on EFSM (extended finite state machine). A WSDL (Web services description language) file alone does not provide dynamic behavior information. Thus, the authors propose augmented it with a behavior specification, in the form of a formal EFSM model.

Contract-Based Testing for Web Services [33]

In [33] Dai et al. describe that the problems of testing services derive from the fact that services are invoked instead of integrated and thus providers can evolve a service without the knowledge of the users. This lack of notification on changes are a problem for users of a service because they want to be sure that the service functions as stipulated when first using the service. The authors conclude that the system "has to be tested dynamically and automatically at runtime without human interaction" and they present an approach which uses contracts as formal agreements between users and providers containing rights and obligations for both sides. The contracts are described using the OWL-S process model. For generating test cases two parts are suggested: the generation of valid test data and the generation of the test process. Both generation processes are based on contracts. In addition, contracts contain enough information on the expected output for using them as test oracles.

Towards Contract-Based Testing of Web Services [61]

In [61], the authors state that due to the loose coupling and the distribution of services, service requestors often bind to services at run-time. In the authors' point of view this prevents integration testing of a service-based application. Thus, the authors propose using 'Design by Contract' for Web services. For the interoperability of service providers and requesters the concept of Design by Contract, which comes from component-based systems, should be complemented by the use of required and provided contracts. A provided contract specifies pre- and post-conditions of the service. A required contract specifies the information the requester is willing to provide and the situation he wants to achieve at the end. For the representation of the contracts Heckel and Lohmann propose the use of graph transformation rules.

Testing BPEL-Based Web Service Composition Using High-Level Petri Nets [41]

As observed in [41], BPEL-based Web service composition essential has dynamic aspect such as recomposition, re-configuration, and dynamic binding during execution, which makes behavior analysis and testing of BPEL-based Web service composition software significantly complicated. To this end, the authors propose a technique for analysis and testing BPEL-based Web service composition using high-level Petri nets (HPN). To illustrate how these compositions are verified, the relationships between BPEL-based Web service composition and high-level Petri nets is constructed. By analyzing the structure of Web service composition based on BPEL, the corresponding HPN is constructed. After translation, the equivalent HPN of the Web service composition based on BPEL can be verified on existing mature tool, and the related researches on HPN, e.g., testing coverage and reduction techniques that have been studied deeply, can be employed in testing of Web service composition based on BPEL.

On Combining Multi-formalism Knowledge to Select Test Models for Model Transformation Testing [124]

In [124] the authors present a method to automatically select test models given any meta-model as input. The selected models are used to test model transformations. Model transformations are ubiquitous in Model-driven Engineering as they automate important software development steps. Testing and validating them is thus a crucial activity for successful MDE. These test models serve as input for black-box testing of a model transformation. The papers outlines a black-box testing tool Cartier that uses Alloy as the relational logic language to represent combined knowledge.

Probabilistic QoS and Soft Contracts for Transaction Based Web Services [111]

To test Web services, one needs test oracles that states whether a contract was satisfied by the input test case. In [111] the authors present probability distributions of QoS properties such as response time as soft contracts that can act as test oracles. In particular they present the idea of obtaining a QoS contract for a Web service orchestrations and choreographies leading to composed services. These soft contracts are better suited to incremental testing compared hard contracts (e.g., response time always less than 5 msec).

Model-Based Functional Conformance Testing of Web Services Operating on Persistent Data [127]

The WSDL standard does not allow behavioral specification (such as pre- and postconditions) of Web services in the presence of persistent data. Thus, the authors in [127] propose the use of existing test generation techniques based on Extended Finite State Machine (EFSM) specifications to address the generation of functional conformance tests for Web services which operate on persistent data. Key contribution is an algorithm to translate a WSDL-S behavioral specification of operations of a Web service into an equivalent EFSM representation which can be exploited to generate an effective set of test cases.

Generation of Conformance Test Suites for Compositions of Web Services Using Model Checking [54]

The complexity of testing compositions of Web services relies on their distributed nature and asynchronous behaviour. To consider this challenge, a new testing method for compositions of Web services is proposed in [54]. Specifically, a formal verification tool (SPIN model checker) is used to automatically generate test suites for compositions specified in BPEL.

7.5.2 Test Execution

Scenario-Based Web Service Testing with Distributed Agents [138]

Tsai et al. [138] motivate that testing Web services is difficult, because they are loosely coupled, have a dynamic behavior, are invoked by unknown parties, can have concurrent threads and object sharing and different parties (client, broker and provider) are involved in testing. To address those challenges, the authors propose a Web service testing framework (WSTF). The proposed WSTF includes the following features: (1) enhanced WSDL, which includes four kinds of extensions as described in [139]; (2) scenario-based testing as a specification-based testing technique which can be applied to WSDL files; (3) automated test script generation based on the scenario specification; (4) automated distributed test execution including test monitors and distributed agents.

Group Testing for Web Services [137, 145, 135, 144, 5, 142]

A large number of Web services is expected to become available on the Internet. In particular, this means that for the same specification different implementations (services) will possibly exist. To test all those services (which satisfy a specification), huge effort for testing needs to be invested. To this end, in [137, 145] suggest progressive group testing as a solution to be applied for unit and integration testing. Two phases are proposed: (1) prescreening, during which 'unlikely-to-win' Web services are eliminated as far as possible; (2) runtime group testing, during which the best candidates identified in the prescreening phase are integrated in the live system and tested. Given a set of functionally equivalent Web services, the input for one of these services is forwarded to all of them. The results from all services are voted (by a voting service). By comparing the output of one service with the weighted majority output (as oracle) faults can be detected.

In [135] the authors integrate their group testing technique into a framework called ASTRAR (Adaptive Service Testing and Ranking with Automated oracle generation and test case Ranking). In [5] group testing is further extended with an adaptive mechanism. This adaptive mechanism provides the possibility to adapt test cases to the continuously changing services (in case of updates or redeployment). Finally, in [142] the authors introduce a stochastic voting for group testing.

Perturbation-Based Testing for Web Services [102, 35, 154]

Web services can be located on different servers belonging to different companies. The Web services interact by passing messages and data (through XML, SOAP). In [102] the authors propose to exploit data perturbation as a means to test the interaction of Web services. Request messages are modified. With the resending of the modified request the messages are used as test cases. The analysis of the response messages reveals uncorrect behavior. In [154] further perturbation operators for XML schema are described. [35] builds upon the work of Offutt and Xu. They introduce new perturbation operators for the modification of SOAP messages.

A Test Bed for Web Services Protocols [110]

Transactions involving more than one Web service can become non-trivial and thus require the use of some pre-agreed or standard protocols. A proper specification and implementation of these transaction protocols this becomes critical for the correct execution and termination of transactions.. To address this problem, the authors of [110] propose a test bed based on conformance checking for automatically testing a given Web service protocol implementation.

Regression Testing Approach of Ruth and Tu [118, 117, 119]

Similar to the modification of traditional software systems, for each modification of a Web service two aspects have to be examined: (1) Do the modified parts function correctly? (2) Does the modification have effects on the unmodified functions? For the second aspect it is common practice to use regression testing and to run previously generated test cases again. In [118] the authors propose to apply safe regression test selection (RTS) to Web services. The approach is based on the safe RTS algorithm by Rothermel and Harrold for monolithic applications and builds on control-flow graphs.

In [117] that approach is extended so that the safe RTS technique can be automated. Another extension is the handling of multiple concurrent modifications which is described in [119].

Quality Analysis of Composed Services through Fault Injection [53]

The authors in [53] present a systematic testing method for service-based, cooperative information system (CIS). The method is based on fault injection during process execution. Two types of testing are supported: (1) black-box testing, when the service implementation code is not accessible; (2) white-box testing, when service code is accessible, and in particular when information on used data sources used by the service is also visible. Two types of faults are considered: data faults and time delays.

7.5.3 Testing Frameworks and Tools

BPEL Unit Testing [92, 84]

One problem with testing BPEL compositions, as observed by the authors, are the numerous external dependencies. These dependencies are based on the Web services that are accessed by the BPEL composition. To systematically consider those dependencies, the authors present an approach for unit testing BPEL processes. The framework supports the following steps: (1) test specification, where the WSDL specification is used to generate test cases; (2) test organization, where test cases are grouped into test suites with links to external artifacts; (3) test execution, where the BPEL process can be tested by simulation or by real-life deployment; (4) test results, where a report of successful test cases, failures and errors is generated.

Biased Covering Arrays for Progressive Ranking and Composition of Web Services [23]

When building a composed service, a choice between several services that provide the same functionality must be taken. The authors of [23] build upon the group testing techniques of Tsai et al. [137, 145, 135, 144, 5]. They apply Group Testing to narrow down the number of prospective candidates for each

Web service. As a subsequent step, they propose to use interaction testing techniques, specifically biased covering arrays, to generate economically sized test suites that further evaluate the candidate Web services.

Testing of Service-Oriented Architectures – A Practical Approach [43]

The authors identify two key issues for testing: (1) whenever a change in one of the source code parts arises, a new test has to be done; (2) very often there is no stable test environment, which leads to the problem of setting up the whole test environment for every single change. Thus, the investment for testing can explode in such a setting. To address these issues, the authors suggest to exploit the automation of the test process, which promises to lead to a decrease of costs and time for testing. In [43] an approach for automated testing of services is presented including a Meta language in XML for the definition of test cases. The authors focus on the presentation of a prototype implementation called SITT (Service Integration Test Tool).

An Abstract Workflow-Based Framework for Testing Composed Web Services [68]

As has been discussed above, testing composed Web services imposes many challenges to existing testing methods, techniques, and tools. To address some of those challenges, the authors in [68] introduce a formal model for an abstract-based workflow framework that can be used to capture a composed Web service under test. The structural-based workflow framework uses workflow graphs for simple composed and complex composed Web services. Additionally a set of applicable structural-based testing criteria to the framework is defined.

WSDLTest – A Tool for Testing Web Services [128]

The authors observe that in general, one cannot guarantee that Web services will work as one might expect, which could lead to serious interaction errors. As a solution, the authors propose to simulate the usage of the services, where both requests are automatically generated and responses are automatically validated. In [128], a tool that accomplishes this goal is introduced. The tool generates Web service requests from the WSDL schemas and adjusts them in accordance with the pre-condition assertions written by the tester. It dispatches the requests and captures the responses. After testing it then verifying the response contents against the post-condition assertions composed by the tester.

Automatic Conformance Testing of Web Services [62]

In [62] Heckel and Mariani present how service descriptions can be supplemented by adding a behavioral specification of the service consisting of graph transformation rules. Still, it remains open whether the actual service implementation is correct with respect to this model, and thus whether such a model is a valid representation of the service. To overcome this problem, the authors introduce high-quality service discovery which adds automatic testing to the approach of behavioural matching with graph transformation rules. In this setting, a service can be entered into the registry only, if it has succesfully passed testing.

WebSob: Automated Robustness Testing of Web Services [88, 89, 90]

In [88, 89] and [90] Martin, Basu and Xie look at Web service testing from the consumer point of view. Consumers mostly don't have the possibility to get implementation details of the used Web services. Thus consumers can only perform black-box testing. Moreover robustness of the Web service is a problem for the consumer. The Web service has to handle input parameters which contain consumer specific information. If the Web service is not robust enough to handle these inputs it is possible that unauthorized instances retrieve consumer specific information. To address those issues, the authors propose the framework "WebSob" for automated robustness testing of Web services.

Distributed Functional and Load Tests for Web services [122]

The authors introduce a flexible test framework that allows executing functional, service interaction and load tests. The framework is presented as being generic in terms of being largely independent of the system to be tested. The paper discusses the automation capabilites of the test framework that rely on the Testing and Test Control Notation (TTCN-3).

A Multi-agent Based Framework for Collaborative Testing on Web Services [7]

The auhtors start from the observation that testing is a challenge due the dynamic and collaborative nature of Web services. To address these issues they introduce a multi-agent based framework to coordinate distributed test agents to generate, plan, execute, monitor and communicate tests on Web services.

The Audition Framework for Testing Web services Interoperability [18]

To consider the SOA propoerties such as loose coupling, distribution and dynamism of 'components', the authors propose a framework that extends UDDI registry role from the current one of a 'passive' service directory, to become an accredited testing 'organism', which validates service behaviour before actually registering it. This specific form of testing, called audition, mainly focuses on interoperability issues, such as to facilitate the coordination among services registered at the same UDDI.

A Framework for Testing Web Services and Its Supporting Tool [95]

In their paper, the authors focus on facilitating the testing of Web services. In particular, they propose a framework for testing Web services, which can help a tester of Web services in two ways: (1) it can help the tester to acquire effective test data; (2) it can help the tester to execute the test cases (that include that data).

Testing Web services [126]

Im their paper, the authors introduce a technique for testing Web services using mutation analysis. The technique is based on applying mutation operators to the WSDL document in order to generate mutated Web service interfaces that are used to test the Web service. For this purpose, the authors define mutant operators that are specific to WSDL descriptions.

A Model-Driven Approach to Discovery, Testing and Monitoring of Web Services [82]

The authors observe that established technology for providing, querying and binding services is largely based on syntactic information. As a consequence of the lack of semantic information in service descriptions, the reliable, automatic integration of services is hindered. To address this problem, the authors introduce a framework which enforces that only tested Web services are participating in (as a consquence) high-quality service-based applications. To achieve this, only successfully tested services are stored in registries (cf. [62]).

A Simple Approach for Testing Web Service Based Applications [132]

In their paper, the authors introduce a technique for supporting the construction of reliable Web applications composed of Web services. All relevant Web services are linked to the component under test at the testing time; thus, the availability of suitable Web services is guaranteed at invocation time. The Web application and its composed components are specified by a two-level abstract model: (1) the Web application itself is represented as task precedence graph (TPG); (2) the behavior of the individual components is represented as a timed labeled transition system (TLTS). Three sets of test sequences are generated and executed automatically using a test execution algorithm and a test framework.

Search-Based Testing of Service Level Agreements [40]

In [40], the authors deal with the problem on how to detect the violation of a Service Level Agreement (SLA), negotiated between service provider and service consumer. As the violation of an SLA is undesirable for both the provider and the consumer, the authors suggest that the SLA should be tested before the service is offered, as this can reduce the probability that the SLA will be violated during the service usage. The presented solution builds on Genetic Algorithms (GAs) for the generation of test data.

Using Test Cases as Contract to Ensure Service Compliance across Releases [22, 107]

In [22], the authors address the issue that services are used but not owned. This means that services are out of the users' control, leading to the problems that the users thus cannot decide on migrating to a new version of the service, or that the users are not always aware of changes the service provider makes to the service implementation. To address those problems, the authors present an approach appropriate for regression testing by users. The approach has its roots in component-based software testing and uses test cases as a contract between the user and the provider of the service. The basic idea of the approach is to bind test cases and a set of QoS assertions to a service and to test after a certain time if the test cases and assertions still hold.

7.5.4 Online Testing

A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications [28]

As the authors observe, a service-based application may bind dynamically to its constituent services. This means that for the same service interface, the actual services that are invoked may behave differently. To address this issue, the authors propose a metamorphic approach for online services testing. During off-line testing first a set of successful test cases is constructed and corresponding follow-up test cases for online testing are determined.

Dynamic Reconfigurable Testing of Service-Oriented Architecture [6]

The authors point out that one problem imposed by the dynamic reconfiguration of service-based applications is that testing needs to adapt to the changes of the service-based applications at runtime. As a solution, the paper presents a testing approach enabling the run-time change of test organization, test scheduling, test deployment, test case binding, and service binding. The approach is based on previous research on the MAST (Multi-Agents-based Service Testing) framework [7]. It extends MAST with a new test broker architecture, configuration management and event-based subscription/notification mechanism.

7.5.5 Classification of Testing Techniques

The various approaches that have been discussed above are classified and summarized (according to the classification scheme introduced in Section 7.4) in tables 7.1 and 7.2.

Table 7.1. Table-A for Testing

Contribution	*Class*	*Quality*	*Life-Cycle*	*Discipline*	*Layer*
[60]	testing	correctness	design	SE, SOC	SCC
[139]	testing	functionality	design	SOC	SI
[143]	testing	functionality	design	SOC	SCC
[8]	testing	functionality	design	SOC	SCC
[136, 140]	testing	trustworthiness, robustness	design	SOC, SE	SCC
[152]	testing	functionality	design	SOC, SE	SCC
[106]	testing	functionality, conformance	design	SE, SOC	SCC
[78]	testing	functionality	design	SE, SOC	SCC
[33]	testing, monitoring	functionality	operation	SOC	SI
[61]	testing	functionality, interoperability	design	SE	SI
[41]	testing	functionality	design	SOC	SCC, BPM
[124]	testing	functionality	operation	SE	SCC
[111]	testing	several quality characteristics	operation	SE	SCC
[127]	testing	functionality, conformance	design	SE, SOC	SCC
[138]	testing, monitoring	functionality	whole life-cycle	SOC, SE	SCC, SI
[137, 145, 135, 144, 5, 142]	testing, monitoring	functionality	design and operation	SOC	SCC
[102, 35]	testing	functionality	design	SE	SI
[92, 84]	testing	functionality	design	SOC	SCC, BPM
[23]	testing	functionality	design	SE	SCC
[43]	testing, monitoring	functionality	design	SOC	SCC
[68]	testing	functionality	design	SOC	SCC
[110]	testing	conformance	design	SOC	SCC
[128]	testing	functionality	design	SE	SCC
[62]	testing	conformance	registration	SE	SI
[88, 89, 90]	testing	robustness	design	SOC, SE	SCC
[118, 117, 119]	testing	functionality	execution	SOC, SE	SCC
[122]	testing	functionality and performance load	design	SE	SCC
[7]	testing, monitoring	functionality	design	SOC, SE	SCC
[18]	testing	interoperability	registration	SE	SCC
[95]	testing	functionality	design	SE	SCC
[126]	testing	functionality	design	SE	SCC
[82]	testing, monitoring	reliability, conformance	whole life-cycle	SE	SCC
[54]	testing	functionality	design	SE, SOC	SCC
[132]	testing	correctness, reliability, availability	design	SOC	SCC
[40]	testing	all QoS characteristics	operation	SE, SOC	SCC
[22, 107]	testing, monitoring	arbitrary QoS characteristics	operation	SOC, SE	SCC
[28]	testing	functionality	operation	SOC	SCC
[6]	testing	functionality	operation	SOC, SE	SCC
[53]	testing	functionality	design	SOC, IS	BPM

Table 7.2. Table-B for Testing

Contribution	*Artifact*	*Reference*	*Formality*	*Automation*	*Evaluation*
[60]	service, service composition	WSDL description	semi-formal	automated	discussion
[139]	service	WSDL spec.	-	-	-
[143]	service, service composition	WSDL spec.	semi-formal	automated	-
[8]	(atomic) service	WSDL spec.	formal	automated	experience report
[136, 140]	web service	OWL-S spec.	formal	non-automated	example appl.
[152]	web service	OWL-S spec.	formal	automated	example appl.
[106]	semantic web services	inputs, outputs, preconditions, effects (IOPEs)	semi-formal	automated	experience report
[78]	web service	EFSM based on WSDL spec.	formal	non-automated	example appl.
[33]	services, service compositions	OWL-S process model	semi-formal	automated	discussion
[61]	(atomic) service	provided and required contracts	formal	automated	discussion
[41]	web service composition	BPEL spec.	semi-formal	automated	discussion
[124]	service, service composition	provided and required contracts	formal	automated	discussion
[111]	service composition	probability distribution functions	formal	automated	discussion
[127]	web services which operate in the presence of persistent data	EFSM based on WSDL-S specification	formal	automated	discussion
[138]	service composition	WSDL description	semi-formal	automated	example appl.
[137, 145, 135, 144, 5, 142]	group of services	majority output	semi-formal	automated	example
[102, 35]	test of interaction between two web services	XML and SOAP messages	semi-formal	automated	example appl.
[92, 84]	BPEL process	WSDL spec.	semi-formal	automated	example appl.
[23]	composed web service	other web services	formal	non-automated	example appl.
[43]	service and service workflow	XML test description	semi-formal	automated	example appl.
[68]	composed service and workflow	-	formal	non-automated	example
[110]	web service protocol	formal model of the web service protocol	formal	automated	example appl.

Table 7.2. (continued)

Contribution	*Artifact*	*Reference*	*Formality*	*Automation*	*Evaluation*
[128]	web service	WSDL spec.	non-formal	automated	experience report
[62]	(atomic) service	specification (as graph-transformation rules)	formal	automated	experience report
[88, 89, 90]	service, service composition	WSDL spec.	semi-formal	automated	explor. case study
[118, 117, 119]	service, service composition	control-flow graph	formal	automated	example appl.
[122]	service-based application	XML description	non-formal	automated	discussion
[7]	atomic and composed services	service spec.	semi-formal	automated	discussion
[18]	web service	protocl state machine spec.	semi-formal	automated	discussion
[95]	composed web service	WSDL spec.	semi-formal	automated	example appl.
[126]	web service	WSDL spec.	semi-formal	automated	example
[82]	service-based application	UML models and graph transformation rules	formal	automated	discussion
[54]	compositions of web services	BPEL spec.	formal	automated	discussion
[132]	web service based application	WSDL spec., TLTS, TPG	formal	automated	discussion
[40]	service composition	SLA	semi-formal	automated	case study
[22, 107]	services, service compositions	WSDL description	semi-formal	automated	example appl.
[28]	service-based application	oracle from offline test	formal	automated	experience report
[6]	service-oriented application	-	semi-formal	automated	example
[53]	service, service composition	WSDL, BPEL	semi-formal	automated	discussion

7.6 Monitoring

7.6.1 Web Service Monitoring

Assumption-Based Monitoring of Service Compositions [108, 10]

The papers [108, 10] focus on run-time checking of the behavioral requirements (expressed in a special temporal logic-based notation) on the services participating to the composition, and collecting certain statistical and non-functional information over their execution.

Monitoring Conversational Web Services [20]

The work presents an approach for checking at run-time that the actual behavior of the conversational service complies with the expected behavior represented in a special algebraic notation that expresses functional constraints on the service evolution.

Smart Monitors for Composed Services [13]

A technique for monitoring functional and non-functional assertions over BPEL process activities is presented. Assertions may be specified either in programming language directly, or using special assertion language. The monitored process is modified in a way to interact with a dedicated monitoring service to provide the relevant data.

Dynamo [14, 15, 11]

Dynamo framework proposes an expressive monitoring language WSCoL for expressing functional and non-functional requirements (assertions and invariants) on the BPEL processes. The framework allows for collecting information from external sources (services). In [11] the language is extended with the possibilities to express more complex (temporal) properties over the executions of underlying processes.

Monitoring for Diagnosis [4]

Monitoring approach in this work has a goal of collecting and reporting a fault diagnosis information, rather than simply detecting property violation. The approach is based on a distributed framework that extends the functionality of the services with the diagnosis evaluation and management facilities.

Monitoring Privacy-Agreement Compliance [17]

The work addresses run-time monitoring of compliance of the privacy agreement defining the user's privacy rights and their possible handling by the service provider. The privacy properties (obligations and data rights) are formally in temporal logic specifications and the corresponding state machines, and then monitored at run-time using the events reported in the system log.

Requirements Monitoring Based on Event Calculus [130, 86, 87]

The authors approach the problem of monitoring service-based applications for conformance to a set behavioral requirements expressed in a rich event calculus-based specification language, and deals with service events, quality metrics, temporal constraints, etc. A run-time logic inference engine is used to detect violations of the properties.

Monitoring Security Patterns [129, 79]

The problem of monitoring security properties is addressed. The security properties are specified as patterns and formally represented in event calculus-based notation. The run-time checking of the properties relies on the approach defined above.

Performance Monitoring for Utility Computing [46]

The work deals with monitoring of SLA. The contract patterns are formally represented in event calculus and monitored using a specific framework.

Planning and Monitoring Execution with Business Assertions [81]

In the work monitoring is used to detect failures in the execution of customized business process in order to dynamically re-plan and adapt the process execution to the user requirements.

Automated SLA Monitoring [120]

The work proposes an approach for monitoring compliance of the service execution to the predefined contract information (SLA). The formalized contract statements are monitored by intercepting service interactions and process logs. The management of the monitored information and the analysis of compliance is performed by a dedicated platform.

WSLA [77]

An industrial approach proposed by IBM for monitoring SLA information is presented. The framework relies on a comprehensive model of contracts, QoS metrics and their aggregation, and on a sophisticated, multi-functional run-time environment.

Cremona [83]

Cremona is an industrial run-time platform for negotiating, managing, and monitoring SLA agreements. The agreements are defined using WS-Agreement notation. Multi-layered and extensible execution platform and API support the monitoring and management actions.

Colombo [31]

Colombo is an industrial platform for the development and enactment of services and service-based applications. Apart from many other facilities, it provides a way to check and enforce service policies expressed in WS-Policy notations. The policies may be attached to services, service operations, and even exchanged message types.

7.6.2 Process Monitoring and Mining

Query-Based Business Process Monitoring [16]

The approach targets the problem of monitoring BPEL processes. The monitoring queries are defined using visual pattern-based notations compliant with BPEL to define when the report should be provided, and using report specifications defining the information to be reported. For monitoring the queries are transformed into BPEL processes that collect information on the monitored processes from a low-level observer.

Model-Driven Development of Monitored Process [96]

The work proposes a way to develop SOA-based business processes with integrated monitoring information utilizing a model-driven approach. The authors have created a metamodel for modeling of process performance metrics (PPIs) based on BPMN process elements. The process augmented with monitoring primitives is automatically generated.

Model-Driven Development for BPM [29]

The work focuses on efficient development and deployment of monitoring information in BPM. The proposal relies on a meta-model extended with the concepts of metrics, business actions and events. The model is transformed into observer specification (for monitoring and evaluating the metrics) and a and data warehouse (to query and visualize information).

Probing and Monitoring WS-BPEL Processes [112]

The work deals with the problem on how to extract events from a BPEL process in order to enable auditing in an interoperable way. The authors propose a way to augment the BPEL processes with auditing activities, and present a set of strategies and mechanisms for collecting the relevant probes at different functional layers.

iBOM [27]

in this work the authors deal with the problem on how to create a monitoring solution, which not only enables to measure KPIs, but also to understand the causes of undesired KPI values, and prediction of future values. The approach is based on the combination of business activity monitoring with data mining to explain the monitored results.

An Agent-Based Architecture for BAM [66]

The work present a multi-layered agent-based architecture for providing continuous, real-time analytics for business processes.

Fuzzy Mining [59]

The work proposes an approach for automated extraction of the behavioral specification of the business process from the actual execution logs of the system. The approach relies data mining techniques for the log analysis.

Conformance Checking with ProM [115, 149]

The presented approach allows for checking (at run-time or posteriori) the conformance of the observed business process execution with respect to the actual process specification and characteristics. The monitoring is based on mining and analyzing information from process logs.

Process Mining for Security [147]

The approach allows for auditing security-critical properties of the process executions. First, the correct model of process is extracted using logs without security violations. Second, the actual executions are verified for conformance to that model using process mining techniques.

Deriving Protocol Models from Logs [101]

The approach to extraction of service interaction protocol models based on monitoring interaction logs is proposed. The approach is semi-automated and provides a way to interact with the designer in order to refine and correct the extracted model.

Timed Transition Discovery from Web Service Conversations [39]

The approach targets extraction of behavior model of the service in terms of temporal constraints (timed transitions) between relevant service interactions and events.

7.6.3 Grid Monitoring

Grid Monitoring Architecture (GMA) [133]

GMA is an abstraction of the essential characteristics needed for scalable high performance monitoring on a large distributed computational Grid. It provides the standard specification of the grid monitoring architecture, the components and their roles, the communication models, without, however, defining the implementation.

SCALEA-G [134]

A unified monitoring and performance analysis tool for the grid is presented. The tool provides flexible facilities for the definition and instrumentation of sensors, access to previously monitored information. The information may be collected both at the system (middleware) level and at the application level.

Globus MDS-2, MDS-4 [32, 51]

In Metacomputing Directory Service (MDS) architecture the monitoring information collected by distributed information providers is collected and stored in a dedicated aggregate directory services. The proposed architecture allows for distributed, standardized and easily extendable implementation of grid monitoring, while suffering from serious performance problems. These problems were taken into account and partially resolved in version MDS-4, where the corresponding directory services are re-implemented using Web service standards and solutions.

R-GMA [48]

R-GMA is a relational database implementation of GMA that can be used not only as a monitoring solution, but as a generic information source. It allows for managing monitoring information providers defined as database providers, stream data providers, or providers of historical data. The information queries are expressed in SQL-like notation and collect the historical data, ongoing events, or latest events of certain type.

MonALISA [100]

MonALISA is a monitoring solution based on peer-to-peer Jini platform. The platform is used for dynamic discovery, loading and replication of relevant common information. The data collection is performed by a special engine, which dynamically loads and controls the monitoring modules. The engine allows also for aggregating the previous information and is equipped with a powerful management user interface.

GridICE [1]

GridICE is a multi-layer centralized grid monitoring platform that is capable of observing simple and composite resource metrics. The collected data is made available for consumers by the publisher service, while the notifications, statistics and periodic reports are provided by the notification service.

7.6.4 Classification of Monitoring Techniques

The various approaches that have been discussed above are classified and summarized (according to the classification scheme introduced in Section 7.4) in tables 7.3 and 7.4.

Table 7.3. Table-A for Monitoring

Contribution	*Class*	*Quality*	*Life-Cycle*	*Discipline*	*Layer*
[108, 10]	monitoring	behavior	run-time (operation)	SOC	SCC
[20]	monitoring	behavioral conformance	run-time (operation)	SOC	SCC
[13]	monitoring	functional correctness	run-time (operation)	SOC	SCC
[14, 15, 11]	monitoring	behavioral correctness	run-time (operation)	SOC	SCC
[4]	monitoring	behavioral correctness	run-time (operation)	SOC	SCC
[17]	monitoring	privacy agreements	run-time (operation)	SOC	SI
[130, 86, 87]	monitoring	behavioral correctness, QoS (e.g., perfomance)	run-time (operation)	SOC	SCC
[129, 79]	monitoring	compliance to security requirements	run-time (operation)	SOC	SI, SCC
[46]	monitoring	correctness with respect to SLA properties	run-time (operation)	SOC, Grid	SI
[81]	monitoring	behavioral correctness	run-time (operation)	SOC, BPM	SCC, BPM
[120]	monitoring	SLA, QoS (security, performance, reliability, cost)	run-time (operation)	SOC	SI
[77]	monitoring	SLA, QoS (performance, reliabiltiy)	run-time (operation)	SOC	SI
[83]	monitoring	SLA, various observable QoS	run-time (operation)	SOC	SI
[31]	monitoring	functional correctness	run-time (operation)	SOC	SI
[16]	monitoring	relatively any characteristic of a process	run-time (operation)	BPM	BPM
[96]	monitoring	performance, KPI	run-rime (operation)	BPM	SCC
[29]	monitoring	business metrics, KPIs	run-time (operation), post-mortem	BPM	BPM
[112]	monitoring	relatively any characteristic of a process	run-time (operation)	BPM	SCC
[27]	monitoring	business parameters	run-time (operation)	BPM	BPM
[66]	monitoring	business properties and metrics	run-time (operation)	BPM	BPM
[59]	monitoring	behavior	post-mortem	BPM	BPM

Table 7.3. (continued)

Contribution	*Class*	*Quality*	*Life-Cycle*	*Discipline*	*Layer*
[115, 149]	monitoring	behavioral correctness	run-time (operation), post-mortem	BPM	BPM
[147]	monitoring	security (security-critical behavior)	run-time (operation), post-mortem	BPM	BPM
[101]	monitoring	behavior	post-mortem	SOC	SCC
[39]	monitoring	behavioral properties	post-mortem	SOC	SCC
[133]	monitoring	Performance	run-time	Grid	SI
[134]	monitoring	Performance	run-time	Grid	SI
[32, 51]	monitoring	Various domain-specific information (QoS, resource properties, available services)	run-time, post-mortem	Grid	SI
[48]	monitoring	Domain-specific information	run-time, post-mortem	Grid	SI
[100]	monitoring	Domain-specific information, performance	run-time, post-mortem	Grid	SI
[1]	monitoring	Domain-specific information, performance	run-time, post-mortem	Grid	SI

7.7 Analysis

7.7.1 Modelling and Simulation

QoS Analysis with PEPA Models [57, 58]

Service-based applications in general must be developed by taking into account both the problem of scalability and security. The solution proposed by the authors uses UML diagrams, such as state diagrams and sequence diagrams, which are suitable for model-driven development. Those diagrams are colored with performance-related characteristics of modeled systems, and they are automatically translated to PEPA, a common used stochastic process algebra. In [58], authors propose the analysis of retrieved PEPA models with the multi-terminal binary decision diagram (MTBDD)-based PRISM stochastic model checker.

Complexity analysis of BPEL Web Processes [26]

The complexity of BPEL descriptions, when used for describing Web service processes, can interfere with maintenance, understandability and effectiveness

Table 7.4. Table-B for Monitoring

Contribution	*Artifact*	*Reference*	*Formality*	*Automation*	*Evaluation*
[108, 10]	composition of BPEL processes	behavioral composition requirements	formal	automated	explor. case study
[20]	composition of BPEL processes	conversation constraints	formal	automated	explor. case study
[13]	BPEL process	func. assertions on BPEL activities	formal	automated	explor. case study
[14, 15, 11]	BPEL process	func. and non-func. assertions, temporal requirements	formal	automated	explor. case studies
[4]	global behavior of service composition	compliance with local service execution models	formal	automated	explor. case study
[17]	use of privacy information by service provider	privacy agreement properties (rights and obligations)	formal	automated	discussion

of both the WS-based application and development. Thus, it can be useful to measure the complexity of such descriptions in term of metrics derived by static analysis of the BPEL descriptions. The proposed approach uses three different static-analysis derived metrics to predict and measure the complexity of a BPEL description.

Performance Modeling of WS-BPEL-Based Web Service Compositions [116]

Classic QoS analysis of service-based applications is applied in the context of this chapter in order to select an optimal set of services to orchestrate a Web service composition by filling a WS-BPEL description of the process, which is the aim of the service integrator. The non-functional contract between the integrator and the third party service providers is composed of a set of Service Level Agreements (SLAs). The overall problem is the development of a performance analysis and a model for the evaluation of the quality of processes created by using WS-BPEL. To address this problem, the approach by the authors introduces a mathematical model, based on operational research, which describes the performance of composed web service processes written in the WS-BPEL language. The authors also introduce a distributed infrastructure which is able to detect if the introduction of a new instance affects (and how) the performance of the web service provider nodes, and detect if SLAs are violated or fulfilled.

Table 7.4. (continued)

Contribution	*Artifact*	*Reference*	*Formality*	*Automation*	*Evaluation*
[130, 86, 87]	BPEL process, service composition execution	func. and non-func. assertions, temporal requirements	formal	automated	explor. case study, experiments
[129, 79]	service composition execution	security properties (patterns)	formal	automated	explor. case study
[46]	BPEL process	func. and non-func. assertions, temporal requirements	formal	automated	discussion
[81]	business process	func. requirements and assertions	formal	automated	explor. case study
[120]	service execution	proprietary SLA properties and assertions	-	automated	discussion
[77]	service execution	proprietary SLA properties and assertions	-	automated	discussion
[83]	service execution	WS-Agreement properties and assertions	-	automated	discussion
[31]	services and message exchanges	WS-Policy spec.	-	automated	discussion
[16]	BPEL process	visual behavioral queries and report spec.	formal	automated	explor. case study
[96]	BPEL process	process performance indicators	-	automated	explor. case study
[29]	business process	business metrics	-	automated	discussion
[112]	BPEL process	process audit spec.	-	automated	discussion
[27]	business process	metrics, KPIs	-	automated	discussion
[66]	business process	metrics, KPIs	-	automated	discussion
[59]	process execution logs	-	formal	automated	explor. case study
[115, 149]	business process execution logs	process spec.	formal	automated	explor. case study, experiments
[147]	business process execution logs	security-correct process spec.	formal	automated	explor. case study
[101]	service interactions	-	formal	semi-automated	experiments
[39]	service interactions	-	formal	automated	discussion, experiments

Table 7.4. (continued)

Contribution	*Artifact*	*Reference*	*Formality*	*Automation*	*Evaluation*
[133]	service messages	Domain-specific metrics	-	automated	discussion
[134]	TCP streams	XPath queries over monitoring data	-	automated	TBD
[32, 51]	specific data providers	specific information queries	-	automated	discussion, case studies
[48]	data streams, specific providers	SQL-like queries	-	automated	discussion
[100]	specific data providers	regular expression predicates	-	automated	discussion
[1]	specific data providers	domain-specific metrics	-	automated	TBD

Performance Prediction of Web Service Workflows [91]

Web service based application play an important role in service-based architectures as they can be selected and composed in order to create highly complex applications. In this case, the Business Process Execution language (BPEL) can be used to express such compositions and interactions. Although, a very important factor to decide how composition can be instantiated, that is, which actual services must be selected, is the whole performance of the BPEL description. The prediction of the BPEL workflow performance can be also useful to detect if a given composition is able to provide the requirements about the quality of service. To this end, the authors propose an integrated framework to resolve the issue of performance prediction and assessment of workflows expressed in the BPEL language. The starting point for the prediction is composed of annotated BPEL and WSDL specifications, from which authors derive performance bounds on response time and throughput. Thus, users are able to assess the efficiency of the BPEL workflow, and service providers can, for example, adapt their compositions by estimating performance gains of different upgrades to existing systems.

A Logic-Based Verification of Contracts in Semantic Web Services[34]

Service contracts describe mutual expectations and commitments of the interacting participants. The dynamic aspect of contracts amounts to the workflow models of the interaction protocols seen from the point of view of each participant. The problem of automated contracting deals with the analysis of compliance between the expectations and commitments between the partners and therefore the verification of the compatibility of the workflow models. In order to capture the dynamic aspects of contracting, the authors in [34] propose a logic called CTR-S. The logic permits representation of the composition participants in terms of workflow, while the desired properties (expectations) are represented as constraints. The automated contracting is therefore a problem of verifying that the expectation is enforceable by the workflow model. A corresponding model and proof theories for capturing the workflow constructs (sequence, concurrent execution, external/internal choices, etc.) and execution constraints (single or serial occurrence of events, their arbitrary nested logical combinations) are developed and represented. Finally, the authors present the reasoning algorithms for the automated verification of the workflows against expectation constraints.

Web Service Interfaces [19]

A Web service often depends on other Web services, which have been implemented by different vendors, and their correct usage is governed by rules. Such rules may constrain data types and service signatures, but they may also express temporal constraints on the service invocations. In order to verify/enforce these rules, specific forms of analysis are necessary. To this end, the authors propose an approach to verify that within a composition one service can correctly collaborate with another or may be substituted by another service according to the predefined set of rules. In order to specify these rules, a special formalism, Web service interface language, is presented. The interface defines three kinds of rules on the Web service use: (*i*) it specifies the service signature rules (methods and their types of input / output parameters); (*ii*) consistency rules, i.e., propositional constraints on method calls and output values that may occur in a Web service conversation; and (*iii*) protocol rules, i.e., temporal constraints on the ordering of method calls. For each kind of rules a specific logic-based notation is presented and formalized.

Transforming BPEL into Annotated Deterministic Finite State Automata for Service Discovery [153]

Web services advocate loosely coupled systems, although current loosely coupled applications are limited to stateless services. The reason for this limitation is the lack of a method supporting matchmaking of state dependent services exemplarily specified in BPEL. In particular, the sender's requirement that the receiver must support all possible messages sent at a certain state are not captured by models currently used for service discovery. To this end, the authors in [153] present the concept of matching business processes in loosely coupled architectures. It proposes a transformation from BPEL to annotated Deterministic Finite State Automata aDFA. The transformation represents messages that might be sent by a party at a particular state as messages that must be supported by the corresponding receiving party. This explicit modeling of mandatory transitions of a sender and optional transitions of a receiver is the main contribution of this approach.

Specification and Validation of the Business Process Execution Language for Web Services [45]

The authors approach the problem of enriching business process models specified with BPEL4WS specification with operational semantics. Specifically, an abstract operational semantics for BPEL4WS in terms of a real-time distributed abstract state machine DASM is proposed in [45]. The BPEL abstract machine is organized into three basic layers reflecting different levels of abstraction. The top layer, called abstract model, provides an overview and defines the modeling framework comprehensively. The second layer, called intermediate model, specifies the relevant technical details and provides the full DASM model of the core constructs of the language. Finally, the third layer, called execution model, provides an abstract executable semantics of BPEL.

7.7.2 Verification of Service Compositions

Adaptive Service Composition in Flexible Processes [3]

The problem of service selection arises whenever complex applications, described as processes invoking services, need to select their composing elements from a set of functionally equivalent services which differ for nonfunctional characteristics (QoS parameters). The problem can be defined as the selection of the best set of available services at runtime. Constraints are both process-related ones, and end-user preferences. To this end, the authors introduce a modeling and analysis technique for the WS selection problem at runtime based on integer linear programming. (optimization problem). The new modeling approach to the service selection problem is based on diverse contributions:

Analysis of Interacting BPEL Web Services [52]

The analysis illustrated in the paper considers interactions of composite web services as conversations, that is, the sequence of messages which have been exchanged by the services. Compositions are described, as usual, with the BPEL language, against which some behavioral properties must be model-checked. The proposed solution translates BPEL specifications of composite web services to an intermediate representation, and then to the target verification language Promela, which can be used, together with a LTL property, as input to the SPIN model checker.

A Model Checking Approach to Verify BPEL4WS Workflows [21]

As in many fields of Software Engineering, the problem of practical formal verification by using model checking can be also used to verify functional properties in service-based applications. As one possible solution, the authors propose to translate Web Service compositions described in BPEL4WS to BIR, the source language of Bogor, a state-of-the-art extensible model checker.

First, the methodology can be used to verify deadlock freedom from Web Service compositions. Moreover, additional properties to be verified can be specified by using WS-CoL and LTL. In the first case, WS-CoL allows the predication on variables containing data both inside and outside the process; they can be verified by using assert statements in the BIR language. LTL properties can be verified by using two ad-hoc Bogor extensions.

Modeling Web Service Orchestration [94]

Current network technologies allow the development of new interaction business paradigms, such as virtual enterprises: different companies pool together their services to offer more complex, added-value products and services. Systems supporting such models are commonly referred to as Cooperative Information Systems (CIS). By using a service-based approach, the cooperative system consists of different distributed applications which integrate the E-services offered by different organizations. Such integration raises issues regarding service composability, correctness, synchronization and coordination. To address those issues, the authors in [94] propose the PARIDE framework (Process-based framework for oRchestratIon of Dynamic E-services) to define a common conceptual component model (and the related description language) for E-services, and the notions of compatibility and dynamic substitution of E-services based on the concept of cooperative process. PARIDE adopts a Petri Net-based model to ensure the description of the orchestration of E-services, and the related design of distributed orchestration engines. Besides, it provides analysis techniques based on the Petri Nets in order to address specific issues such as deadlocks, possible timeouts, configuration reachability, etc.

Workflow Verification [146, 148, 114]

In their work Aalst et al. address the problem of the verification and the analysis of service-based workflows. Based on a Petri-net-based representation of workflows, Aalst et al. [146] provide techniques to verify soundness property, e.g., a workflow is sound if and only if, for any case, the process terminates properly, i.e., termination is guaranteed, there are no dangling references, and deadlock and live-lock are absent. The correctness of a process can be decided by partitioning the workflow into sound subprocesses. A Petri-net-based workflow analyzer called Woflan is proposed to support the application of the approach. In [114], Aalst et al. are interested in providing answers to conformance problem due to the coexistence of event logs and process models of business workflows. They use Petri nets to model processes; this is argued by the fact that Petri nets are formal and have associated analysis techniques to easily parse any event log. [114] shows that conformance has two dimensions: fitness (the event log may be the result of the process modeled) and appropriateness (the model is a candidate from a structural and behavioral point of view). Metrics measuring fitness and appropriateness are supported by the Conformance Checker, a tool which has been implemented by the ProM Framework.

Modeling and Model Checking Web Services [123]

Schlingloff et al. [123] address the problem of checking correctness of composite web service processes. The original goal of modeling BPEL processes with Petri nets is to give the language BPEL4WS a formal semantic, and to compare the applicability of several formalisms for this task (e.g.,Abstract State Machines). The work described in [123] shows how to build Petri net models of web services formulated in the BPEL4WS specification language. The main goal is to define an abstract correctness criterion, called usability and to study the automated verification according to this criterion. This work relates correctness of web service models to the model checking problem for alternating temporal logics.

Model-Checking Verification for Reliable Web Service [98]

Model checking is a technique for the verification of software systems. In this paper, the author attempts to assess model checking techniques in the case of distributed service-based applications. The SPIN model-checker is used in [98] to verify a set of properties related to business flows, described by the WSFL workflow. SPIN provides a specification language Promela that describes the target system to be a collection of Promela processes (automata) with channel communications. The flow description written in WSFL is translated into Promela, the input specification language of SPIN. The properties to be checked are reachability, deadlock-freedom, or application specific progress properties. The application specific properties are expressed as formulas of LTL (Linear Temporal Logic), which are also fed into SPIN.

Modeling and Verifying Web Service Orchestration by Means of the Concurrency Workbench [80]

In their work, the authors address the problem of how to exploit verification techniques like model checking, preorder checking and equivalence checking to model and verify web service orchestrations. They propose the Concurrency Workbench (CWB) as a generic and customizable verification tool. The CWB supports model checking, preorder checking and equivalence checking. In the work presented in [80], authors show how the CWB and the Process Algebra Compiler (PAC) can be exploited to model and verify web service orchestration. To this end, a new calculus for formalizing web service orchestration is introduced. The operational semantics of BPE-calculus is used as input of the PAC to produce modules for the CWB.

Model Checking with Abstraction for Web Services [125]

In their paper, the authors address the problem of verifying the applications that implement the web services. Particularly, the authors address the state explosion problem of model checking techniques. They propose to use abstraction data techniques to face this problem. They apply this solution in the case of distributed service-based applications. They introduce the SatAbs tool [125] that allows for the analysis of service-based applications. It relies on model checking techniques to identify eventual flaws in such concurrent systems. [125] formalizes the semantics of a PHP-like language and enables modeling of both synchronous and asynchronous communication between services. The resulting models are amenable to verification using model checking.

Compatibility Verification for Web Service Choreography [49]

In [49], the authors address the problem of verifying process interactions for coordinated web services composition. Web Service workflow languages aim to fulfil the requirement of a coordinated and collaborative service invocation specification to support long running and multi-service transactions. Amongst the key issues in the design and implementation of components in this architecture style for critical business applications, is the formation of compositions as a series of interacting workflows and how transactions of activities interact to support the underlying business requirements. The authors proposes to use finite state machine representations of web service orchestrations to analyze process interactions of web service compositions. The aim of this analysis concentrates on the compatibility of processes that take part in the complete composition environment.

Modeling Component Connectors in Reo by Constraint Automata [9]

Coordination models and languages close the conceptual gap between the cooperation model used by the constituent parts of an application and the lower-level communication model used in its implementation. In [9], the authors introduce constraint automata as a formalism to describe the behavior and possible data flow in coordination models that connect anonymous components to enable their coordinated interaction. Constraint automata are used as an operational model for Reo, an exogenous coordination language for compositional construction of component connectors based on a calculus of channels. Constraint automata make modeling subtle timing and input/output constraints of Reo connectors possible, specifically their combined mix of synchronous and asynchronous transitions.

A Model-Checking Verification Environment for Mobile Processes [47]

A global computing system is a network of stationary and mobile components. The primary features of a global computing system are that its components are autonomous, software versioning is highly dynamic, the network's coverage is variable and often its components reside over the nodes of the network (WEB services), membership is dynamic and often ad hoc, without a centralized authority. Global computing systems must be made very robust since they are intended to operate in potentially hostile dynamic environments. The authors in [47] exploit History Dependent automata HD-automata as a basis for the design and development of verification toolkits for reasoning about the behavior of mobile systems. A verification environment, called HD-Automata Laboratory HAL, is used to exploits HD-automata of systems specified in the π-calculus. The HAL environment includes modules that implement decision procedures to calculate behavioral equivalences.

Describing and Reasoning on Web Services Using Process Algebra [121]

Web services are an emerging and promising area involving important technological challenges. Some of the main challenges are to correctly describe web services, to compose them adequately and/or automatically, and to discover suitable services working out a given problem. In their work, the authors propose a framework to that uses Process Algebra called CCS as an abstract representation means to describe, compose and reason (simulation, property verification, correctness of composition) on service-based applications. The techniques, used to check whether a service-based application described in process-algebraic notations respects temporal logic properties (e.g., safety and liveness properties), are referred to as model checking methods.

LTSA-WS: A Tool for Model-Based Verification of Web Service Compositions and Choreography [50]

Web service composition languages such as the BPEL4WS aim to fulfill the requirement of a coordinated and collaborative service invocation specification to support running transactions and multi-service scenarios. However, a composition alone does not fulfill the requirement of an assured collaboration in cross-enterprise service domains. Participating services must adhere to policies set out to support these collaborative roles with permissions and obligations constraining the interactions between services. The authors introduce LTSA-WS, as a tool implementing a model-based approach to verifying service-based applications. This tool supports verification of global properties (e.g., absence of deadlock and liveness) created from design specifications and implementation models to confirm expected results from the viewpoints of both the designer and implementer. Scenarios are modeled in UML, in the form of Message Sequence Charts, and then compiled into the Finite State Process FSP process algebra to concisely model the required behavior. BPEL4WS implementations are mechanically translated to FSP to allow an equivalence trace verification process to be performed.

Formal Verification of Web Service Composition [113]

Current Web services composition proposals, such as BPML, BPEL4WS, WSCI, and OWL-S, provide solutions for describing the control and data flows in Web service composition. However, such proposals remain at the descriptive level, without providing any kind of mechanisms or tool support for analysis and verification. The work presented in [113] proposes an event-based approach for checking both functional and non-functional requirements of web service compositions. The properties to be monitored are specified using the Event Calculus formalism. Functional requirements are initially extracted from the specification of the composition process that is expressed in WS-BPEL. This ensures that they can be expressed in terms of events occurring during the interaction between the composition process and the constituent services that can be detected from the execution log. Non-functional requirements (e.g., such as security policies) to be checked are subsequently defined in terms of the identified detectable events by service providers.

Execution Semantics for Service Choreographies [36]

A service choreography is a model of interactions in which a set of services engage to achieve a goal. Choreographies have been put forward as a starting point for building service-oriented systems since they provide a global picture of the system's behavior. In [36], the authors present a new approach that proposes to define a service interaction modeling language as well as techniques for analyzing and relating global and local models of service interactions. This work introduced a formal semantics for a service interaction modeling language, namely Let's Dance, which supports the high-level capture of both global models (i.e., choreographies) and local models of service interactions. The semantics is defined by translation to π-calculus.

Integrating Business Requirements and Business Processes [74, 73]

In order to support continuous changes and adaptation of business process models to business goals and requirements it is necessary to explicitly relate the strategic goals and requirements to the business process models. The ability to formally analyze how the changes in the requirements affect the process models enables the requirements traceability and improves the reliability of the system through its continuous evolution. The authors propose to explicitly associate the strategic models representing business requirements and goals to the business process models and then provide a formal analysis support for the verification of their compliance. In order to perform formal analysis of these models, the requirements specifications and the annotated process models are translated into an internal unified representation, and then the analysis is performed using model checking techniques.

WS-VERIFY: Formal Analysis of Web Service Compositions [76, 75, 72, 69]

The necessity to detect requirements violations and problems in the specifications of the composition behavior is an important issue in the service-oriented design and requires high level of formalisation and automation support. Such analysis, however, has to tackle several problems specific to service composition behavior. First, the service communications are essentially asynchronous and rely on complex message management systems. Second, the service specifications extensively use complex data structures and operations. Third, the correctness of a wide class of systems strongly relies on a compliance of the time-related composition requirements. In [76] a unified framework for the formal verification of Web service compositions is introduced. The framework integrates the methods and techniques for modeling and analyzing specific aspects, such as asynchronous interactions [75], data-flow properties [72] and qualitative/quantitative time characteristics [69].

Choreography Analysis [70, 71]

Service choreographies aim at representing global observable behavior of the collaborating services. The realizability problem addresses the possibility to "project" the choreography on the participants so that the behavior of the projection composition is guaranteed to correspond to the original choreography. The conformance analysis instead aims at checking whether the observable behavior generated by the service implementations corresponds to the choreography specification. These complementary forms of analysis are equally necessary for the correct implementation of the Web service collaborations. The authors present an approach that relies on a formal model that allows for defining both the prescribed choreography behavior and the behavior of the composition of the implementing services. The formalism is given in terms of state transition systems communicating through a certain model of message queues. In order to address the realizability problem, the authors present a hierarchy of realizability notions that allow one to efficiently analyze whether the given choreography can be implemented and under which conditions. In order to address the problem of conformance, the authors formally define the notion of conformance relation between the choreography and the composition implementation, extended with a set of constraints on the information alignment between partners.

Petri Net-Based Analysis of WS-BPEL Processes [105, 104, 150]

BPEL language is de-facto standard for implementing executable business processes on top of Web services. The necessity to carry out verification activities before the process deployment requires (*i*) correct and unambiguous language formalization and (*ii*) the techniques for the efficient and automated checking of relevant correctness properties.

The authors present a rigorous and detailed scheme for the formalization of BPEL structures. For this purpose a special class of Petri nets, called workflow nets, is adopted. In this way the formal semantics of BPEL is defined. Based on this formalism, the authors define two tools that in combination allow for the automated verification of BPEL. The BPEL2PNML tool translates the BPEL process into the Petri Net Modeling Language (PNML), and the resulting model is analyzed with the WofBPEL tool. Within the approach the following forms of analysis have been defined: detection of unreachable BPEL activities, detection of multiple concurrently enabled activities that may consume the same type of message, and determining for every reachable state of the process, which message types may be consumed in the rest of the execution.

Verification of Web Service Compositions with Service/Resource Nets[131]

In [131] Tang et al. target the analysis of the Web service composition workflows. In particular, the concurrent resource access and linear time analysis problems are studied. They introduce a special Petri Net-based formalism, called Service/Resource Net (SRN) that is able to capture not only the control flow of the composition workflow, but also other relevant aspects of the composition, such as time bounds of activities, resources (different data variables or services), and conditions. The authors show how the service composition may be represented in this formalism. When the composition is translated into the SRN model, and the target net is simplified it is possible to perform traditional analysis methods applicable to Petri Nets. Beyond these methods, it is possible to perform certain specific analysis techniques, namely resource matching and linear temporal inference.

CP-Net Based Verification for Web Service Compositions [155]

In Web service composition the integration of different services should be done efficiently and correctly. One of the key issues here is to provide a precise and reliable way of integrating conversation partner into the composition specification. Accordingly, the composition design framework should provide the means for the correctness analysis of such integration. To this end, the authors in [155] present a design and verification framework for service compositions based on the Coloured Petri Nets (CP-Nets) formalism. Given its expresiveness, the formalism allows for the representation of data types and data manipulations, as well as the concurrency issues and interprocess synchronization. The proposed formal model enables specification of both the complex conversation protocols of single partners and the overall composition model.

Semantic Based Verification of BPEL4WS Abstract Processes[42]

Abstract processes represent the behavioral interface of a composition component and therefore defines the restrictions on the component use. The abstract process corresponds to a workflow, where elementary tasks correspond to the basic Web service calls. When the preconditions and effects of the elementary tasks are known it is possible to check whether the whole process is compatible with this information, and, moreover, construct from elementary tasks new abstract workflows satisfying the required properties. In [42], the authors' solution relies on a logical model, which allows for associating preconditions and effects with the atomic service calls and with the complex control flow constructs. The preconditions and effects are given as boolean expressions over state propositions that hold in source and target states of the process respectively. Given a process model and a set of semantic annotations of the elementary tasks, it is possible to infer the precondition and effect of the whole process.

Verification of Data-Driven Web Services[38]

The behavioral verification of service composition problem is a complex analysis problem due to various features of the underlying service models, such as asynchronous and lossy interactions, data, object cardinality. These features may seriously restrict the applicability of any analysis techniques and should be carefully studied. In [38], the authors present a formalism for specifying compositions as a set of asynchronously interacting Web service peers. Each peer is represented with a local database, a control flow model, input output queues, and reaction rules. The composition The reaction is described by queries over the database, internal state, user input and received messages. The composition model is parametric with respect the queue bounds, message loss, openness, etc. The correctness properties of the composition have the form either of temporal first order logic sentences (e.g., ordering constraints on the action execution) or Büchi automata representing the expected conversation protocol. The authors discuss modular verification, where the correctness is checked when the complete specification of other peers is not available or partial. Modular verification is useful when some peers are provided by autonomous parties unwilling to disclose implementation details, or when verification of a partially specified composition is desired.

Automated Model Checking and Testing for Composite Web Services [65]

The process-oriented models of model checking approaches successfully capture the temporal properties among atomic WSs. However, if the internal structure of each atomic Web Service is blank in the model specification, then it is inherently hard to describe and check more delicate properties involving the effect and output of each atomic Web Service. In [65], the solution consists of the following steps: a) use OWL-S to bound the behavior of atomic WSs; b) convert OWL-S to the C-like language of BLAST [63]; c) enhance BLAST for checking concurrency in OWL-S; d) embed data-bound and safety temporal properties in the C-like code; e) use BLAST for model-checking and positive test case generation; f) use of the typological algorithm of [141] and the positive test cases as input for the generation of the corresponding negative test cases.

Simulation, Verification and Automated Composition of Web Services[99]

The success of the Web Service paradigm has led to a proliferation of available WSs. These WSs can be combined in a composition in order to perform a complex task. The big challenge is to describe and prove properties of these WSs in order to: a) test the composed system by simulating its execution under different input conditions; b) to logically verify certain maintenance and safety conditions associated with the composed WS; c) to automatically compose WSs. The authors take the DAML-S ontology for describing the capabilities of WSs and define the semantics for a relevant subset of DAML-S in terms of a first-order logical language. The basic idea for composing and verifying is to first map salient aspects of the DAML-S model into a situation calculus, and from there into the Petri net domain. Then, they relate the complexity of various Web Service tasks (simulation, verification, performance analysis) to the expressiveness of DAML-S by providing proofs for their claims. Most importantly, they show that the complexity of the reachability problem for DAML-S Process Models is PSPACE-complete.

Towards a Formal Verification of OWL-S Process Models[2]

Verification of the interaction protocol of WSs can be used to prove that the protocol to be advertised is indeed correct (e.g., does not contain deadlocks) and to guarantee additional properties, e.g., purchased goods are not delivered if a payment is not received. In [2], the authors work extends the work analyzed in [99] in three directions. First, a model of Web Service data flow is provided in addition to control flow. Second, an OWL-S process model is translated into a simpler model of the PROMELA specification language [64] that preserves all the essential behavior to be verified. Third, the PROMELA specifications are fed into the SPIN [64] model checking tool for automatically verifying that the interaction protocol satisfies the claims.

Towards Efficient Verification for Process Composition of Semantic Web Services[85]

OWL-S provides no way to validate a Web Service composition. In practice, Web Service composition is usually a complex and error-prone process whether happening at design time or at runtime. Moreover, if an erroneous composition plan is executed without being previously verified, deployment of the Web Service composition process often results in runtime errors, which need to be repaired on-the-fly at high costs. The authors propose an analysis and verification technique based on Colored Petri Nets (CPNets) [67].The main idea is to define a composite process in a three-level specification: *interface net*, *orchestration net* and *composition net*, and to specify the control and data flow by mapping control constructs of OWL-S into the CPNet representation. Then the constructed CPNet model is simulated and validated to detect the composition errors, such as deadlocks, and to verify whether the composition process does have certain expected dynamic properties.

7.7.3 Classification of Analysis Techniques

The various approaches that have been discussed above are classified and summarized (according to the classification scheme introduced in Section 7.4) in tables 7.5 and 7.6.

Table 7.5. Table-A for Analysis

Contribution	*Class*	*Quality*	*Life-Cycle*	*Discipline*	*Layer*
[57, 58]	analysis (verification)	QoS (scalability, security)	design	SE, SOC	SCC
[26]	analysis (synthesis)	code complexity	design	SE, SOC	SCC
[116]	analysis (synthesis)	QoS (many)	design	SE, SOC	SCC
[91]	analysis (synthesis)	QoS (many)	design, execution	SE, SOC	SCC
[3]	analysis (synthesis)	QoS (many)	design, execution	SOC, SE	SCC
[52]	analysis (verification)	functional behavior	design	SE, SOC	SCC
[21]	analysis (verification)	behavior	design	SE, SOC	SCC
[94]	analysis (verification)	Deadlock, Timeouts, Reachability	Execution	SOC and BPM	SCC
[146, 148, 114]	analysis (verification)	Correctness, Conformance	Design, Execution	BPM	SCC
[123]	analysis (synthesis)	Usabiliy	Design	SOC	SCC
[98]	analysis (verification)	Dataflows properties (e.g., deadlocks, reachability)	Design	SE, SOC	SCC
[80]	analysis (verification)	Deadlocks	Execution	BPM, SOC	SCC
	Verification and Validation	SE, SOC	SCC		
[125]	analysis (verfication)	Safety properties	Execution	SOC	SCC
[153]	analysis (synthesis)	Exchanged message properties	Design	BPM, SOC	SCC
[45]	analysis (verification)	Correlation and synchronous receive/reply messages	Execution	BPM, SOC	SCC
[49]	analysis (verification)	Interface Compatibility, Safety Compatibility, Liveness Compatibility	Design	BPM, SOC	SCC
[9]	analysis (verification)	Synchronous and asynchronous transition properties	Design	SE, SOC	SCC
[47]	analysis (verification)	Behavioral and safety properties	Design	SE, SOC	SCC
[121]	analysis (verification)	Safety and liveness properties	Design	SE, SOC	SCC

Table 7.5. (continued)

Contribution	*Class*	*Quality*	*Life-Cycle*	*Discipline*	*Layer*
[50]	analysis (verification)	Deadlock, liveness	Design, execution	SOC	SCC
[113]	analysis (verification)	Security policies	Execution	SOC	SCC
[36]	analysis (synthesis)	Reachability	Design, execution	SOC	SCC
[74, 73]	analysis (verification)	behavioral correctness	design	BPM, RE	BPM, SCC
[76, 75, 72, 69]	analysis (verification)	behavioral correctness	design	BPM, SOC	SCC
[70, 71]	analysis (verification)	behavioral correctness	design	SOC	SCC
[105, 104, 150]	analysis (verification)	behavioral correctness	design	BPM	SCC, BPM
[131]	analysis (verification)	behavioral correctness	design	SOC	SCC
[155]	analysis (verification)	protocol conformance	design	SOC	SCC
[19]	analysis (verification)	behavioral correctness	design, execution	SOC	SCC, SI
[42]	analysis (verification)	behavioral correctness	design	SOC	SCC
[38]	analysis (verification)	behavioral correctness	design	SOC	SCC
[34]	analysis (verification)	behavioral correctness	design	SE, SOC	SCC
[65]	analysis (verification)	behavioral correctness	design	SE, SOC	SCC
[99]	analysis (verification), simulation	behavioral correctness	design	SE, SOC	SCC
[2]	analysis (verification)	behavioral correctness	design	SE, SOC	SCC
[85]	analysis (verification), simulation	behavioral correctness	design	SOC	SCC

Table 7.6. Table-B for Analysis

Contribution	*Artifact*	*Reference*	*Formality*	*Automation*	*Evaluation*
[57, 58]	colored UML statecharts, sequence diagrams	QoS requirements	formal	automated	explor. case studies
[26]	WS-BPEL descriptions	developer interpretation, complexity standards	formal	automated	explor. case study
[116]	WS-BPEL descriptions	SLAs	formal	automated	not evaluated
[91]	BPEL, WSDL descritions	QoS requirements	formal	automated	explor. case study
[3]	BPEL descriptions	QoS parameters	formal	automated	test cases
[52]	BPEL descriptions	LTL properties	formal	automated	explor. case study

Table 7.6. (continued)

Contribution	*Artifact*	*Reference*	*Formality*	*Automation*	*Evaluation*
[21]	BPEL4WS compositions	LTL properties, WS-CoL, deadlock freedom	formal	automated	explor. case studies
[94]	Petri nets	Orchestration schemas	formal	automated	explor. case study
[146, 148, 114]	Petri nets	Workflow models	formal	automated	explor. case study
[123]	Petri nets	BPEL4WS specifications	formal	not automated	-
[98]	Promela processes, LTL formula	WSFL specifications	formal	automated	explor. case study
[80]	BPE-calculus	BPEL4WS specifications	formal	automated	empirical evaluations
[125]	PHP	Safety properties	formal	automated	-
[153]	aDFA	BPEL4WS specifications	formal	automated	explor. case study
[45]	DASM	BPEL4WS specifications	formal	automated	explor. case study
[49]	LTS	BPEL4WS specifications	formal	automated	explor. case study
[9]	Constraint automata	Graphs	formal	automated	theoretical proves
[47]	HD-automata	π-calculus processes	formal	automated	explor. case studies
[121]	CCS	BPEL4WS specifications	formal	automated	explor. case studies
[50]	FSP	BPEL4WS specifications	formal	automated	empirical case studies
[113]	Event Calculus	BPEL4WS specifications	formal	not automated	empirical case studies
[36]	π-calculus	Let's Dance	formal	not automated	Theoretical proves
[74, 73]	business process specification in BPEL	formal Tropos requirements specifications	formal	automated	explor. Case Study
[76, 75, 72, 69]	BPEL descriptions	behavioral composition requirements	formal	automated	explor. case studies
[70, 71]	composition of stateful services	choreography specification	formal	automated	Simple Scenarios
[105, 104, 150]	BPEL descriptions	predefined execution properties	formal	automated	explor. case study
[131]	formal model of the composition specification	predefined execution properties	formal	automated	explor. case study
[155]	formal model of the service composition	generic correctness properties, and conversation protocol of a participant	formal	automated	explor. case study

Table 7.6. (continued)

Contribution	*Artifact*	*Reference*	*Formality*	*Automation*	*Evaluation*
[19]	WS interface models	other Web service interface models	formal	automated	Simple Scenario, Theoretical proves
[42]	abstract process models	elementary service preconditions and process postconditions (goals)	formal	automated	not evaluated
[38]	WS composition model	temporal properties (LTL), conversation protocols	formal	not automated	not evaluated
[34]	protocols of the composition participants	dynamic properties of service contracts	formal	automated	explor. case Study
[65]	OWL-S Composite WS Descriptions	temporal properties	formal	automated	explor. case study
[99]	AML-S Composite WS Descriptions	safety conditions	formal	automated	explor. case study
[2]	OWL-S Composite WS Descriptions	temporal properties	formal	automated	explor. case study
[85]	OWL-S Composite WS Descriptions	reachability, liveness, fairness properties	formal	automated	explor. case study

7.8 Observations and Future Research Directions

Based on the results of the literature review, this section discusses key research challenges in quality assurance for services and service-based applications and how they have been addressed in the literature thus far.

Concerning testing, the reviewed papers and their classification show several areas of interest in the testing community. The majority of the approaches that has been reviewed addresses the problem of generating test cases for testing services and service-based applications. This includes deriving test cases from service descriptions ("black box"), based on WSDL for instance, and deriving test cases from service compositions ("white box"), based on BPEL for example. Approaches are presented for deriving (or generating) test inputs, as well for determining the expected outputs (test oracles). In addition, several of these approaches include techniques for executing the test cases, once these have been defined.

Concerning monitoring, the reviewed papers provide very rich and comprehensive set of approaches that cover wide range of goals, problem aspects, and information types, as well as the different components of the SBA architecture. The monitoring problem is considered from various stakeholder perspectives and it addresses functional and also QoS aspects.

Concerning analysis, the literature review results show that a lot of research effort has been spent by the community to build the current state of the art in analysis of service-based applications from a quality assurance point of view. Different formal approaches, ranging from Petri nets to process algebras, and different quality characteristics, ranging from non-functional to functional properties, and for different forms of service-based applications specifications, have been proposed and analyzed by the research community.

Run-Time Quality Assurance

Services are often provisioned in the context of short-term, volatile and thus highly dynamic business relationships between service providers and requestors which are not known at design time. Thus, services will have to be enabled to collaborate in highly distributed environments, cutting across the boundaries of various organizations (including enterprises, governmental and non-profit organizations). The aggregation of services to build service-based applications is very likely to happen mostly at run-time in the near future. Which means that the actual functional behavior and quality aspects of the service-based application will have to be determined during its actual operation. Moreover, determining the relevant aspects of the context in which the service-based application is executed becomes a run-time issue. As an example, the types of users which interact with the service-based application, or the actual situation or location in which the application is operated is only determined at run-time.

There is thus a strong need for quality assurance techniques that can be applied while the service-based application is in operation. The review has uncovered, that currently typically monitoring techniques are proposed for assuring the quality of an application during its operation. The problem with monitoring is that it only checks the current (actual) execution. It does not allow to pro-actively uncover faults which are introduced, e.g., due to a change in the application, if they are not leading to a failure in the current execution. Other techniques, such as testing or static analysis, examine sets of classes of executions and thus would allow to pro-actively uncover such faults before they lead to an actual failure during the operation of the system. Therefore, standard and consolidated “off-line” software quality assurance techniques (like testing and analysis) should be extended such that they are applicable while the application operates (“on-line” techniques).

There are some contributions on regression testing for service-based applications. A regression test, i.e., re-testing the application, is essential due to its dynamic nature (the service composition can change due to the evolution of the services or other changes in the context of the service-based application). Interestingly, although dynamics as an important characteristic of service-based applications is acknowledged and many of the testing techniques support the automatic generation and execution of test cases, there are but a few contributions that propose performing tests during the actual operation of the system ('on-line' testing).

One important requirement of those "on-line" techniques, however, is that their overhead and costs should not be an overkill to the point that they become unpractical for this reason. Overheads and costs that incur under any execution environment typically include time, computational resources (e.g., processing power and memory) as well as communication resources (e.g., network bandwidth). Therefore, work on making the "on-line" techniques as light-weight as possible is needed.

Finally, as (self-)adaptation of service-based applications becomes an essential characteristic, there is a strong need to assure that the adaptation of a service-based application behaves as expected. This requires specific testing and analysis techniques to verify the adaptation behavior; e.g., this could include checking whether the behavior after the adaptation conforms to the expected behavior before the adaptation took place (in order to keep backwards compatibility). Those techniques and methods to assure the correctness of adaptations are badly lacking at the moment.

Assuring End-to-End Quality

In a dynamic business scenario, the contract life-cycle should be automated as much as possible, in order to allow organizations to dynamically change service providers (business partners) or to re-negotiate SLAs (see above). That requires that QoS aspects need to be checked during the operation, e.g., by monitoring the QoS characteristics, in order to determine whether the new service provider meets the desired QoS or whether there is a need for renegotiating the SLAs. As this literature review has revealed, there are only few and isolated research contributions on assuring QoS aspects. There is thus a strong need for novel techniques and methods that address QoS characteristics in a comprehensive and end-to-end fashion across all layers of a service-based application. In addition, approaches that consider the context of a service-based application and its impact on QoS are needed in order to pave the way towards context-aware service-based application.

Due to the dynamic world in which service-based applications operate, techniques are needed to aggregate individual QoS levels of the services involved in a service composition in order to determine and thus check the end-to-end QoS during run-time. This aggregation will typically span different layers of a service-based application and thus a common understanding of what the different QoS characteristics mean within and across these layers is needed (also see above). In fact, the definition of a QoS taxonomy is addressed within S-Cube by defining a "Quality reference model for service-based applications" (see Chapter 6).

Another important trend is to consider more and more aspects during the monitoring task, such as different types of information, sources, types of event, their distribution. However, only few approaches go beyond a particular monitoring problem or provide a wider perspective on the application execution. Furthermore, there are no existing approaches that cross all the functional layers of a service-based application, consider evolutionary aspects of executions with respect to a variety of information, etc.

Only very few isolated testing approaches provide solutions for considering specific QoS characteristics. However, as it has been motivated in the introduction to this chapter, QoS is an important aspect of the contracts between service providers and consumers. Thus, assuring whether the services conform with the agreed levels of quality is an essential activity.

Despite the great effort in the area of analysis, an evident problem can be found in the lack of integrated methods for the overall analysis of functional and quality properties of service-based applications. In other words, a set of comprehensive, integrated techniques and methods able to incorporate different aspects of quality analysis at different levels of abstraction are needed. Such techniques should take into account the specific layers of a service-based application, and face coherence issues among different models and aspects of the same application.

Synergies between Approaches

The literature review results have shown that first attempts are made to exploit potential synergies between the different classes of analytical quality assurance techniques. As an example, testing can be used as preventive mechanism for finding faults while monitoring is used during the operation of the service-based application. A combination of both techniques can compensate the weaknesses of the single approaches [25, 24]. Further examples are the use of dedicated tests during the operation of the service-based applications in order to determine the conformance to SLAs, the use of static analysis tools (like model checkers) to derive test cases, or the analysis of monitoring logs ("post-mortem" analysis or audit).

Despite these initial attempts, synergies that can be achieved by joining and integrating different kinds of techniques have not been fully exploited. For example, research can be directed at facilitating the use of monitoring results as input for run-time verification. Or, testing could be combined with monitoring in such a way that when a deviation is observed during monitoring, dedicated test cases are executed in order to pinpoint the issues that lead to the deviation (also see next sub-section).

Debugging and Diagnosis

In this chapter, the focus has been on dynamic techniques and methods for uncovering failures and on static techniques for uncovering faults. Obviously, once a failure has been identified, the cause for this failure, i.e., the fault in the artifact, needs to be uncovered. This is the realm of debugging and diagnosis. In this area, we see many open issues in the context of quality assurance. Specifically, this leads to issues on how one can interact with services for the purpose of debugging without this interaction having side-effects on the current execution (actual operation) of the service-based application. This also means that current service interfaces must be enhanced for testability, diagnosis and debuggability, which however must be well balanced with desired characteristics such as information hiding, encapsulation and loose coupling.

* * *

This chapter gave an overview of this broad field of “service quality assurance” and identified the key areas where research contributions are currently available. Based on this review of the state of the art, important and emerging research challenges were highlighted. In S-Cube, the research challenges “run-time quality assurance”, “end-to-end quality” and “synergies between approaches” are addressed.

References

1. Andreozzi, S., De Bortoli, N., Fantinel, S., Ghiselli, A., Rubini, G.L., Tortone, G., Vistoli, M.C.: GridICE: a monitoring service for grid systems. Future Generation Computer Systems 21(4), 559–571 (2005)
2. Ankolekar, A., Paolucci, M., Sycara, K.: Towards a formal verification of OWL-S process models. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 37–51. Springer, Heidelberg (2005)
3. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Transactions on Software Engineering 33(6), 369–384 (2007)
4. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: Fault tolerant web service orchestration by means of diagnosis. In: Gruhn, V., Oquendo, F. (eds.) EWSA 2006. LNCS, vol. 4344, pp. 2–16. Springer, Heidelberg (2006)
5. Bai, X., Chen, Y., Shao, Z.: Adaptive web services testing. In: 31st Annual International Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 233–236 (2007)
6. Bai, X., Xu, D., Dai, G., Tsai, W., Chen, Y.: Dynamic reconfigurable testing of service-oriented architecture. In: Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC), vol. 1, pp. 368–375 (2007)
7. Bai, X., Dai, G., Xu, D., Tsai, W.-T.: A multi-agent based framework for collaborative testing on Web services. In: The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance.SEUS 2006/WCCIA 2006, p. 6 (2006)

8. Bai, X., Dong, W., Tsai, W.-T., Chen, Y.: WSDL-Based Automatic Test Case Generation for Web Services Testing. In: Proceedings of the IEEE International Workshop on Service-Oriented System Engineering (SOSE), pp. 215–220. IEEE Computer Society (2005)
9. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in reo by constraint automata. Sci. Comput. Program. 61(2), 75–113 (2006)
10. Barbon, F., Traverso, P., Pistore, M., Trainotti, M.: Run-Time Monitoring of Instances and Classes of Web Service Compositions. In: IEEE International Conference on Web Services (ICWS 2006), pp. 63–71 (2006)
11. Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: A Timed Extension of WSCoL. In: 2007 IEEE International Conference on Web Services (ICWS 2007), pp. 663–670 (2007)
12. Baresi, L., DiNitto, E.: Test and Analysis of Web Services. Springer, Heidelberg (2007)
13. Baresi, L., Ghezzi, C., Guinea, S.: Smart Monitors for Composed Services. In: Service-Oriented Computing – ICSOC 2004, Second International Conference, pp. 193–202 (2004)
14. Baresi, L., Guinea, S.: Towards Dynamic Monitoring of WS-BPEL Processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 269–282. Springer, Heidelberg (2005)
15. Baresi, L., Guinea, S., Plebani, P.: WS-Policy for Service Monitoring. In: Technologies for E-Services, 6th International Workshop, TES 2005, pp. 72–83 (2005)
16. Beeri, C., Eyal, A., Milo, T., Pilberg, A.: Monitoring Business Processes with Queries. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 603–614 (2007)
17. Benbernou, S., Meziane, H., Hacid, M.S.: Run-time monitoring for privacy-agreement compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 353–364. Springer, Heidelberg (2007)
18. Bertolino, A., Polini, A.: The audition framework for testing Web services interoperability. In: Proceedings. 31st Euromicro Conference on Software Engineering and Advanced Applications, pp. 134–142 (2005)
19. Beyer, D., Chakrabarti, A., Henzinger, T.A.: Web Service Interfaces. In: Proceeding of the International Conference on World Wide Web, WWW (2005)
20. Bianculli, D., Ghezzi, C.: Monitoring Conversational Web Services. In: IW-SOSWE'07 (2007)
21. Bianculli, D., Ghezzi, C., Spoletini, P.: A model checking approach to verify BPEL4WS workflows. In: Proceedings of the 2007 IEEE International Conference on Service-Oriented Computing and Applications (IEEE SOCA 2007), Newport Beach, USA, June 2007, pp. 13–20. IEEE Computer Society Press, Los Alamitos (2007)
22. Bruno, M., Canfora, G., Di Penta, M., Esposito, G., Mazza, V.: Using Test Cases as Contract to Ensure Service Compliance Across Releases. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 87–100. Springer, Heidelberg (2005)
23. Bryce, R.C., Chen, Y., Colbourn, C.J.: Biased covering arrays for progressive ranking and composition of Web Services. International Journal of Simulation and Process Modelling 3(1-2), 80–87 (2007)

24. Canfora, G., di Penta, M.: SOA: Testing and Self-checking. In: Proceedings of International Workshop on Web Services – Modeling and Testing – WS-MaTE, pp. 3–12 (2006)
25. Canfora, G., di Penta, M.: Testing Services and Service-Centric Systems: Challenges and Opportunities. IT Professional 8(2), 10–17 (2006)
26. Cardoso, J.: Complexity analysis of BPEL web processes. Software Process: Improvement and Practice 12(1), 35–49 (2007)
27. Castellanos, M., Casati, F., Shan, M.-C., Dayal, U.: iBOM: A Platform for Intelligent Business Operation Management. In: ICDE '05: Proceedings of the 21st International Conference on Data Engineering, pp. 1084–1095 (2005)
28. Chan, W.K., Cheung, S.C., Leung, K.R.P.H.: A metamorphic testing approach for online testing of service-oriented software applications. International Journal of Web Services Research 4(2), 61–81 (2007)
29. Chowdhary, P., Bhaskaran, K., Caswell, N.S., Chang, H., Chao, T., Chen, S.-K., Dikun, M., Lei, H., Jeng, J.-J., Kapoor, S., Lang, C.A., Mihaila, G., Stanoi, I., Zeng, L.: Model Driven Development for Business Performance Management. IBM Syst. J. 45(3), 587–605 (2006)
30. Curbera, F.: Components contracts in Service-Oriented architectures. IEEE Computer 11, 74–80 (2007)
31. Curbera, F., Duftler, M.J., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Colombo: Lightweight Middleware for Service-Oriented Computing. IBM Systems Journal 44(4), 799–820 (2005)
32. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 2001), pp. 181–194 (2001)
33. Dai, G., Bai, X., Wang, Y., Dai, F.: Contract-based testing for web services. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 1, pp. 517–524 (2007)
34. Davulcu, H., Kifer, M., Ramakrishnan, I.V.: CTR-S: A Logic for Specifying Contracts in Semantic Web Services. In: Proceeding of the International Conference on World Wide Web (WWW), pp. 144–153 (2004)
35. Júnior de Almeida, L.F., Vergilio, S.R.: Exploring Perturbation Based Testing for Web Services. In: IEEE International Conference on Web Services (ICWS), pp. 717–726 (2006)
36. Decker, G., Zaha, J.M., Dumas, M.: Execution semantics for service choreographies. In: WS-FM, pp. 163–177 (2006)
37. Delgado, N., Gates, A.Q., Roach, S.: A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans. Software Eng. 30(12), 859–872 (2004)
38. Deutsch, A., Sui, L., Vianu, V., Zhou, D.: Verification of Communicating Data-driven Web Services. In: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), pp. 90–99 (2006)
39. Devaurs, D., Musaraj, K., De Marchi, F., Hacid, M.-S.: Timed Transition Discovery from Web Service conversation Logs. In: 20th International Conference on Advanced Information Systems Engineering, CAISE'08 (2008)
40. Di Penta, M., Canfora, G., Esposito, G., Mazza, V., Bruno, M.: Search-based Testing of Service Level Agreements. In: Proceedings of the Conference on Genetic and Evolutionary Computation – GECCO, pp. 1090–1097. ACM Press, New York (2007)

41. Dong, W.-L., Yu, H., Zhang, Y.-B.: Testing BPEL-based Web Service Composition Using High-level Petri Nets. In: EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference, pp. 441–444. IEEE Computer Society (2006)
42. Duan, Z., Bernstein, A.J., Lewis, P.M., Lu, S.: Semantics Based Verification and Synthesis of BPEL4WS Abstract Processes. In: Proceeding of the International Conference on Web Services (ICWS), pp. 734–737 (2004)
43. Dustdar, S., Haslinger, S.: Testing of service-oriented architectures – A practical approach. In: Weske, M., Liggesmeyer, P. (eds.) NODe 2004. LNCS, vol. 3263, pp. 97–109. Springer, Heidelberg (2004)
44. Hankin, C., Nielson, F., Nielson, H.R.: Principles of Program Analysis, 2nd edn. Springer, Heidelberg (2005)
45. Farahbod, R., Glässer, U., Vajihollahi, M.: Specification and validation of the business process execution language for web services. In: Abstract State Machines, pp. 78–94 (2004)
46. Farrell, A., Sergot, M., Bartolini, C., Salle, M., Trastour, D., Christodoulou, A.: Using the Event Calculus for the Performance Monitoring of Service-Level Agreements for Utility Computing. In: Proceedings of First IEEE International Workshop on Electronic Contracting, WEC 2004 (2004)
47. Ferrari, G.-L., Gnesi, S., Montanari, U., Pistore, M.: A model-checking verification environment for mobile processes. ACM Trans. Softw. Eng. Methodol. 12(4), 440–473 (2003)
48. Fisher, S.: Relational Model for Information and Monitoring. Technical Report GWD-GP-7-1, Global Grid Forum (2001)
49. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Compatibility verification for web service choreography. In: ICWS '04: Proceedings of the IEEE International Conference on Web Services, Washington, DC, USA, p. 738. IEEE Computer Society (2004)
50. Foster, H., Uchitel, S., Magee, J., Kramer, J.: LTSA-WS: a tool for model-based verification of web service compositions and choreography. In: ICSE '06: Proceedings of the 28th international conference on Software engineering, New York, NY, USA, pp. 771–774. ACM (2006)
51. Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J., von Reich, J.: The Open Grid Services Architecture, Version 1.0. Technical Report GFD-I.030, Global Grid Forum (January 2005)
52. Fu, X., Bultan, T., Su, J.: Analysis of interacting BPEL web services. In: Proceedings of the 13th International World Wide Web Conference, WWW'04 (2004)
53. Fugini, M.G., Pernici, B., Ramoni, F.: Quality analysis of composed services through fault injection. Information System Frontiers, Special Issue on Collaborative Business Processes (in press)
54. Garcia-Fanjul, J., de la Riva, C., Tuya, J.: Generation of Conformance Test Suites for Compositions of Web Services Using Model Checking. In: TAIC-PART '06: Proceedings of the Testing: Academic & Industrial Conference on Practice And Research Techniques, pp. 127–130. IEEE Computer Society (2006)
55. Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering. Prentice-Hall, Englewood Cliffs (1991)

56. Ghezzi, C., Guinea, S.: Run-Time Monitoring in Service-Oriented Architectures. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 237–264. Springer, Heidelberg (2007)
57. Gilmore, S., Haenel, V., Kloul, L., Maidl, M.: Choreographing security and performance analysis for web services. In: Bravetti, M., Kloul, L., Zavattaro, G. (eds.) EPEW/WS-EM 2005. LNCS, vol. 3670, pp. 200–214. Springer, Heidelberg (2005)
58. Gilmore, S., Kloul, L.: A unified tool for performance modelling and prediction. Reliability Engineering and System Safety 89, 17–32 (2005)
59. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
60. Hanna, S., Munro, M.: An approach for specification-based test case generation for Web services. In: IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2007, pp. 16–23 (2007)
61. Heckel, R., Lohmann, M.: Towards Contract-based Testing of Web Services. In: Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004). Electronic Notes in Theoretical Computer Science, vol. 116, pp. 145–156. Elsevier B.V., Amsterdam (2005)
62. Heckel, R., Mariani, L.: Automatic conformance testing of web services. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 34–48. Springer, Heidelberg (2005)
63. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT Annual Symposium on Principles of Programming Languages, Portland, Oregon, USA, pp. 58–70. ACM (2002)
64. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual, 2003. Addison-Wesley Professional, Reading (Sept. 2003)
65. Huang, H., Tsai, W.-T., Paul, R., Chen, Y.: Automated Model Checking and Testing for Composite Web Services. In: 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005), pp. 300–307. IEEE Computer Society (2005)
66. Jeng, J.-J., Schiefer, J., Chang, H.: An Agent-based Architecture for Analyzing Business Processes of Real-Time Enterprises. In: EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing, p. 86 (2003)
67. Jensen, K. (ed.): Coloured Petri Nets – Basic concepts, analysis methods and practical use, vol. 1, 2nd edn. Monographs in Theoretical Computer Science. Springer, Heidelberg (1997)
68. Karam, M., Safa, H., Artail, H.: An abstract workflow-based framework for testing composed web services. In: International Conference on Computer Systems and Applications (AICCSA), pp. 901–908 (2007)
69. Kazhamiakin, R., Pandya, P.K., Pistore, M.: Representation, Verification, and Computation of Timed Properties in Web Service Compositions. In: Proceeding of the International Conference on Web Services (ICWS), pp. 497–504 (2006)
70. Kazhamiakin, R., Pistore, M.: Analysis of Realizability Conditions for Web Service Choreographies. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 61–76. Springer, Heidelberg (2006)

71. Kazhamiakin, R., Pistore, M.: Choreography Conformance Analysis: Asynchronous Communications and Information Alignment. In: Proceedings of the International Workshon on Web Services and Formal Methods (WS-FM), pp. 227–241 (2006)
72. Kazhamiakin, R., Pistore, M.: Static Verification of Control and Data in Web Service Compositions. In: Proceeding of the International Conference on Web Services, ICWS (2006)
73. Kazhamiakin, R., Pistore, M., Roveri, M.: Formal Verification of Requirements using SPIN: A Case Study on Web Services. In: Proceedings of the International Conference on Software Engineering and Formal Methods (SEFM), pp. 406–415 (2004)
74. Kazhamiakin, R., Pistore, M., Roveri, M.: A Framework for Integrating Business Processes and Business Requirements. In: Proceedings of the International Enterprise Distributed Object Computing Conference (EDOC), pp. 9–20 (2004)
75. Kazhamiakin, R., Pistore, M., Santuari, L.: Analysis of Communication Models in Web Service Compositions. In: Proceeding of the International Conference on World Wide Web, WWW (2006)
76. Kazhamiakin, R.: Formal Analysis of Web Service Compositions. PhD thesis, University of Trento (2007)
77. Keller, A., Ludwig, H.: The WSLA framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management 11(1), 57–81 (2003)
78. Keum, C., Kang, S., Ko, I.-Y., Baik, J., Choi, Y.-I.: Generating test cases for web services using extended finite state machine. In: Uyar, M.Ü., Duale, A.Y., Fecko, M.A. (eds.) TestCom 2006. LNCS, vol. 3964, pp. 103–117. Springer, Heidelberg (2006)
79. Kloukinas, C., Spanoudakis, G.: A pattern-driven framework for monitoring security and dependability. In: Lambrinoudakis, C., Pernul, G., Tjoa, A.M. (eds.) TrustBus. LNCS, vol. 4657, pp. 210–218. Springer, Heidelberg (2007)
80. Koshkina, M., van Breugel, F.: Modelling and verifying web service orchestration by means of the concurrency workbench. SIGSOFT Softw. Eng. Notes 29(5), 1–10 (2004)
81. Lazovik, A., Aiello, M., Papazoglou, M.P.: Associating Sssertions with Business Processes and Monitoring their Execution. In: Service-Oriented Computing – ICSOC 2004, Second International Conference, pp. 94–104 (2004)
82. Lohmann, M., Mariani, L., Heckel, R.: A Model-Driven Approach to Discovery, Testing and Monitoring of Web Services, pp. 173–204. Springer, Heidelberg (2007)
83. Ludwig, H., Dan, A., Kearney, R.: Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. In: Service-Oriented Computing – ICSOC 2004, Second International Conference, pp. 65–74 (2004)
84. Luebke, D.: Unit Testing BPEL Compositions, pp. 149–171. Springer, Heidelberg (2007)
85. Luo, N., Yan, J., Liu, M.: Towards efficient verification for process composition of semantic web services. In: IEEE SCC, pp. 220–227. IEEE Computer Society (2007)
86. Mahbub, K., Spanoudakis, G.: Run-time Monitoring of Requirements for Systems Composed of Web Services: Initial Implementation and Evaluation Experience. In: 2005 IEEE International Conference on Web Services (ICWS 2005), pp. 257–265 (2005)

87. Mahbub, K., Spanoudakis, G.: Monitoring WS-Agreements: An Event Calculus-Based Approach. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 265–306. Springer, Heidelberg (2007)
88. Martin, E., Basu, S., Xie, T.: Automated Robustness Testing of Web Services. In: Proc. 4th International Workshop on SOA And Web Services Best Practices, SOAWS 2006 (2006)
89. Martin, E., Basu, S., Xie, T.: Automated Testing and Response Analysis of Web Services. In: IEEE International Conference on Web Services (ICWS), pp. 647–654 (2007)
90. Martin, E., Basu, S., Xie, T.: WebSob: A tool for robustness testing of web services. In: Companion to the proceedings of the 29th International Conference on Software Engineering (ICSE), pp. 65–66 (2007)
91. Marzolla, M., Mirandola, R.: Performance prediction of web service workflows. In: Overhage, S., Szyperski, C., Reussner, R., Stafford, J.A. (eds.) QoSA 2007. LNCS, vol. 4880, pp. 127–144. Springer, Heidelberg (2008)
92. Mayer, P., Luebke, D.: Towards a BPEL unit testing framework. In: Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications, TAV WEB'06, volume 2006, pp. 33–42 (2006)
93. McGregor, J.D., Sykes, D.A.: A Practical Guide to Testing Object-oriented Software. Addison-Wesley Professional, Reading (2001)
94. Mecella, M., Parisi Presicce, F., Pernici, B.: Modeling *E*-service orchestration through petri nets. In: Buchmann, A., Casati, F., Fiege, L., Hsu, M.-C., Shan, M.-C. (eds.) TES 2002. LNCS, vol. 2444, pp. 38–47. Springer, Heidelberg (2002)
95. Mei, H., Zhang, L.: A Framework for Testing Web Services and Its Supporting Tool. In: SOSE '05: Proceedings of the IEEE International Workshop, pp. 207–214. IEEE Computer Society (2005)
96. Momm, C., Malec, R., Abeck, S.: Towards a Model-driven Development of Monitored Processes. Wirtschaftsinformatik 2 (2007)
97. Myers, G.J.: The Art of Software Testing. Wiley, Chichester (2004)
98. Nakajima, S.: Model-checking verification for reliable web service. In: OOPSLA Workshop on Object-Oriented Web Services (2002)
99. Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, Honolulu, Hawaii, USA, pp. 77–88. ACM (2002)
100. Newman, H.B., Legrand, I.C., Galvez, P., Voicu, R.: MonALISA: A Distributed Monitoring Service Architecture. In: International Conference on Computing in High Energy Physics, CHEP2003 (2003)
101. Nezhad, H.R.M., Saint-Paul, R., Benatallah, B., Casati, F.: Deriving Protocol Models from Imperfect Service Conversation Logs. In: IEEE Transactions on Knowledge and Data Engineering (TKDE), to appear (2008)
102. Offutt, J., Xu, W.: Generating Test Cases for Web Services Using Data Perturbation. In: Workshop on Testing, Analysis and Verification of Web Services (2004)
103. Osterweil, L.J.: Strategic directions in software quality. ACM Comput. Surv. 28(4), 738–750 (1996)
104. Ouyang, C., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M., Verbeek, H.M.W.: Formal Semantics and Analysis of Control Flow in WS-BPEL. Technical report, BPMcenter.org, BPM Center Report BPM-05-15 (2005)

105. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M.: WofBPEL: A tool for automated analysis of BPEL processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 484–489. Springer, Heidelberg (2005)
106. Paradkar, A.M., Sinha, A., Williams, C., Johnson, R.D., Outterson, S., Shriver, C., Liang, C.: Automated Functional Conformance Test Generation for Semantic Web Services. In: IEEE International Conference on Web Services, ICWS 2007, pp. 110–117 (2007)
107. Di Penta, M., Bruno, M., Esposito, G., Mazza, V., Canfora, G.: Web Services Regression Testing, pp. 205–234. Springer, Heidelberg (2007)
108. Pistore, M., Traverso, P.: Assumption-Based Composition and Monitoring of Web Services. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 307–335. Springer, Heidelberg (2007)
109. Radha Krishna, P., Karlapalem, K., Chiu, D.K.W.: An ER^{EC} framework for e-contract modeling, enactment, and monitoring. Data Knowl. Eng. 51, 31–58 (2004)
110. Ramsokul, P., Sowmya, A., Ramesh, S.: A test bed for web services protocols. In: Second International Conference on Internet and Web Applications and Services, ICIW (2007)
111. Rosario, S., Beneveniste, A., Haar, S., Jard, C.: Probablistic QoS and soft contracts for transaction based web services. In: IEEE ICWS, pp. 126–133 (2007)
112. Roth, H., Schiefer, J., Schatten, A.: Probing and Monitoring of WSBPEL Processes with Web Services. In: CEC-EEE '06: Proceedings of the The 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, p. 30 (2006)
113. Rouached, M., Perrin, O., Godart, C.: Towards formal verification of web service composition. In: Business Process Management, pp. 257–273 (2006)
114. Rozinat, A., van der Aalst, W.M.P.: Conformance testing: Measuring the fit and appropriateness of event logs and process models. In: Bussler, C., Haller, A. (eds.) Business Process Management Workshops, pp. 163–176 (2006)
115. Rozinatand, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Inf. Syst. 33(1), 64–95 (2008)
116. Rud, D., Schmietendorf, A., Dumke, R.: Performance modeling of WS-BPEL-based web service compositions. scw 0, 140–147 (2006)
117. Ruth, M., Oh, S., Loup, A., Horton, B., Gallet, O., Mata, M., Tu, S.: Towards automatic regression test selection for web services. In: Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 2, pp. 729–734 (2007)
118. Ruth, M., Tu, S.: A safe regression test selection technique for Web services. In: Second International Conference on Internet and Web Applications and Services, ICIW (2007)
119. Ruth, M.E.: Concurrency in a decentralized automatic regression test selection framework for web services. In: MG '08: Proceedings of the 15th ACM Mardi Gras conference, pp. 1–8. ACM (2008)
120. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A.P.A., Casati, F.: Automated SLA Monitoring for Web Services. In: Feridun, M., Kropf, P.G., Babin, G. (eds.) DSOM 2002. LNCS, vol. 2506, pp. 28–41. Springer, Heidelberg (2002)

121. Salaün, G., Bordeaux, L., Schaerf, M.: Describing and reasoning on web services using process algebra. In: ICWS '04: Proceedings of the IEEE International Conference on Web Services, Washington, DC, USA, IEEE Computer Society (2004)
122. Schieferdecker, I., Din, G., Apostolidis, D.: Distributed functional and load tests for Web services. International Journal on Software Tools for Technology Transfer 7(4), 351–360 (2005)
123. Schlingloff, B.-H., Martens, A., Schmidt, K.: Modeling and model checking web services. In: Proceedings of the 2nd International Workshop on Logic and Communication in Multi-Agent Systems, vol. 126, Elsevier, Amsterdam (2005)
124. Sen, S., Baudry, B., Mottu, J.-M.: On combining mullti-formalism knowledge to select test models for model transformaion testing. In: ACM/IEEE International Conference on Software Testing, Lillehammer, Norway (April 2008)
125. Sharygina, N., Krning, D.: Model checking with abstraction for web services. In: Test and Analysis of Web Services, pp. 121–145 (2007)
126. Siblini, R., Mansour, N.: Testing Web services. In: ACS/IEEE 2005 International Conference on Computer Systems and Applications (AICCSA), p. 135. IEEE Computer Society (2005)
127. Sinha, A., Paradkar, A.: Model-based functional conformance testing of Web services operating on persistent data. In: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications, vol. 2006, pp. 17–22 (2006)
128. Sneed, H.M., Huang, S.: WSDLTest – A tool for testing web services. In: Proceedings of the Eighth IEEE International Symposium on Web Site Evolution (WSE'06), pp. 14–21 (2006)
129. Spanoudakis, G., Kloukinas, C., Androutsopoulos, K.: Towards security monitoring patterns. In: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), pp. 1518–1525 (2007)
130. Spanoudakis, G., Mahbub, K.: Requirements Monitoring for Service-Based Systems: Towards a framework based on Event Calculus. In: 19th IEEE International Conference on Automated Software Engineering (ASE 2004), Linz, Austria, 20-25 September 2004, pp. 379–384 (2004)
131. Tang, Y., Chen, L., He, K.T., Jing, N.: SRN: An Extended Petri-Net-Based Workflow Model for Web Service Composition. In: Proceeding of the International Conference on Web Services (ICWS), pp. 591–599 (2004)
132. Tarhini, A., Fouchal, H., Mansour, N.: A simple approach for testing web service based applications. In: Bui, A., Bui, M., Böhme, T., Unger, H. (eds.) IICS 2005. LNCS, vol. 3908, pp. 134–146. Springer, Heidelberg (2006)
133. Tierney, B., Aydt, R.A., Gunter, D., Smith, W., Taylor, V., Wolski, R., Swany, M.: A Grid Monitoring Architecture. Informational Document GFD-I.7, Global Grid Forum (January 2002)
134. Truong, H.-L., Fahringer, T.: SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid. Scientific Programming (AxGrids 2004 Special Issue) 12(4), 225–237 (2004)
135. Tsai, W., Chen, Y., Paul, R., Huang, H., Zhou, X., Wei, X.: Adaptive testing, oracle generation, and test case ranking for web services. In: 29th Annual International Computer Software and Applications Conference (COMPSAC), vol. 1, pp. 101–106 (2005)

136. Tsai, W., Wei, X., Chen, Y., Paul, R., Xiao, B.: Swiss cheese test case generation for web services testing. IEICE Transactions on Information and Systems, E88-D(12):2691–2698 (2005)
137. Tsai, W.-T., Chen, Y., Cao, Z., Bai, X., Huang, H., Paul, R.: Testing web services using progressive group testing. In: Chi, C.-H., Lam, K.-Y. (eds.) AWCC 2004. LNCS, vol. 3309, pp. 314–322. Springer, Heidelberg (2004)
138. Tsai, W.T., Paul, R., Yu, L., Saimi, A., Cao, Z.: Scenario-Based Web Services Testing with Distributed Agents. IEICE Transaction on Information and System, E86-D(10):2130–2144 (2003)
139. Tsai, W.T., Paul, R., Wang, Y., Fan, C., Wang, D.: Extending WSDL to Facilitate Web Services Testing. In: 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02), vol. 00, p. 171. IEEE Computer Society (2002)
140. Tsai, W.T., Wei, X., Chen, Y., Paul, R.: A Robust Testing Framework for Verifying Web Services by Completeness and Consistency Analysis. In: SOSE '05: Proceedings of the IEEE International Workshop, pp. 159–166. IEEE Computer Society (2005)
141. Tsai, W.T., Wei, X., Chen, Y., Xiao, B., Paul, R., Huang, H.: Developing and assuring trustworthy web services. In: ISADS 2005: Proceedings of the 7th International Symposium on Autonomous Decentralized Systems, Chengdu, China, 2005, pp. 43–50. IEEE Computer Society,
142. Tsai, W.T., Zhang, D., Paul, R., Chen, Y.: Stochastic Voting Algorithms for Web Services Group Testing. In: QSIC '05: Proceedings of the Fifth International Conference on Quality Software, pp. 99–108. IEEE Computer Society (2005)
143. Tsai, W.-T., Paul, R.A., Song, W., Cao, Z.: Coyote: An XML-Based Framework for Web Services Testing. In: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE), pp. 173–176 (2002)
144. Tsai, W.T., Bai, X., Chen, Y., Zhou, X.: Web Service Group Testing with Windowing Mechanisms. In: IEEE International Workshop on Service-Oriented System Engineering (SOSE), pp. 213–218 (2005)
145. Tsai, W.T., Chen, Y., Paul, R., Liao, N., Huang, H.: Cooperative and Group Testing in Verification of Dynamic Composite Web Services. In: Workshop on Quality Assurance and Testing of Web-Based Applications, in conjunction with COMPSAC, pp. 170–173 (2004)
146. van der Aalst, W.M.P.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: Business Process Management, pp. 161–183 (2000)
147. van der Aalst, W.M.P., de Medeiros, A.K.A.: Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. Electr. Notes Theor. Comput. Sci. 121, 3–21 (2005)
148. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.): BPM 2003. LNCS, vol. 2678. Springer, Heidelberg (2003)
149. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W(E.), Weijters, A.J.M.M.T., van der Aalst, W.M.P.: The proM framework: A new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)

150. Verbeek, H.M.W., van der Aalst, W.M.P.: Analyzing BPEL Processes using Petri Nets. In: Proceedings of the 2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management, pp. 59–78 (2005)
151. Vonk, J., Grefen, P.: Cross-organizational transaction support for E-services in virtual enterprises. Distrib. Parallel. Dat. 14, 137–172 (2003)
152. Wang, Y., Bai, X., Li, J., Huang, R.: Ontology-based test case generation for testing web services. In: Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems, pp. 43–50 (2007)
153. Wombacher, A., Fankhauser, P., Neuhold, E.: Transforming BPEL into annotated deterministic finite state automata for service discovery. In: ICWS '04: Proceedings of the IEEE International Conference on Web Services, Washington, DC, USA, p. 316. IEEE Computer Society (2004)
154. Xu, W., Offutt, J., Luo, J.: Testing Web services by XML perturbation. In: Proceedings. 16th IEEE International Symposium on Software Reliability Engineering, p. 10 (2006)
155. Yi, X., Kochut, K.J.: A CP-nets-based Design and Verification Framework for Web Services Composition. In: Proceeding of the International Conference on Web Services (ICWS), p. 756 (2004)
156. Zannier, C., Melnik, G., Maurer, F.: On the success of empirical studies in the international conference on software engineering. In: ICSE '06: Proceedings of the 28th international conference on Software engineering, New York, NY, USA, pp. 341–350. ACM (2006)

8

Service Engineering

Vasilios Andrikopoulos[1], Antonio Bucchiarone[2], Elisabetta Di Nitto[3], Raman Kazhamiakin[2], Stephen Lane[4], Valentina Mazza[3], and Ita Richardson[4]

[1] Tilburg University, The Netherlands
[2] Fondazione Bruno Kessler (FBK), Trento, Italy
[3] Politecnico di Milano, Italy
[4] Lero — the Irish Software Engineering Research Centre, Ireland

Chapter Overview. Service Engineering and Design (SED) aims at establishing, understanding and managing the entire service lifecycle, including identifying, finding, designing, developing, deploying, evolving, quality assuring, and maintaining services. SED principles, techniques and methods interweave and exploit the mechanisms provided by the S-Cube technology stack with the aim of developing high-quality service-based systems. For example, the SED plane provides specifications to the BPM and SAM layers that can guide the service composition and coordination layer in composing services in a manner that guarantees that the composition behaves as expected.

This chapter focuses on the analysis of existing life cycle approaches for adaptable and evolvable service-based applications with an emphasis on how the lack of a life cycle that can handle adaptation lead to the definition of a reference service life cycle for the development of adaptable service based applications. This chapter also identifies the main concepts, issues, and challenges concerning the various phases of our reference life cycle as they have been identified in the literature.

8.1 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 history of software engineering shows a progressive departure from the strict boundaries of the closed-world assumption [14] toward more flexibility to support continuous evolution. Methods, techniques, and tools were developed to support the need for change without compromising product quality and cost-efficient developments. 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

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 271–337, 2010.

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 [34].

In this chapter the focus is on the analysis of the existing life cycles for adaptable and evolvable service-based applications. In particular, the lack of a life cycle taking explicitly adaptation into account led to the definition of a reference service life cycle for the development of adaptable service based applications. Moreover through the chapter are identified the main concepts, issues, and challenges concerning the various phases of our reference life cycle as they have been identified in the literature. Finally are analyzed 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.

Consistently with its objectives, the chapter is structured as follows. Section 8.2 introduces some basic definitions relevant for the content of the chapter. Section 8.3 presents the life cycles for service-based applications and details the various activities in the life cycle. Section 8.4 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 8.5 draws the identified gaps and the Section 8.6 draws the conclusions.

8.2 Preliminary Definitions

The goal of this section is to give a short overview of the main basic concepts that are relevant for the content of the chapter. 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 and the relationships among them will be described using UML (Unified Modeling Language) diagrams. The UML was intended as a 'language that is used to specify, visualize, construct and document the artifacts of a software system' (see [94, p. 3]) and not as a language for describing such concepts. In the diagrams reported in the following sections, some of the UML constructs are used to identify the main concepts of the conceptual model and the relationships among them. Such diagrams are an extension and clarification of some of those belonging to the conceptual model proposed in the SeCSE project [2]. As an extension of such model, moreover, we have introduced the concepts of Service-Based Application and Adaptable Service-

Based Application showing the relationships between them and the other concepts.

8.2.1 Agents and Actors

The model of Figure 8.1 [2] 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.

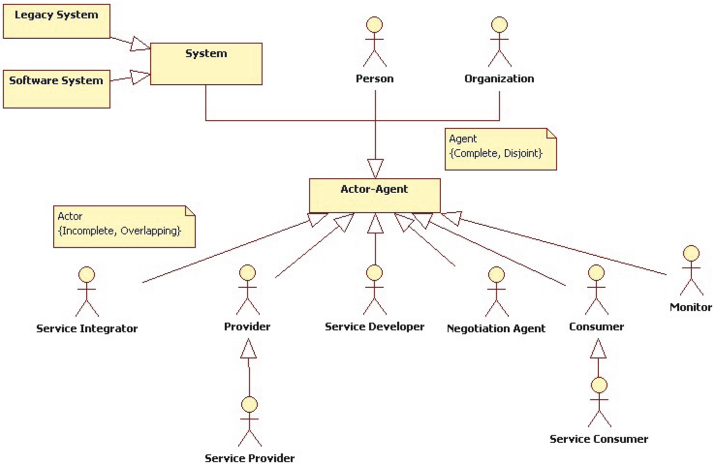


Fig. 8.1. The Agent-Actor Diagram

8.2.2 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. 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 parameters are formed and used for further calls. This orchestration is also called Service Process. The Service Integrator (see Figure 8.1) 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 8.2 shows the defined terms and highlights the relationships among them.

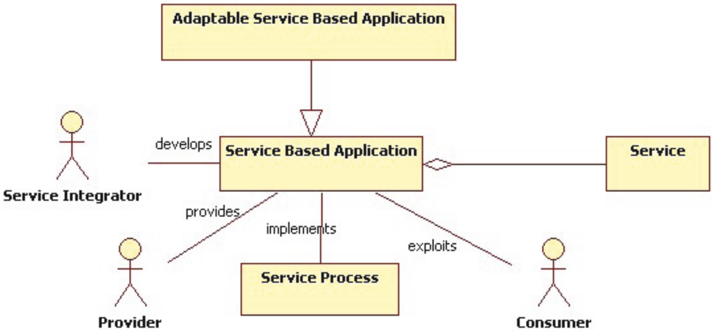


Fig. 8.2. The Service Based Application Diagram

8.2.3 Types of Services

Services exploited in a service-based application can be offered by various different agents, as highlighted in Section 8.2.1 (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 8.3. 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.

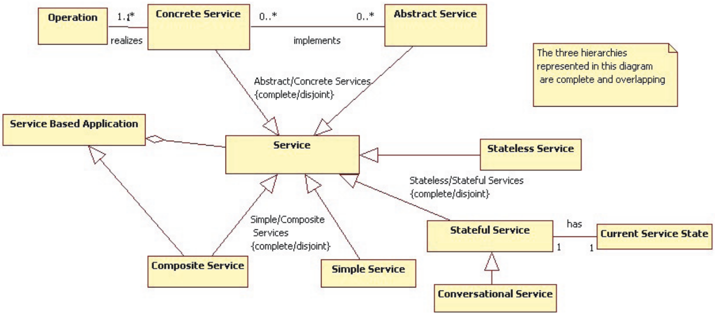

Operation
1..*
realizes
Concrete Service
0..*
0..*
Abstract Service
implements
The three hierarchies represented in this diagram are complete and overlapping
Abstract/Concrete Services {complete/disjoint}
Service Based Application
Service
Stateless Service
Stateless/Stateful Services {complete/disjoint}
Simple/Composite Services {complete/disjoint}
Stateful Service
has
1
1
Current Service State
Composite Service
Simple Service
Conversational Service

Fig. 8.3. The Service Type Diagram

8.3 Engineering Service Based Application

The purpose of this section is the analysis of the state of the art of the service life cycles that can be found in literature. Moreover a reference life cycle is proposed in order to provide a service life cycle taking explicitly adaptation into account. Then, each phase of the life cycle is analysed.

8.3.1 SBA Life Cycles

Existing Life Cycles

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, to the application context, and the capability to enact the corresponding adaptation strategy. In this section, a description of the existing SBA life-cycle methodologies is reported.

The Web Services Development Lifecycle (SLDC)

The Web Services Development Lifecycle (SLDC) [85] is a continuous and iterative approach to the development, implementation, deployment and maintenance of software services.

Nevertheless, [86] 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 [121], 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 [26, 43].

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 is a service oriented analysis and design (SOAD) process for modeling, analysing, designing and producing a SOA that aligns with business goals [12]. 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. 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 [13].

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 [121]. While SOAD process and notation have yet to be defined in detail [121], 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.

ASTRO

ASTRO is a methodology that is focused primarily on service composition or service orchestration. It covers all the life cycle of service-based systems, 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 [36], covering all the phases of service development and maintenance lifecycle. 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 8.4.

The Reference Service Life Cycle

As seen in the descriptions before, almost all life-cycles in the literature do not address explicitly the possibility of a service-based application to be adaptable. There is a need for introducing a life-cycle for SBAs that takes adaptation into explicit account. The life-cycle (proposed in S-Cube network of Excellence in order to address this issue) shown in Figure 8.4 aims at filling this gap. 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 the right hand side 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. The two cycles are not conflicting with each other; instead, they cohexist and support each other during the lifetime of the application. We say, in particular, that design time activities allow for evolution of the application, that is, for the introduction of permanent and, usually, important changes, while the runtime activities allow for temporary adaptation of the application to the specific circumstances that are occurring at a certain time.

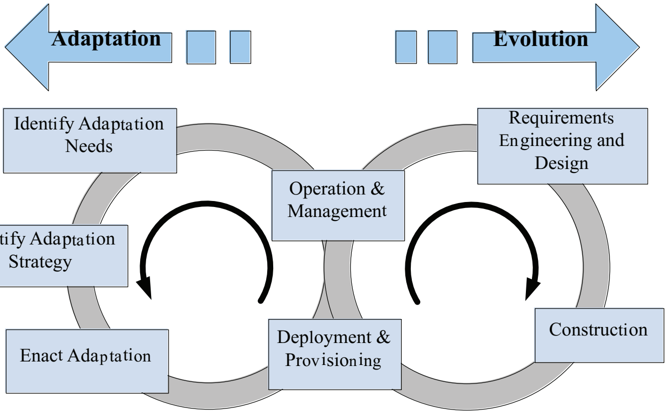

Adaptation
Evolution
Identify Adaptation Needs
Identify Adaptation Strategy
Enact Adaptation
Operation & Management
Deployment & Provisioning
Requirements Engineering and Design
Construction

Fig. 8.4. The Lifecycle for Evolvable and Adaptable Service-Based Applications

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

At the *requirements engineering and design* phase the adaptation and monitoring requirements are used to perform the design for adaptation and monitoring. During SBA *construction*, together with the construction of the SBA, the corresponding monitors and the adaptation mechanisms are being realized. The *deployment* phase also involves the activities related to adaptation and monitoring: deployment of the adaptation and monitoring mechanisms and deployment time adaptation actions (e.g., binding). During the *operation and management* phase, the run-time monitoring is executed, using some designed properties, and help the SBA to detect relevant context and system changes. After this phase the left-side of the life-cycle is executed. Here, we can proceed in two different directions; executing evolution or adaptation of the SBA. In the first case we re-start the right-side of the cycle with the requirements engineering and design phase while in the second case we proceed identifying adaptation needs that can be triggered from monitored events, adaptation requirements or context conditions. For each adaptation need it is possible to define a set of suitable strategies. Each adaptation strategy can be characterized by its complexity and its functional and non functional properties. The identification of the most suitable strategy is supported by a reasoner that also bases its decisions on multiple criteria extracted from the current situation and from the knowledge obtained from previous adaptations and executions. After this selection, the enactment of the adaptation strategy is performed. The execution of all activities and phases in all runtime phases may be performed autonomously by SBAs or may involve active participation of the various human actors.

As said, 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, 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 is to characterize the context of service-based applications and 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 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 of a hard-coded logic.

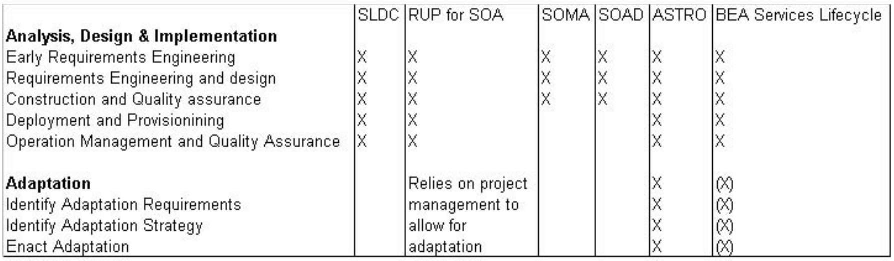
As humans have a very important role in the open world either as users of service-based systems or as service providers themselves, the aspects concerned with the so called Human-Computing Interaction (HCI) have to be considered with particular attention. In particular, it's important to select and codify 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.

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.

Comparison

Figure 8.5 shows how each of the discussed lifecycles compare to the reference lifecycle introduced in Figure 8.4. 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 reference life cycle of Figure 8.4 is based on SLDC; 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. The following sections investigate each phase in more depth.



	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 management to allow for adaptation			X	(X)
Identify Adaptation Requirements					X	(X)
Identify Adaptation Strategy					X	(X)
Enact Adaptation					X	(X)

Fig. 8.5. Lifecycles vs. Reference Lifecycle

8.3.2 Life Cycle Phases

Requirement Engineering and Design

In this section we address the phase of Requirement Engineering and Design. In particular will be explained, first, the phase of Requirement Engineering, and then the phase of Design focusing on adaptation and monitoring.

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 [102]:

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. Such activities are not always executed in a fixed sequence: if needed some activity can be re-executed. This is mainly due to the fact that 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.

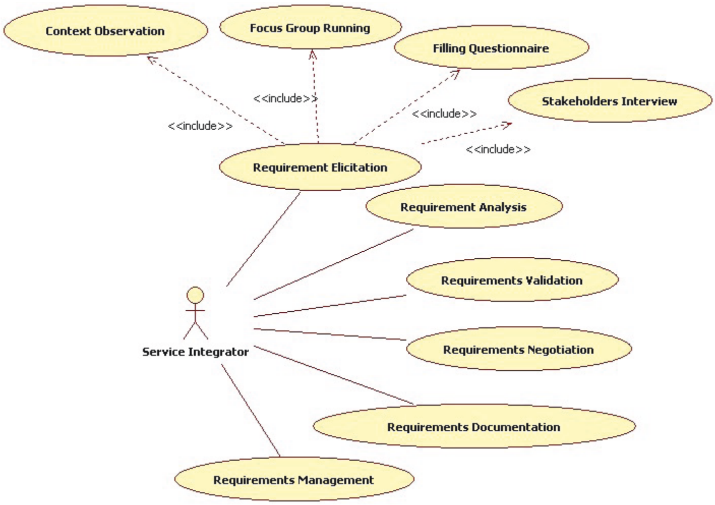


Fig. 8.6. Requirement Engineering Process

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 [32] 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. The actions of the process are sketched in Figure 8.6. The action 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 run-

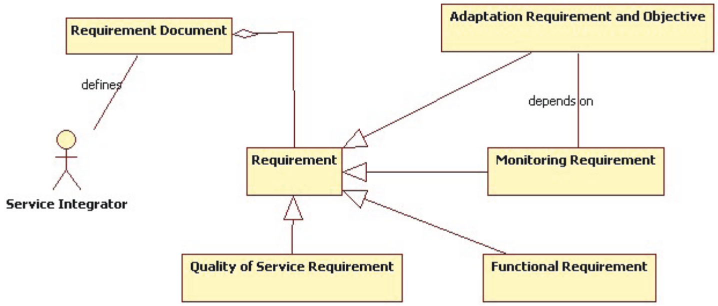


Fig. 8.7. Requirement Engineering Diagram

ning, interviews to the stakeholders, 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 action 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 action (*Requirement Negotiation*) is required to obtain a set of consistent requirements then formalized in a *Requirement Documentation*. Obviously the a continuous management requirement (*Requirement Management*) is performed to manage changes. Figure 8.7 highlights that the main outcome of the RE process is a requirements document that defines what is to be implemented. The association between the Service Integrator and the Requirement Document implies the actions shown in the diagram of Figure 8.6. The Requirement Document contains a formalization of the collected Requirements. A Requirement could have different nature: it could be a Quality of Service Requirement or a Functional Requirement. When adaptation is needed, Adaptation Requirements must be defined and consequently Monitoring Requirements.

Boehm [21] 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.

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 [108].

Maiden [74] 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. [22] 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 [71] 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. [68] 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. [44] 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.

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 [48], data and process mining for monitoring purposes, and service discovery, binding and automated composition [92] 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 [8]. 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 [47].
- *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 [16]: the underlying BPEL process is augmented with the calls to the a special proxy that evaluates the monitored properties. In [118] analogous code modification is applied in order to inject the necessary adaptation actions. On the other side, monitoring approaches presented in, e.g., [92, 72], 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

The presented design for adaptation and monitoring concepts are represented in Figure 8.8. 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).

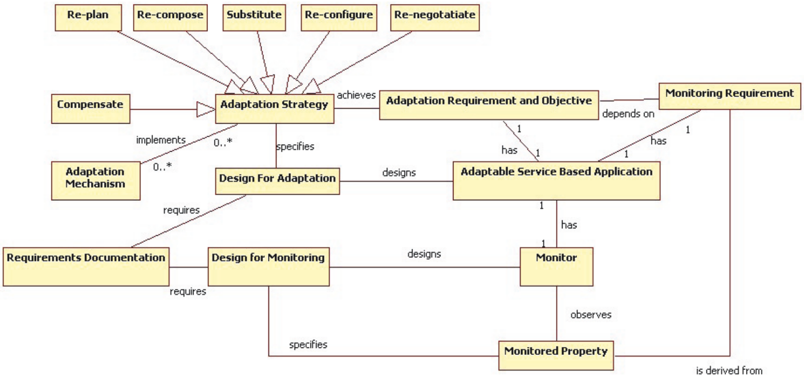


Fig. 8.8. The Design for Adaptation Diagram

Below, some aspects relevant for the specification of the adaptation strategies and monitoring properties are presented.

Adaptation and Monitoring Specification

As 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 [92], WS-CoL [16]). 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 [47], declarative functional goal specification [63], 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 [118, 15].
- 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 [93].

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 shortly introduced in the following:

- *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.

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 8.9). 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 literature a variety of languages for the construction of SBA are presented. Among them Business Process Execution Language (BPEL for short, [83]) 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 [87], jPDL [1], 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 [80] or WS-CDL [115] 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., [91, 90]).

Another important activity that should be accomplished during the construction phase as well as in other phases of the development and operation process

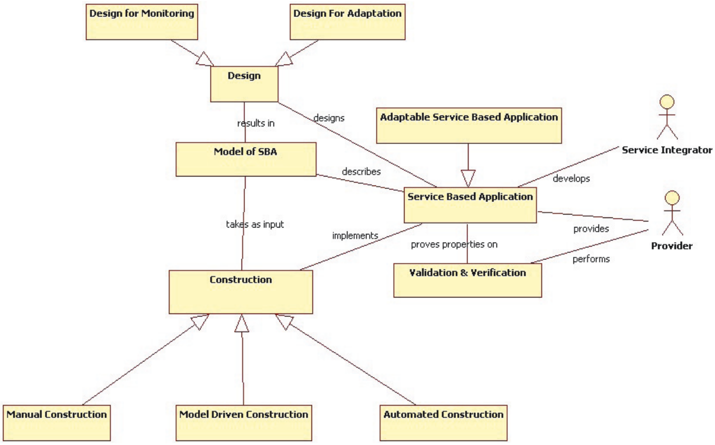


Fig. 8.9. The Construction Diagram

is the *verification and validation* of the SBA against various requirements and constraints. Section 8.3.2 provides a summary of this issue.

Deployment and Provisioning

The phase "Deployment and Provisioning" of the lifecycle in Figure 8.4 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

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) [30]: 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, is not 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 [105] 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) [40] 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 [6] and BPML [9] /WSCI [10]: 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 8.10 [2] 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.

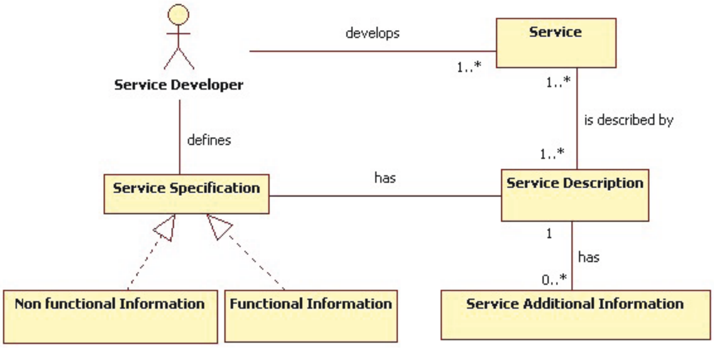


Fig. 8.10. The SeCSE Service Description Diagram

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) [3] repositories, which act 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) [4]. 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 [114].

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 [31]. 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. [37]. 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 [60]. The METEOR-S project [113] 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 [89], [101]).

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" category); where to deploy these services/components, a strategy for deployment, and an infrastructure for executing the deployment strategy. Tawlar et al. [106] 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. [11] 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 8.11.

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.

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

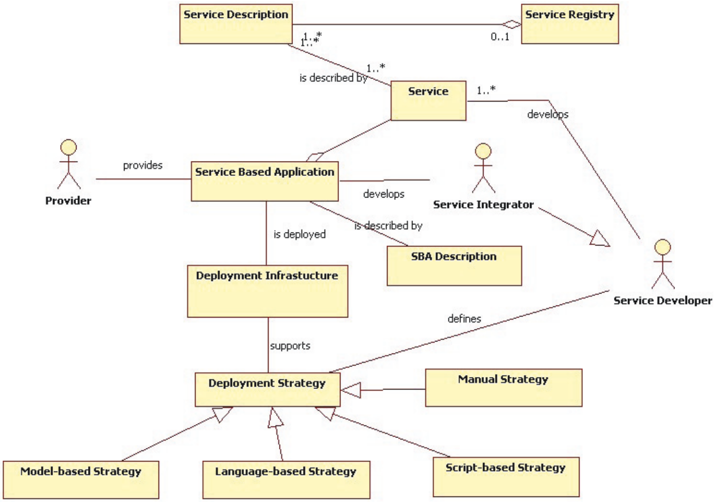


Fig. 8.11. The Service Publication/Deployment Diagram

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 [85].

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 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 [5] 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. Service operation requires a service governance (see section 8.3.2) ensuring that the architecture is operating as expected maintaining a certain QoS level (Figure 8.12). Of particular interest for the service

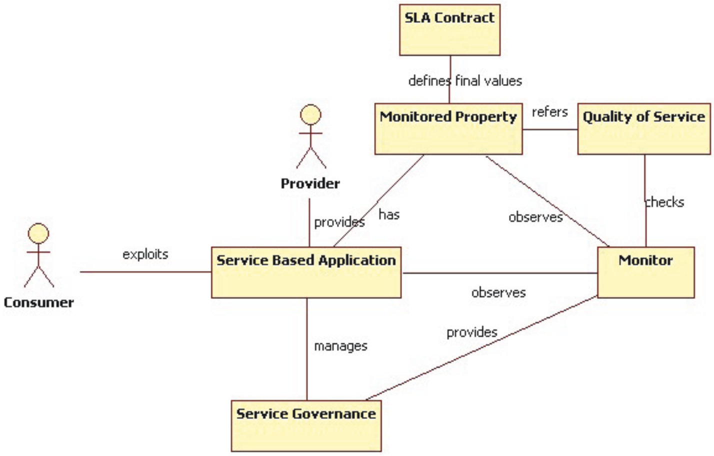


Fig. 8.12. The Operation and Management Diagram

operation is the service fault, since the identification of service faults permits the triggering of adaptation mechanisms needed to adapt SBAs.

Fault Detection

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. [81] 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 and faults [38] 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 8.13 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

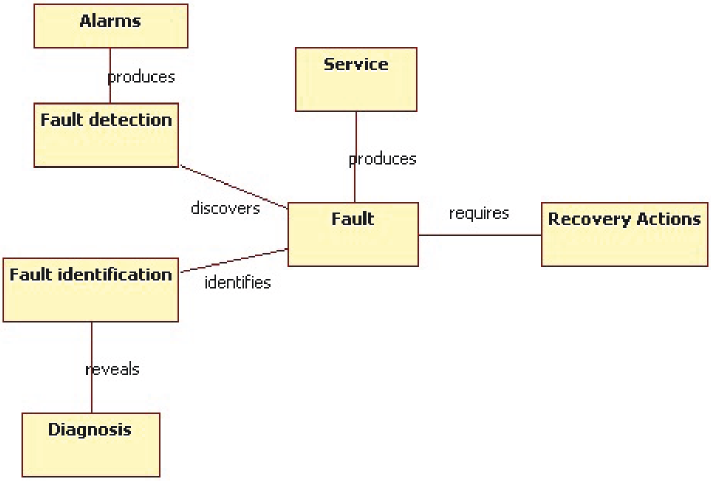


Fig. 8.13. The Service Fault Diagram

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. [35] 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 [97].

Since the nature of a fault depends on a lot of causes, some authors [7] 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, 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 [14]. 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.

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 8.14).

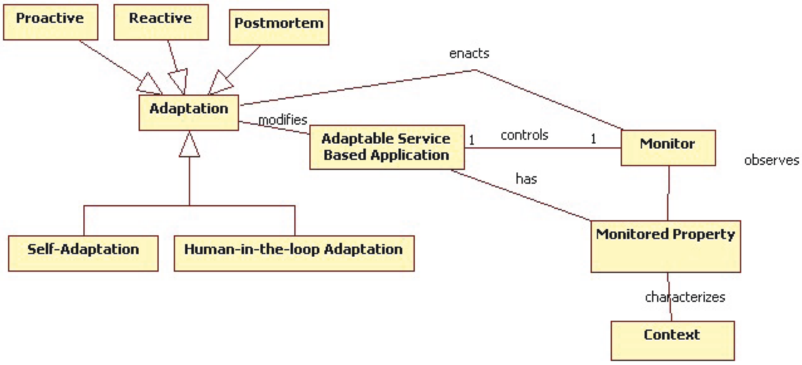


Fig. 8.14. The Adaptation Diagram

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 8.4. This general definition becomes more concrete when we consider different forms of adaptation (see Figure 8.14): *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 8.4, the adaptation cycle consists of the following principle steps:

- 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 8.3.2 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.

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 [25, 17], tools supporting customization of the process models [59], 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.

Cross-Cutting Concerns

The previous sections presented methodologies and processes for each the phases of the service life cycle of Figure 8.4. 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.

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 it is useful to describe the meaning of *IT governance* as SOA governance stems from and is deeply rooted in IT governance [20]. 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 [56].

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 [77].

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 [75]. 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 [85]. Two different governance models are possible [85]:

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

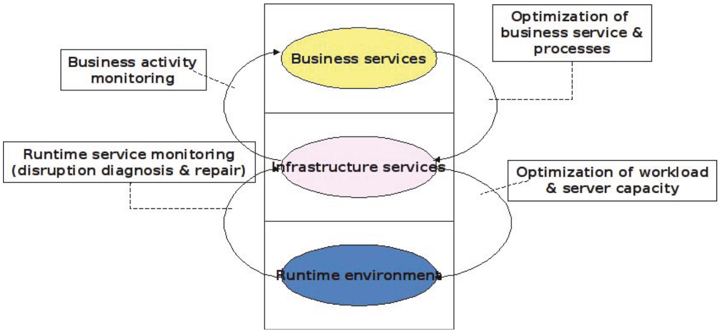

Business activity monitoring
Business services
Optimization of business service & processes
Runtime service monitoring (disruption diagnosis & repair)
Infrastructure services
Optimization of workload & server capacity
Runtime environment

Fig. 8.15. Developing and Managing SBAs

existing service infrastructure within any business unit. Federated governance suits enterprise chains better.

Figure 8.15 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 8.16).

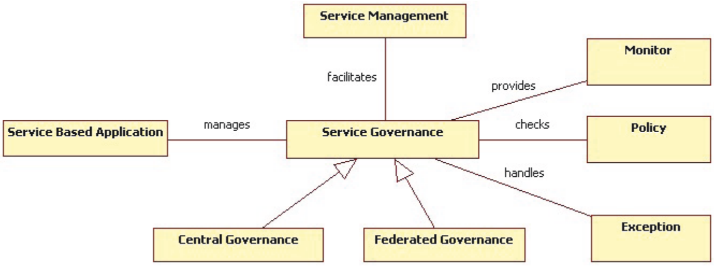


Fig. 8.16. The Governance of Service Based Applications Diagram

Quality Assurance (QA) of SBAs

In this section the focus is on the major points of QA for SBA throughout all the phases of the life-cycle (see the section 7 for further details).

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

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." [84] In a broader sense, static analysis can be extended to documents at all stages of the software life cycle.
2. Testing: "... testing entails executing a program and examining the results produced." [84] 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... " [33] 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.

Figure 8.17 summarizes the QA 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.

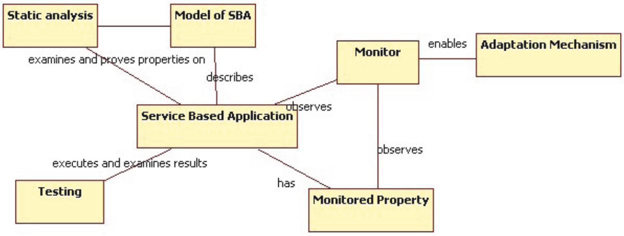


Fig. 8.17. The QA Diagram

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) [3] 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 [30], 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.[100] proposed an ontology-based approach to capture real world knowledge for a finer granularity annotation of Web services. Moreover [70] 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 [58]. Figure 8.18 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 8.4 : 1) when the requirements for a new system are gathered (*Early Discovery*) (Requirement Engineering activity in Requirement Engineering and Design Phase in figure 8.4), 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 8.4), 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 8.4). The latter type of discovery is required during adaptation enactment (see section 8.3.2). Some researches attempt to optimize the runtime service discovery process [103], using the information gathered during design time service discovery as a sort of cache.

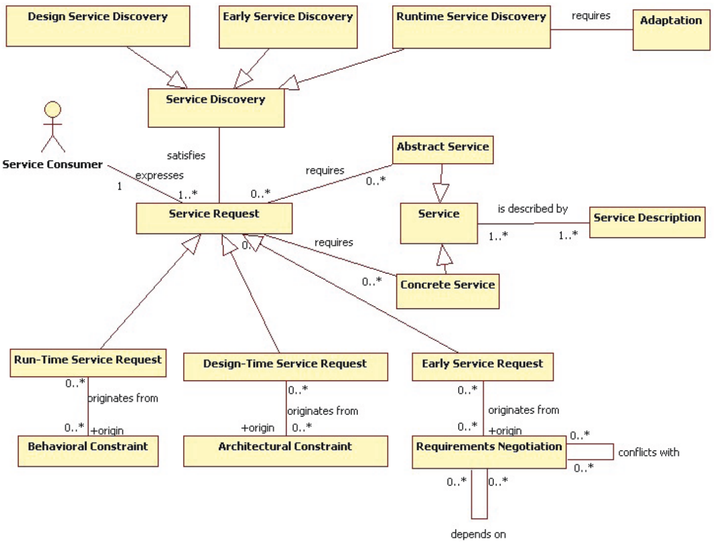


Fig. 8.18. The Service Discovery Diagram

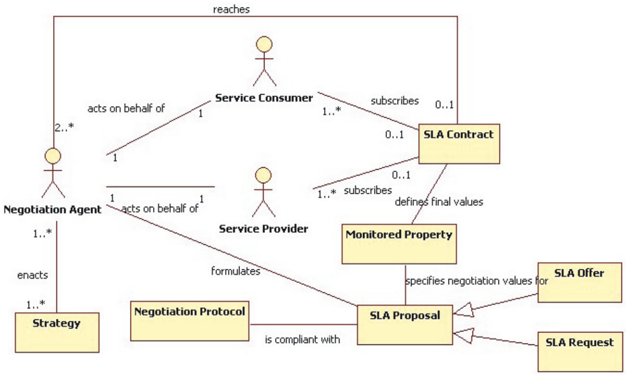


Fig. 8.19. The Service Level Agreement Negotiation Diagram

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. Figure 8.19 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.

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

8.4 Software Engineering Practices Relevant to Service-Based Applications

This section covers “classical” design and development methodologies and issues, some of which have been established as industry-wide accepted standards in the last decades. More specifically, Section 8.4.1 presents established methods and theories for software process quality and assurance. Furthermore, Section 8.4.1 discusses the predecessor of service orientation, i.e., the component-based paradigm; Section 8.4.1 provides some insight into the issue of legacy system re-engineering and how it affects SBA engineering. Consequently, Section 8.4.1 discusses the issues of software maintenance and evolution and how they are related to SBA adaptation. Finally, Section 8.4.2 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.

8.4.1 Classical Software Engineering

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 [49] defines a software process as “the set of tools, methods and practices we use to produce a software product”. Paulk et al. [78] 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 8.20. 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.

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

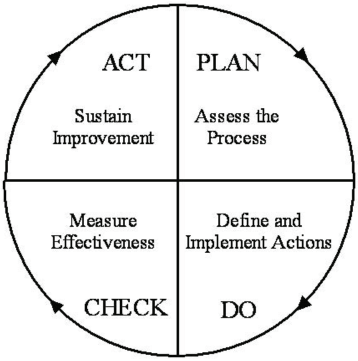


Fig. 8.20. Plan-Do-Check-Act for Software Process

(CMMI^{TM}) [82, 107]. 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 [107].

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 [79, 52, 57] there is also evidence that there have been increases in productivity and efficiency due to the implementation of planned processes [19, 104, 24, 50, 41].

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 [98]. 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. [65] 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 [62]. Despite the fact that they are "simple" and "quick" [55], 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.

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 left-hand side of Figure 8.4 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 8.3.1 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$^{\mathrm{TM}}$(CMMI$^{\mathrm{TM}}$[107]) as it is an accepted exemplar containing process areas relevant to the development of systems and software. The CMMI$^{\mathrm{TM}}$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 [42]. [76] 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^{TM}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.

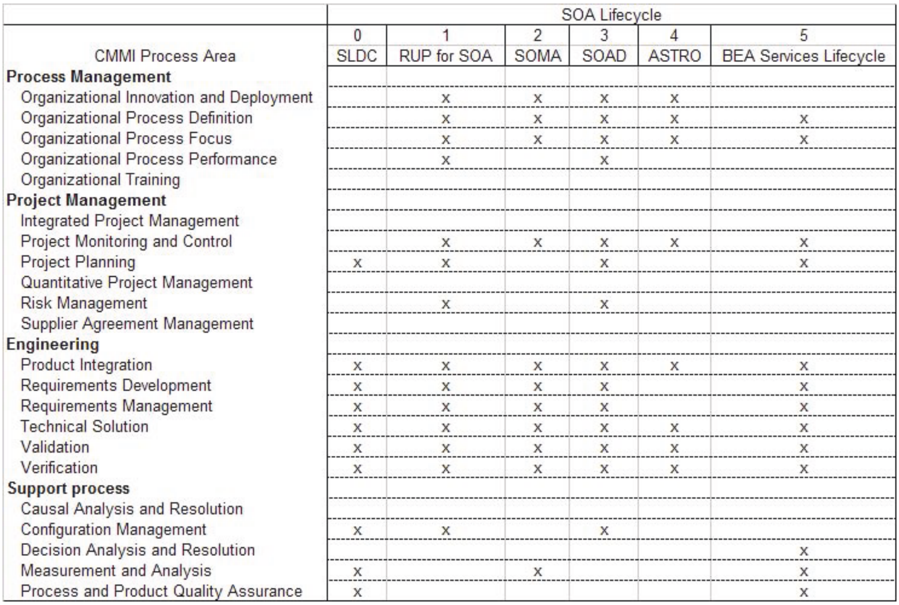
Comparing Lifecycles and CMMI^{TM}

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 8.21 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^{TM}into services development does not take into account the left-hand side of the reference lifecycle shown in Figure 8.4. Software process models will need to be adjusted to cope with the adaptation phases of the reference life-cycle.

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 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



	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

Fig. 8.21. SBA Lifecycles vs. CMMI

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. [21] 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. [64] 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 [73], 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. [27] 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. [61] 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 [88].

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 ([23], [117], [116], [109]): 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 ([116], [117]): 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 [110], or black-box modernization [117]. 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 [66]. 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 [99].

  To implement the encapsulation and integration strategy it suffices to recreate 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 [28].

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 [45]. 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 [120]. Further examples are [39, 51]. 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 [96]. 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 [67].

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 ([112], [111]), 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.

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 [69] categorized maintenance activities into four classes (the classification is in the Standard IEEE 610.12[38]):

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' [18], 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 [119] 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. [29] illustrates the use of multi-dimensional libraries to manage the versions; moreover to track the dependencies 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. [53] 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. [53] 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 [86], 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.

8.4.2 Business Process Methodologies

A business process methodology (please refer to section 2 for further details) 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 [95], 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.

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 organization's 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.

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 [46]. 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 [54]:

- 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 [46]. 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 [46]. These are shown in Figure 8.22. 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 8.22 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.

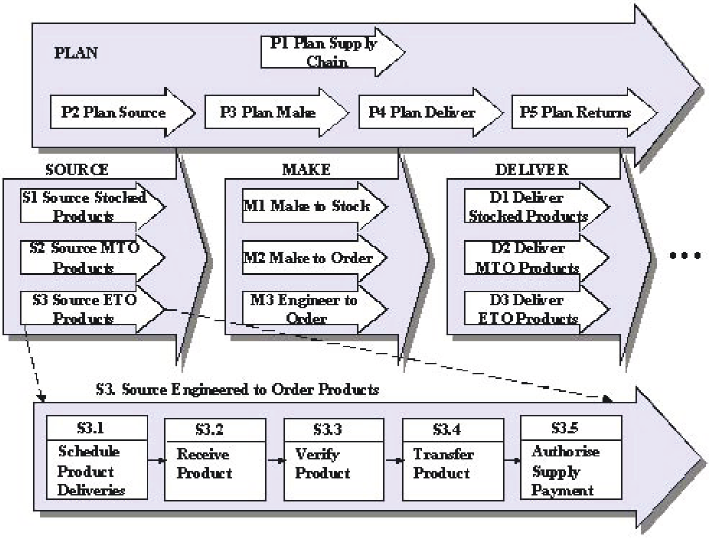


Fig. 8.22. Levels of SCOR Processes (Source: [46])

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.

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.

Evaluation of DMAIC and SCOR

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.

8.5 Gaps

This chapter wants to offer an overview of the issues behind the service engineering focusing on the aspects inherited by the classical software engineering.

In summary, section 8.3 classified Service Oriented Computing methods, techniques and tools according to a proposed service lifecycle model (Figure 8.4). 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. Moreover, the various SBA life-cycles are, in summary, discussed. 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 8.4. 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.

Section 8.4 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. 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 it is recognized 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.

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 [85], 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.

8.6 Conclusion

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. 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. 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.

References

1. jBPM Process Definition Language (JPDL), http://docs.jboss.org/jbpm
2. SeCSE Project, http://www.secse-project.eu/
3. Uddi white papers, http://www.uddi.org/whitepapers.html
4. UNSPSC. http://www.unspsc.org/
5. Web Services Distributed Management (WSDM)
6. Business process execution language for web services (2003), ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf
7. Ardagna, D., Cappiello, C., Fugini, M.G., Mussi, E., Pernici, B., Plebani, P.: Faults and recovery actions for self-healing web services, 2006. In: www 2006, Edinburg, UK (2006)
8. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: Fault tolerant web service orchestration by means of diagnosis. In: Gruhn, V., Oquendo, F. (eds.) EWSA 2006. LNCS, vol. 4344, pp. 2–16. Springer, Heidelberg (2006)
9. Arkin, A.: Business Process Modeling Language (BPML) (November 2002)
10. Arkin, A.: Web Service Choreography Interface (WSCI) 1.0 (Aug. 2002)
11. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totok, A.A.: Pattern based SOA deployment. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 1–12. Springer, Heidelberg (2007)

12. Arsanjani, A.: Service-oriented modeling and architecture (November 2004)
13. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: SOMA: A method for developing service-oriented solutions. IBM Systems Journal 47(3) (2008)
14. Baresi, L., Di Nitto, E., Ghezzi, C.: Toward open-world software: Issue and challenges. Computer 39(10), 36–43 (2006)
15. Baresi, L., Guinea, S., Pasquale, L.: Self-healing BPEL processes with Dynamo and the JBoss rule engine. In: ESSPE '07: International workshop on Engineering of software services for pervasive environments, pp. 11–20 (2007)
16. Baresi, L., Guinea, S.: Towards dynamic monitoring of WS-BPEL processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 269–282. Springer, Heidelberg (2005)
17. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing Adapters for Web Services Integration. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 415–429. Springer, Heidelberg (2005)
18. Bennett, K.H., Rajlich, V.T.: Software maintenance and evolution: a roadmap. In: Conference on The Future of Software Engineering, New York, NY, USA, pp. 73–87. ACM (2000)
19. Bergman, B., Klefsjo, B.: Quality From Customer Needs to Customer Satisfaction. Studentlitteratur (1994)
20. Bieberstein, N., et al.: Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap. IBM Press (2006)
21. Boehm, B., Abts, C.: Cots integration: Plug and pray? Computer 32(1), 135–138 (1999)
22. Bohmann, T., Junginger, M., Krcmar, H.: Modular service architectures: a concept and method for engineering it services. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, Jan. 2003, p. 10 (2003)
23. Brodie, M.L., Stonebraker, M.: Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach. Morgan Kaufmann, San Francisco (1995)
24. Brodman, J.G., Johnson, D.L.: A software process improvement approach tailored for small organisations and small projects. In: 19th International Conference on Software Engineering, Boston, Massachusetts, USA (1997)
25. Brogi, A., Popescu, R.: Automated Generation of BPEL Adapters. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 27–39. Springer, Heidelberg (2006)
26. Brown, A.W., Johnston, S.K., Larsen, G., Palistrant, J.: SOA Development Using the IBM Rational Software Development Platform: A Practical Guide (2005)
27. Cai, X., Lyu, M.R., Wong, K.f., Ko, R.: Component-based software engineering: Technologies, development frameworks, and quality assurance schemes. In: Lecture Notes, pp. 372–379. IEEE Computer Society (2000)
28. Canfora, G., Fasolino, A.R., Frattolillo, G., Tramontana, P.: A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. J. Syst. Softw. 81(4), 463–480 (2008)
29. Casanova, M., Van Der Straeten, R., Jonckers, V.: 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, p. 123. IEEE Computer Society (2003)

30. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C (2001), http://www.w3.org/TR/wsdl
31. Clark, J., Casanave, C., Kanaskie, K., Harvey, B., Smith, N., Yunker, J., Riemer, K.: ebXML Business Process Specification Schema Version 1.01. Technical report, UN/CEFACT and OASIS (May 2001), http://www.ebxml.org/specs/ebBPSS.pdf
32. Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., Theofanos, M.: Identifying and measuring quality in a software requirements specification. In: First International Software Metrics Symposium, Proceedings, May 1993, pp. 141–152 (1993)
33. Delgado, N., Gates, A.Q., Roach, S.: A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans. Software Eng. 30(12), 859–872 (2004)
34. Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K.: A journey to highly dynamic, self-adaptive service-based applications. Automated Software Engineering 15(3-4), 313–341 (2008)
35. Dialani, V., Miles, S., Moreau, L., De Roure, D.C., Luck, M.: Transparent fault tolerance for web services based architectures. In: Monien, B., Feldmann, R.L. (eds.) Euro-Par 2002. LNCS, vol. 2400, pp. 889–898. Springer, Heidelberg (2002)
36. Durvasula, S., et al.: SOA Practitioner's Guide. Published: BEA Systems (2007)
37. Dustdar, S., Treiber, M.: A view based analysis on web service registries. Distributed and Parallel Databases 18(2), 147–171 (2005)
38. Institute O. Electrical and Electronics E. (ieee). IEEE 90: IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society (1990)
39. Erradi, A., Maheshwari, P., Tosic, V.: Policy-driven middleware for self-adaptation of web services compositions. In: van Steen, M., Henning, M. (eds.) Middleware 2006. LNCS, vol. 4290, pp. 62–80. Springer, Heidelberg (2006)
40. Fensel, D., Bussler, C.: The web service modeling framework wsmf (2002)
41. Galin, D., Avrahami, M.: Are cmm program investments beneficial? analyzing past studies. IEEE Software, 81–87 (2006)
42. Gallagher, B., Brownsword, L.: The rational unified process and the capability maturity model – integrated systems/software engineering. In: RUP/CMMI Tutorial – ESEPG (2001)
43. Ganci, J., Acharya, A., Adams, J., Diaz de Eusebio, P., Rahi, G., Strachan, D., Utsumi, K., Washio, N.: Patterns: SOA Foundation Service Creation Scenario. IBM Redbooks (2006)
44. Goldsby, H.J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Hughes, D.: Goal-based modeling of dynamically adaptive system requirements. In: 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2008. ECBS 2008. 31 2008-April 4 2008, pp. 36–45 (2008)
45. Graaf, B., Weber, S., van Deursen, A.: Model-driven migration of supervisory machine control architectures. J. Syst. Softw. 81(4), 517–535 (2008)
46. Harmon, P.: Second generation business process methodologies. Business Process Trends: Newsletter 1(5) (2003)
47. Harney, J., Doshi, P.: Speeding up adaptation of web service compositions using expiration times. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, pp. 1023–1032 (2007)

48. Hielscher, J., Kazhamiakin, R., Metzger, A., Pistore, M.: A framework for proactive self-adaptation of service-based applications based on online testing. In: Mähönen, P., Pohl, K., Priol, T. (eds.) ServiceWave 2008. LNCS, vol. 5377, pp. 122–133. Springer, Heidelberg (2008)
49. Humphrey, W.S.: Managing the Software Process. Addison-Wesley, Reading (1989)
50. Humphrey, W.S.: Three dimensions of process improvement, part i: Process maturity. CROSSTALK The Journal of Defense Software Engineering (1998)
51. Hutchinson, J., Kotonya, G., Walkerdine, J., Sawyer, P., Dobson, G., Onditi, V.: Evolving existing systems to service-oriented architectures: Perspective and challenges. In: ICWS, pp. 896–903. IEEE Computer Society (2007)
52. Jones, C.: Patterns of Software Systems Failure and Success. International Thompson Computer Press (1996)
53. Kajko-Mattsson, M., Tepczynski, M.: 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, pp. 665–668. IEEE Computer Society (2005)
54. Kasi, V.: 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)
55. Kent, B.: Extreme Programming Explained: Embrace Change. Adison-Wesley, Reading (2000)
56. Graham, S., Holley, K., Palistrant, J.: Effective SOA Governance (2006)
57. Kolind, J.P., Wastell, D.G.: The sei's capability maturity model: a critical survey of adoption experiences in a cross-section of typical uk companies. In: McMaster, T., Mumford, E., Swanson, E.B., Warboys, B., Wastell, D. (eds.) IFIP TC8 WG8.6 International Working Conference on Diffusion, Adoption and Implementation of Information Technology, Ambleside, Cumbria, U.K, pp. 305–319 (1997)
58. Küster, U., König-Ries, B., Stern, M., Klein, M.: 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, pp. 1033–1042. ACM (2007)
59. Lambers, L., Ehrig, H., Mariani, L., Pezze, M.: Iterative Model-Driven Development of Adaptable Service-Based Applications. In: ASE '07, pp. 453–456 (2007)
60. Lara, R., Corella, M., Castells, P.: A flexible model for web service discovery. In: 1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives, Seoul, Korea (September 2006)
61. Lau, K.-K., Wang, Z.: A taxonomy of software component models. In: EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, Washington, DC, USA, pp. 88–95. IEEE Computer Society (2005)
62. Law, A., Learn, S.: Waltzing with Changes. In: Proceedings of the Agile Development Conference, pp. 279–288. IEEE Computer Society Press, Washington, DC (2005)
63. Lazovik, A., Aiello, M., Papazoglou, M.P.: Associating Sssertions with Business Processes and Monitoring their Execution. In: Service-Oriented Computing - ICSOC 2004, Second International Conference, pp. 94–104 (2004)

64. Lee, J., Kim, J., Shin, G.-S.: Facilitating reuse of software components using repository technology. In: APSEC '03: Proceedings of the Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference, p. 136, Washington, DC, USA, IEEE Computer Society (2003)
65. Leszak, M., Perry, D.E., Stoll, D.: A Case Study in Root Cause Defect Analysis. In: Proceedings of the 22 nd international conference on Software engineering, vol. 4, pp. 428–437 (2000)
66. Levey, R.: Reengineering COBOL with Objects: Step by Step to Sustainable Legacy Systems. McGraw-Hill, New York (1996)
67. Lewis, G., Morris, E., Smith, D., O'Brien, L.: Service-oriented migration and reuse technique (smart). In: STEP '05: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice, pp. 222–229, Washington, DC, USA, IEEE Computer Society (2005)
68. Lichtenstein, S., Nguyen, L., Hunter, A.: Issues in it service-oriented requirements engineering. Australasian Journal of Information Systems 13(1) (2007)
69. Lientz, B.P., Swanson, E.B.: Software Maintenance Management. Addison-Wesley Longman Publishing Co., Inc., Boston (1980)
70. Ma, J., Cao, J., Zhang, Y.: A probabilistic semantic approach for discovering web services. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, pp. 1221–1222, New York, NY, USA, ACM (2007)
71. Macaulay, L.A.: Requirements engineering. Springer, London (1996)
72. Mahbub, K., Spanoudakis, G.: Monitoring WS-Agreements: An Event Calculus-Based Approach. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 265–306. Springer, Heidelberg (2007)
73. Mahmood, S., Lai, R., Kim, Y.-S., Kim, J.H., Park, S.C., Oh, H.S.: A survey of component based system quality assurance and assessment. Information & Software Technology 47(10), 693–707 (2005)
74. Maiden, N.: Servicing your requirements. Software, IEEE 23(5), 14–16 (2006)
75. Marks, E.A., Bell, M.: Service Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology. Wiley, Chichester (2006)
76. Melo, W.: Rup for cmmi compliance: A methodological approach (July 2008)
77. Mitra, T.: Business-driven Development (2005), http://www-128.ibm.com/developerworks/webservices/library/ws-bdd/index.html
78. Paulk, M., Curtis, B., Chrissis, M., Weber, C.: The capability maturity model for software. Technical report, SE Institute Carnegie Mellon (1993)
79. Lizuka, Y., Fenton, N., Whitty, R.: Software Quality Assurance Measurement Perspective. International Thomson Computer Press, UK (1995)
80. Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, OMG (Feb. 2006), http://www.bpmn.org
81. Oppenheimer, D., Patterson, D.A.: Studying and using failure data from large-scale internet services. In: EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop, pp. 255–258, New York, NY, USA, ACM (2002)
82. International Standards Organisation. Information technology - software process assessment. 2, International Standards Organisation, Parts 1-9 (1998)
83. Web Services Business Process Execution Language Version 2.0 – OASIS Standard. Technical report, Organization for the Advancement of Structured Information Standards (OASIS) (Mar. 2007)
84. Osterweil, L.: Strategic directions in software quality. ACM Comput. Surv. 28(4), 738–750 (1996)

85. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. Computer 40(11), 38–45 (2007)
86. Papazoglou, M.P.: The challenges of service evolution. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 1–15. Springer, Heidelberg (2008)
87. Pautasso, C., Alonso, G.: The jopera visual composition language. Journal of Visual Languages and Computing (JVLC) 16, 119–152 (2005)
88. Pei-Breivold, H., Larsson, M.: 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)
89. Pilioura, T., Kapos, G.-D., Tsalgatidou, A.: Seamless federation of heterogeneous service registries. In: Bauknecht, K., Bichler, M., Pröll, B. (eds.) EC-Web 2004. LNCS, vol. 3182, pp. 86–95. Springer, Heidelberg (2004)
90. Pistore, M., Marconi, A., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning at the Knowledge Level. In: Proc. IJCAI '05 (2005)
91. Pistore, M., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. In: Proc. ICAPS '05 (2005)
92. Pistore, M., Traverso, P.: Assumption-Based Composition and Monitoring of Web Services. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 307–335. Springer, Heidelberg (2007)
93. Reichert, M., Dadam, P.: Adeptflex: Supporting dynamic changes of workflow without loosing control. Journal of Intelligent Information Systems 10, 93–129 (1998)
94. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Pearson Higher Education, London (2004)
95. Rummler, G.A., Brache, A.P.: Improving Performance: How to Manage the White Space on the Organization Chart, 2nd edn. Jossey-Bass, San Francisco (1995)
96. Rus, I., Lindvall, M.: Knowledge management in software engineering. IEEE Software 19(3) (2002)
97. Santos, G.T., Lung, L.C., Montez, C.: Ftweb: A fault tolerant infrastructure for web services. In: EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference, pp. 95–105, Washington, DC, USA, IEEE Computer Society (2005)
98. Schatz, B., Abdelshafi, I.: Primavera gets agile: a successful transition to agile development. Software, IEEE 22(3), 36–42 (2005)
99. Seacord, R.C., Plakosh, D., Lewis, G.A.: Modernizing Legacy Systems. Addison-Wesley, Reading (2003)
100. Yu, S., Liu, J., Le, J.: Intelligent web service discovery in large distributed system. In: Yang, Z.R., Yin, H., Everson, R.M. (eds.) IDEAL 2004. LNCS, vol. 3177, pp. 166–172. Springer, Heidelberg (2004)
101. Sivashanmugam, K., Verma, K., Sheth, A.: Discovery of web services in a federated registry environment. In: ICWS '04: Proceedings of the IEEE International Conference on Web Services, p. 270, Washington, DC, USA, IEEE Computer Society (2004)
102. Sommerville, I., Kotonya, G.: Requirements Engineering: Processes and Techniques. John Wiley & Sons, Inc., New York (1998)

103. Stollberg, M., Hepp, M., Hoffmann, J.: A caching mechanism for semantic web service discovery. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 480–493. Springer, Heidelberg (2007)

104. Strader, L.B., Beim, M.A., Rodgers, J.A.: The motivation and development of the space shuttle onboard software (obs) project. Software Process ' Improvement and Practice 1, 107–113 (1995)

105. Sycara, K., et al.: OWL-S 1.0 Release. OWL-S Coalition (2003), http://www.daml.org/services/owl-s/1.0/

106. Talwar, V., Wu, Q., Pu, C., Yan, W., Jung, G., Milojicic, D.: Comparison of approaches to service deployment. In: 25th IEEE International Conference on Distributed Computing Systems, 2005, ICDCS 2005, Proceedings, pp. 543–552 (June 2005)

107. CMMI Product Team. Capability maturity model^{TM}integration for development. Technical report, S. E. Institute (2006)

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

109. Ulrich, W.M.: Legacy Systems Transformation Strategies. Prentice-Hall, Upper Saddle (2002)

110. Umar, A.: Application (Re) Engineering: Building Web-Based Applications and Dealing with Legacies. Prentice-Hall, Englewood Cliffs (1997)

111. van den Heuvel, W.-J.: Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems). MIT Press, Cambridge (2006)

112. van den Heuvel, W.-J., van Hillegersberg, J., Papazoglou, M.P.: 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, pp. 81–103. Kluwer, B.V, Dordrecht (2002)

113. Verna, K., Sivashanmugam, K., Shet, A., Patil, A., Oundhakar, S., Miller, J.: Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. Information Technology and Management 6, 17–39 (2005)

114. Lemahieu, W.: Web Service decription, advertising and discovery: WSDL and beyond. In: New Directions in Software Engineering, pp. 135–152. Leuven University Press, Leuven (2001)

115. W3C. Web Services Choreography Description Language Version 1.0 (2005), http://www.w3.org/TR/ws-cdl-10/

116. Warren, I.: The Renaissance of Legacy Systems: Method Support for Software-System Evolution. Practitioner Series. Springer, London (1999)

117. Weiderman, N., Northrop, L., Smith, D., Tilley, S., Wallnau, K.: Implications of distributed object technology for reengineering. Technical Report CMU/SEI-97-TR-005 / ESC-TR-97-005 (1997)

118. WS-Diamond. Characterization of diagnosability and repairability for self-healing web services. Technical report, WS-DIAMOND Project IST-516933, Deliverable D5.1 (2007)

119. Wu, Y., Pan, D., Chen, M.-H.: Techniques of maintaining evolving component-based software. In: IEEE International Conference on Software Maintenance, 0:236 (2000)
120. Zdun, U.: Reengineering to the web: A reference architecture. In: CSMR '02: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, p. 164, Washington, DC, USA, IEEE Computer Society (2002)
121. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of service-oriented analysis and design, Published: IBM developerWorks White Paper (2005)

9

Architecture Views Illustrating the Service Automation Aspect of SOA

Qing Gu[1], Félix Cuadrado[2], Patricia Lago[1], and Juan C. Duenãs[2]

[1] Dept. of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
[2] Dept. de Ingenieríade Sistemas Telemáticos, Universidad Politécnica de Madrid, Spain

Chapter Overview. Earlier in this book, Chapter 8 provided a detailed analysis of service engineering, including a review of service engineering techniques and methodologies. This chapter is closely related to Chapter 8 as shows how such approaches can be used to develop a service, with particular emphasis on the identification of three views (the automation decision view, degree of service automation view and service automation related data view) that structure and ease elicitation and documentation of stakeholders' concerns. This is carried out through two large case studies to learn the industrial needs in illustrating services deployment and configuration automation. This set of views adds to the more traditional notations like UML, the visual power of attracting the attention of their users to the addressed concerns, and assist them in their work. This is especially crucial in service oriented architecting where service automation is highly demanded.

9.1 Introduction

Service-oriented architecture (SOA) as an architectural style has drawn the attention from both industry and academia. SOA-based systems (i.e., Service-Based Applications or SBA) are constructed by integrating heterogeneous services that are developed using various programming languages and running on different operating systems from a range of service providers. Services are loosely coupled entities, often designed under open-world assumptions, distributed across organizational boundaries and executed remotely at their service providers' environment. They require a smoother transition from development to operation than traditional applications.

Consequently, the architecting of SBAs pose additional concerns as compared to traditional software applications. Some examples of these concerns include how to reason about a SOA design and how to represent the characteristics of SOA that the design delivers, how to architect the SBA to operate in an unknown environment, or how business processes can be supported by means of the collaboration of multiple services.

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, pp. 339–372, 2010.

Clearly, traditional software engineering and architecting techniques, methods and tools are no longer sufficient to deliver SBAs, as they do not take into account specificities of services, such as the need for smooth transition from development to operation, the need to integrate third-party components, or the possibility to be hosted by a different organization. Therefore, it is necessary to propose new techniques that supplement the traditional models, enabling the capture at design time of all the relevant information about those new concerns and improving the usefulness of the architecture description.

We have chosen as the aspect under study the degree of automation of SBAs. The current situation is that the design decisions on whether a service can be possibility automated, its benefits and limitations, or the degree of automation are often left implicit in the architectural design and its description. Our goal in this paper is to make them explicit and to find a notation useful for this purpose, able to be understood for most stakeholders (including the user if relevant to the domain).

We carried out two large case studies to learn the industrial needs in illustrating services deployment and configuration automation, from now on service automation. As a result, we broke down service automation into three important sub-aspects, and we developed a corresponding set of architecture views (automation decision view, degree of service automation view and service automation related data view) that expresses the different concerns of stakeholders who share interest in service automation.

The first perspective (the decision view) conveys the decisions about service automation by making explicit which architecture constraints may impact the degrees of automation, and which services are affected by each constraint. The second perspective (the degree view) shows the degree of service automation that the service flow is expected to achieve, but not the details on how to get it. Last, the automation-related data view contains explicit information about the generation, management and provision of additional input that are required from either human actors or policies.

In addition to constructing these views, we highlighted the added-value of the graphic notations we used. We argue that this set of views adds to the more traditional notations like UML, the visual power of attracting the attention of their users to the addressed concerns, and assist them in their work. Moreover, we also reflected on the relationship between the degree of automation and the granularity of services and the applicability of these views to SOA in general.

The reminder of the chapter is organized as follows. In Section 9.2, we provide some background information on architecture views and management systems for SBAs. In Section 9.3, we discuss the need of documenting SOA design decisions and rationale in effective illustrations and present a set of concerns that we elicited from the case studies, which points out what needs to be illustrated in SOA architecture description. With these requirements in mind, we present the three service automation views in Sections 9.4, 9.5

and 9.6. We highlight the power of visualization in Section 9.7 and we discuss our observations in Section 9.8. We conclude the chapter in Section 8.6.

9.2 Background Information

9.2.1 Architecture Views

The architecture of a software system should be documented with the purpose of capturing early design decisions, providing re-useable abstractions of software systems and enabling communication of the software architecture among stakeholders [4]. To produce relevant documentation for a software system, one has to decide what information needs to be documented and which notations or diagrams are suitable for representing this information. These decisions heavily depend on who is the target reader of the documentation.

A software system typically involves multiple *stakeholders* that have different concerns. For instance, the architect is concerned about the structure of the system; the project manager is concerned about the resources (e.g., cost, time, number of developers) needed for developing the system; and the developer has concerns about the implementation of the system. The architectural design of the system therefore should be documented in such a way that the concerns of each stakeholder are addressed.

Following the *separation of concerns* principle, software architects have already been using multiple views for years to represent the software systems of interest from multiple perspectives. These views facilitate the management of the complexity of software engineering artifacts. For instance, a structure view can be used to describe the construction of a software system (including, e.g., components and connectors); while a data view can be used to describe the data flow between the components. By representing the architecture of the system in these two separate views, the software architect may focus on the construction design of the system by using the structure view, while the data manager may concentrate on the management of data by using the data view.

One of the original goals behind IEEE 1471 and ISO/IEC 42010 was to "establish a frame of reference of terms and concepts for architectural description" [5]. This frame of reference provides a basis for the community to develop a collection of views which addresses concerns that occur commonly across software projects. Practitioners may directly benefit from the application of these viewpoints in that they enable an effective architecture description.

However, the existing reusable views are limited in the sense that they address concerns that often appear in traditional software architectures. With the wide adoption of recently emerged software architecture styles (like SOA), additional concerns (often specific to the architecture styles) challenge the reusability of the existing viewpoints. The lack of available views make practitioners face difficulties to find an effective way to illustrate any new characteristics introduced by any architecture style. As a result, views that enable the illustration of specific concerns introduced by modern software architecture styles are needed.

9.2.2 Management System for SBAs

Hegering has described the management of networked systems as the set of measures necessary to ensure the effective and efficient operation of a system and its resources, according to an organization's goals [7]. In service-based applications the functionality is not provided by individual, monolithic elements, but is achieved by collaboration between multiple services. In order to get this collaboration, the management system must be aware of the participating elements, deploy and configure them if necessary. This is a complex process, because as the number of services grows, the possible combinations that must be considered by the management system increase exponentially. On top of that, the distribution of the participating elements over a computing network further complicates the process. Those are common characteristics for every SBA. However, they are not the only relevant factors. The domain-specific characteristics of each SBA, such as the characteristics of the services, the capabilities of the runtime resources, or the organization's business aspects must also be supported by the management system. It is clear that the potential variation of all the factors complicates defining a general solution for SBAs deployment and configuration.

In the field of systems management, there are two opposite approaches for controlling the deployment and configuration process: traditional management processes and autonomic management [9]. In traditional processes, a human administrator is continuously in control of the change process. He or she performs diagnostics on the system, manually defines the required changes and controls every aspect of the execution. This approach is very costly and cumbersome, because it implies that every activity executed by the architecture must be performed or at least validated by a human actor. On the other hand, autonomic computing promotes to automate as much as possible the operation of the system. Ideally, completely automated closed control loops are implemented, where the system reacts automatically to a change in the environment, diagnoses its severity and implications and applies the required corrections in order to restore the environment functionality. This approach eliminates the bottleneck inherent to human operation, consequently improving scalability and efficiency of the management system.

Although autonomic control would be the most desirable approach, it is not always feasible to achieve it, because of either technical factors (e.g., a monitoring interface from a managed server does not provide information about service faults so a human administrator has to manually diagnose the incidences by inspecting the server and system logs) or, organizational aspects (e.g., manual control is preferred because the service update process is considered critical for the organization, so an automated system cannot have complete control over the process). For most cases an adequate balance between traditional and autonomic management will be the right approach. Management systems should pursue the autonomic approach to the greatest extent possible, while respecting the requirements derived from the domain of application.

Supporting the diversity of managed services, operation environments and organizational aspects with the same management architecture demands a high level of flexibility, which is pushed forward adopting a service-oriented approach. Service orientation can offer great flexibility and agility so that the architecture can easily adapt to the characteristics of different environments with reduced required configuration. The Service Deployment and Configuration Architecture (from this point onwards SDCA) [11] is an example of such an approach, which is described further in Section 9.3.1.

9.3 The Requirements for Illustrating the Automation Aspect of SBAs

The SOA paradigm promotes creating new functionality from the dynamic combination of services provided by different stakeholders. A SBA can be viewed as a set of dynamically combined services.

While automation in a traditional software system refers to the degree to which the execution of the *software process* can be automated without human intervention, automation in SOA systems refers to the degree to which *services*, comprising the system of interest, can be executed automatically without any human intervention. While the two definitions are quite similar, due to a set of characteristics that differentiate SBAs from traditional software systems [6], in service-oriented development the decision on the degree of automation of each service is heavily influenced by (and has impact on) at least two quality attributes.

The first quality attribute is trust, i.e., confidence (especially from the users perspective) on the truth of what delivered or promised. SBAs are typically not fully controlled by the company: some integrated services execute in the domain of remote, dynamically determined service providers, and can be discovered and integrated at runtime. This means that if something goes wrong, malfunctions might decrease the satisfaction of ones customers, and hence influence the overall company business. Especially in traditional business domains (like banking and services to the public) the tendency is to develop applications with service-oriented technologies, but with the properties of "old fashion" software systems: low level of automation, static integration of services, no dynamic discovery and no dynamic composition. In the case of required interaction with third-party services, the requirements – both functional as non functional – of these and the penalties for failure, are governed by Business Level Agreements (BLA) or Services Level Agreements (SLA).

The second quality attributes is reliability, i.e., the ability of a software system to perform its required functions under stated conditions for a specified period of time [3]. By automatically integrating services during execution, reliability of the whole execution depends on various unpredictable factors, like the correct specification of the requirements of the services to be dynamically integrated, availability of such services, or their correct execution. If third-party service discovery & composition is automated, the company does not have anymore full control on the software products delivered to its customers.

It is often claimed that SBAs have the agility to adapt to customer needs by automatically reacting to continuous changes in business processes. Consequently, the more services in a SOA system can be automated (i.e., do not need humans to make decisions for execution), the higher agility a SBA can achieve.

Users of highly automated SBAs clearly benefit from less human intervention and thus less labor costs. However, automating the execution of services and delivering agile and reliable SBAs is not always possible (as we explained above in the examples of trust and reliability) and poses additional concerns. Some of the concerns relate to the decisions on the degree of service automation; while some of the concerns are related to the realization of these decisions.

However, design decisions and their associated rationale on whether a service can be possibility automated, the benefits and limitations of automating a service, or how to automate a service are often either ignored or left implicit in the architectural design and its description. In spite of the evidence for the need of documenting design decisions and rationale in effective illustrations [10, 1] little work exists so far in the area of SOA [6]. This need has been further highlighted in the S-Cube analysis of the state of the art in [2], where one major challenge is in identifying and representing relevant concerns in SBA engineering, like monitoring, self-organization and adaptation. Viewpoints are mentioned as means to capture multiple perspectives on a given SBA. Though, they are meant to aid engineering of specific systems, whereas the corresponding architecture descriptions have not been sufficiently addressed yet. This motivated us to investigate what the stakeholders are concerned about with respect to service automation and how to address these concerns in the architecture description.

To answer this question, we analyzed the service automation aspect of the SDCA as well as two concrete industrial case studies where the SCDA has been applied to. The first case study (BankFutura) describes the deployment and configuration system of a banking organization. As for most enterprise systems, in this case the non-functional requirements such as the criticality of the delivered services, guaranteed performance levels, and organizational aspects are the dominating factors for driving the decisions on the degree of

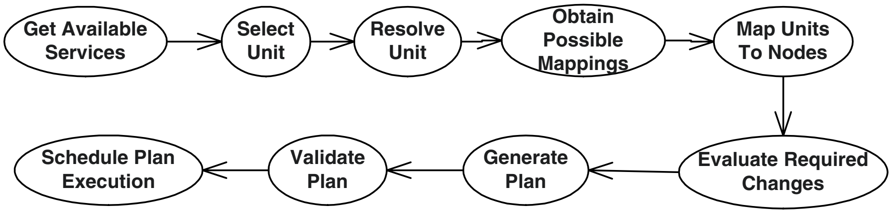


Fig. 9.1. The SDCA Deployment Service Execution Flow

automation of the service execution flow. In the second case (HomeFutura) the implementation of SDCA provides the services of multiple third-party providers, which are presented to the end users through a service catalog, allowing them to select the functionality they require. While the same service execution flow is adopted in both cases, there are significant variations in the automation related aspects, due to the impact of their domain-specific and organization-specific constraints.

In the remaining of this section, we present an overview of the SDCA and its industrial case studies (BankFutura and HomeFutura), focusing on the concerns related to service automation that we have elicited from the cases. These concerns serve as the requirements for illustrating the automation aspect of SOA in the architecture description.

9.3.1 The Service Deployment and Configuration Architecture

The Service Deployment and Configuration Architecture (SDCA) is a flexible, service-oriented management architecture that can address the requirements of distributed, heterogeneous SBAs (a.o. dynamic discovery, dynamic composition, adaptation, runtime evolution). The management functions are provided by a set of services, which collaborate to identify the required changes to the environment in order to fulfill the SBA business objectives (a.o. service availability). SDCA Services are automated, reasoning over models representing the characteristics of the managed services and the runtime environment. Finally, in order to adapt to the domain-specific characteristics, the specific behavior of the services can be customized through the definition of policies that govern the decisions taken over the process.

The objective of the SCDA is to provision new functionality (in the form of services) by identifying and applying a set of changes to the managed environment. This function is achieved by an execution flow consisting of the combined invocation of nine deployment services, as shown in Figure 9.1.

A typical execution starts when an external change to the system is triggered (e.g., an updated version of a service has been released and must be deployed, or a hardware malfunction caused a server to stop working, and the affected services must be redeployed at another node). After it is decided that a change is necessary, the deployment service Get Available Units is invoked. This first step retrieves the complete list of units and services currently available. The second step is the deployment service Select Unit, where one of those available units is selected, in order to be deployed to the environment. The *unit selection criteria* will be provided to the service as an external input. After that, the deployment service Resolve Unit is invoked, where the deployment unit containing the service is analyzed, in order to find a closed set of units satisfying all their dependencies. There might be multiple candidate units satisfying one dependency (e.g., multiple units with minor, compatible versions) and for those cases a *criteria for selecting among them* must be provided as external input. Once the complete set of units that will participate in the operation has been identified, the deployment service Obtain Possible Mappings evaluates the available resources from the container, and returns for each unit the potential nodes of the environment where those can be deployed. Starting with that input, the deployment service Map Units To Nodes decides on the final destination for each one of the involved units, according to external *distribution criteria*. After those mappings have been established, the deployment service Evaluate Required Changes compares the current environment status with the desired changes in order to obtain the set of required changes that must be applied to it (e.g., install selected deployment units if they are not currently running at the environment). Those changes are packed and sorted into a deployment plan in the deployment service Generate Plan, whose purpose is to ensure a correct execution of the list of changes, by adding restrictions to their execution order. Defined plans can either be instantly applied to change the environment, or can be temporarily stored at a change repository. Before being applied to the environment, plans must pass through the deployment service Validate Plan. This step checks that the automatically obtained plan is coherent with the environment state, and will obtain the desired result. Finally, the deployment service Schedule Plan Execution receives the accepted plan and schedules it for execution at some point in the future, which will be also determined by an external *schedule agenda*.

Some of the deployment services can be *completely automated* as they do not require any external intervention, such as Get Available Service and Obtain Possible Mappings; whereas some others cannot be completely automated as *external input* (i.e., additional input external to the service invocation flow)(e.g., distribution criteria, unit selection criteria) are required during the deployment process. The architect of SDCA provided services requiring external input with certain degree of flexibility, supporting two alternatives for their implementation. One of the solutions is to create a user interface so that a dedicated human actor can provide the required input to those services. Another solution is to formalize the necessary knowledge for providing the required input in terms of policies, which can be automatically consumed bt the deployment service. SDCA services with the format approach are called *semi-automated services* whereas the latter are called *policy-driven automated services*.

The stakeholders of the SDCA that are concerned about service automation include *the SOA architect*, who is responsible for defining a deployment service flow that can support the automated provisioning of services, adapting to the hardware characteristics of each environment; *the SOA manager*, who governs the design and implementation of the SDCA, and *the users of the SDCA*, who are often the SOA architects that intend to apply the SDCA to specific domains.

Being the designer of the SDCA, the SOA architect is mainly concerned about how to provide enough flexibility with the degree of automation, in order to allow adaptation of the flow to specific domain requirements. Additionally, the SOA architect is concerned about how to support the other stakeholders in terms of service automation. SDCA users are mainly concerned about how to customize the SDCA in such a way that domain specific constraints are fulfilled.

The complete list of concerns of each stakeholder is presented in Table 9.1, where each concern is described by its associated stakeholder, a description, and a concern ID.

Table 9.1. Concerns Relevant to Service Automation in the SDCA

Stakeholder	Concern ID	Concern description
SOA architect	SDCACon1	Justify whether their decisions on the degree of service automation are reasonable.
	SDCACon2	Provide enough flexibility with the degree of automation, in order to allow adaptation of the deployment service flow to specific domain requirements.
	SDCACon3	Suggest policies that are required for assisting the deployment service flow.
SOA manager	SDCACon4	Trace, verify and control the decisions on the degree of service automation.
	SDCACon5	Gain an overview of the degree of service automation supported by the SDCA.
	SDCACon6	Gain an overview of the re-configurability of the SDCA.
SDCA Users	SDCACon7	Be aware of which deployment services are domain specific (hence customization is needed) and which ones are domain independent (hence no customization is necessary).

9.3.2 BankFutura: An Application of the SDCA to an Enterprise Domaina

A Spanish banking company, called from this point on BankFutura, with several millions of clients over the world, and more than two thousand branches, renovates its services portfolio, which includes client services (internet banking, cashiers), internal services (for company workers at the bank offices) and B2B services for inter-bank transactions. As those services capture the company knowledge, they are internally developed and provided, with no third party dependencies. This is understandable, as they constitute the core of the company business and consequently must be under full control of the company. The company services have been architected following the SOA / BPM paradigm, in order to cope with the complexity.

The services runtime infrastructure that will replace the legacy systems and mainframes is composed by artifacts such as relational databases, JEE application servers, and BRM (Business Rule Managers) systems. Each artifact of the system is presented as a banking service, hiding its implementation details and providing a uniform high-level view. Banking services are published in directories and connected through an ESB (Enterprise Service Bus). The complete runtime infrastructure is dimensioned and defined beforehand, in order to support the strict non-functional requirements for the service operation, as well as adequately support the types of services that will provide the core banking functionality.

In BankFutura, every deployed banking service must be always available, respecting the requirements defined at the SLA, while dispatching the requests from a potentially enormous number of consumers. Neither hardware and software malfunction, or denial of service attacks from ill-intended actors, should be able to disrupt the service operation, as service downtime would imply huge monetary costs. The stability of the banking system becomes one of the most important non-functional requirements.

Another non-functional requirement for the banking system is security. The exchanged information of banking services is very sensible, as it contains the financial status and personal data of the clients, so it is not only critical for their trust but also legally protected by the personal data confidentiality regulations. Because of that, it is fundamental to safeguard the security of the underlying systems, and provide complete logging and traceability of the performed operations. This way, change initiation, approval and execution must be registered and supported by the change management architecture, including a responsibility chain for any identified incidences.

In order to respect all these restrictions and facilitate at the same time system evolution, the BankFutura infrastructure is replicated into several, tiered environments (integration, pre-production and production) which present a balance between agility of changes and criticality. The complexity and predefined structure of the deployment and configuration architecture justifies that BankFutura employs specialized staff, such as Environment administrators, for watching over the runtime health, diagnosing malfunctions and controlling the execution of planned changes to the environment, despite the costs the bank is incurring by keeping this staff.

The stakeholders who are concerned about service automation in BankFutura include the SOA architect, the banking deployment plan creator, the environment administrator, and the service deployment manager.

The SOA architect is responsible for applying and customizing the SDCA so that the resulting deployment service flow can support the complete provisioning of the released services to the several tiered environments of the infrastructure of the company. He or she is mainly concerned about automating the deployment services as much as possible (moving away from handcrafted scripts) while at the same time integrating human control and responsibility over the complete process.

The banking deployment plan creator and *the environment administrator* are both deployment actors in the banking deployment service flow. The former is responsible for creating a deployment plan which will provide the desired functionality when applied to the environment; while the latter is responsible for the correct configuration of the managed infrastructure and the selection of the right physical node for each newly deployed service, taking into account the additional resources consumption by each new service. Both of them are mainly concerned about how to perform their roles in the banking deployment service flow.

The deployment manager is in charge of supervising the execution of banking deployment service flow and ensuring that the deployed banking system is aligned with the business objectives of BankFutura.

The detailed concerns of each stakeholder are listed in Table 9.2, where each concern is presented with its associated stakeholder, a description, and a concern ID.

9.3.3 HomeFutura – An Application of the SDCA to a Personal Domain

The service aggregator of this case study (called from this point on HomeFutura) wants to offer subscribers a large catalog of services that can be consumed from the devices available at the digital home. The digital home is the house of the near future, an always connected entity, provided with network and devices to access Internet resources. It allows users to consume a wide range of services; multimedia entertainment services, surveillance services, e-learning services or consumer electronics control, just to mention a few. Services are provisioned over the Internet and accessed through multiple home devices. The specific hardware elements that will be available are controlled by the end users, which can dynamically decide to acquire additional equipment.

The ultimate goal is to create an environment that benefits end users, service providers, and service aggregators. End users should be able to browse all available services and subscribe to those they are interested in, automatically accessing them without technical skills. Service providers develop and offer services to be consumed by the end users. Service aggregators are the point of contact with the users, managing their subscription, interacting with the service providers, and ensuring correct and seamless service provisioning.

In this case study the deployment architecture plays a fundamental role. In order for those services to be available, it is necessary to execute deployment and configuration activities over the home infrastructure. The general characteristics of the environment are similar to the previous case, with the required

Table 9.2. Concerns Relevant to Service Automation in BankFutura

Stakeholder	Concern ID	Concern description
SOA architect	BankCon1	Justify whether their decisions on the degree of service automation are reasonable given the specific constraints in the BankFutura.
	BankCon2	Understand what specific constraints affect each deployment services and how each constraint influences the degree of service automation.
	BankCon3	Analyze the possible alternatives on the degree of service automation in order to evaluate how the deployment service flow can react to changing requirements or constraints.
The deployment manager	BankCon4	Trace, verify and control the decisions on the degree of service automation.
	BankCon5	Gain an overview of the degree of service automation in the BankFutura deployment service flow.
	BankCon6	Ensure that the environment administrator and banking deployment plan creator carry out the assigned tasks as expected and are able to trace responsibility in case an error occurs.
	BankCon7	Ensure the availability of required policies that are required for the deployment services in time.
	BankCon8	Ensure that the required policies for the deployment process are aligned with the organizational goals and regulations.
Environment administrator	BankCon9	Ensure the stability of the managed environment after executing the deployment services.
	BankCon10	Define the role and responsibility in preparing policies.
	BankCon11	Define the role and responsibility in the deployment service flow.
Banking deployment plan creator	BankCon12	Select the right physical node for each newly deployed service, taking into account the reasons that led to the initial definition of the environment topology
	BankCon13	Know which services (and what version of the service) must be made available in each environment.
	BankCon14	Define the role and responsibility in the deployment process.

operations consisting of managing services running over a distributed, heterogeneous infrastructure. However, the specific characteristics of this scenario lead to a different solution. In contrast with the case study of BankFutura, environment stability is not the dominating constraint. This is because the domain is personal, and the services are consumed and used without warrantee of performance nor agreed Quality of Service expressed through BLAs or SLAs. On top of that, guaranteeing stability is much harder, because of the high degree of uncertainty about the specific equipment that will be available at every moment.

Instead, the fundamental goal in HomeFutura is being able to provide the end user with a seamless experience in the process of acquiring new services. The user is not concerned about the technical details behind the services or the installation process. Those aspects must be correctly managed by the architecture, while the user is only informed about the relevant information, like functionality, or pricing.

The stakeholders who are concerned about service automation in HomeFutura include the SOA architect, the service aggregator, and the end user.

The SOA architect is responsible for defining the architecture of the digital home service deployment system by applying the SDCA. The main concern consists of how to provide a flexible deployment system that is able to adapt to the available infrastructure at each home while at the same time hiding all the technical details from the end user.

The role of a *service aggregator* is to manage the service catalog available to the different users and handle the signed contracts with service providers, ensuring that the portfolio of services offered to the users can have all their technical dependencies correctly satisfied. The service aggregator is also responsible for providing selection policies which determine what providers / versions for the services can be accessed by each different client.

The end user consumes the available services offered by HomeFutura, demanding as much variety in the services catalog as possible. The end user is mainly concerned about the simplicity of the process of accessing the desired functionality.

The detailed concerns of each stakeholder are listed in Table 9.3, where each concern is presented with its associated stakeholder, a description, and a concern ID.

Table 9.3. Concerns Relevant to Service Automation in HomeFutura

Stakeholder	Concern ID	Concern description
SOA architect	HomeCon1	Justify whether the decisions on the degree of service automation are reasonable given the specific constraints in HomeFutura.
	HomeCon2	Understand what specific constraints affect each deployment service and how each constraint influences the degree of service automation.
	HomeCon3	Analyze the possible alternatives on the degree of service automation in order to evaluate how the deployment service flow can react to changing requirements or constraints.
	HomeCon4	Design a highly automated deployment process, with a minimal requirement on human intervention
The deployment manager	HomeCon5	Trace, verify and control the decisions on the degree of service automation
	HomeCon6	Gain an overview of the degree of service automation in the HomeFutura deployment service flow.
	HomeCon7	Ensure the policies that are required in the deployment process are ready in time
	HomeCon8	Ensure the policies that are required in the deployment process are aligned with the organizational goals and regulations
Service aggregator	HomeCon9	Define the role and responsibility in preparing policies.
End user	HomeCon10	Participate in the deployment process the simplest way possible

9.3.4 Summary

From the analysis of the SDCA and two industrial case studies, we observed that service automation is especially important during design and is relevant to multiple stakeholders in that we identified a considerable number of service automation related concerns. Being considered, designed and implemented, however, those concerns have not been explicitly addressed in the architecture description.

Instead, the current architecture description of the SDCA (as well as the two case studies) addresses the service automation related concerns in a very abstract way. For instance, it is stated that the SDCA provides a flexible solution that can be easily customized in various domain applications. However, the information about how flexible the solution is, how easy the solution can be applied, and how to customize the SDCA in specific domains is lacking. Hence, there is a need to find an effective way to illustrate how the concerns related to service automation are addressed in the architecture description.

In other words, we face the questions of *what information should be documented in the architecture description* and *how to document it in an effective way so that the stakeholders can easily understand it.* To answer the first question, we synthesized the concerns listed in Tables 9.1, 9.2, and 9.3. The reason for doing so is that we noticed that a reasonable numbers of concerns are overlapping and demanding for the same type information. For instance, the concerns with ID SDCACon1, BankCon1 and HomeCon1 are all about justifying the decisions on service automation but in different cases (hence overlapping); and concerns with ID SDCACon3, BankCon7, BankCon10, BankCon13, HomeCon7, HomeCon9 all demand for illustrating the information that is related to generate and access policies.

After the synthesis, we identified eight main concerns that are representative for the complete set elicited from the SDCA and its two case studies. *Decision on the degree of automation* covers all the concerns that are related to the decisions on service automation and their justification ; *Reconfigurability in terms of automation* covers all the concerns related to alternatives on the degree of service automation; *The impact of architecture constraints on the degree of automation* covers all the concerns related to domain-specific constraints; *Degree of automation* covers all the concerns related to the degree of automation a SBA can achieve; *Accountability* covers all the concerns related to the responsibility of stakeholders; *The preparation of policies* covers all the concerns related to the readiness of policies; *The specification of policies* covers all the concerns related to the content of policies; and *Human participation* covers all the concerns related to human actors with regards to their involvement in the deployment service flow.

Further, we noticed that some of these main concerns are inter-related. More specifically, the first three main concerns are about decisions, alternatives and constraints, which form a cause-effect-rationale relation. As such, we decided to address all these concerns using a **decision view**. The next two main concerns are about the degree of automation resulted from the design and the impact of such a degree on the execution of the deployment service flow. As such, we decided to address these concerns using a **degree view**. The last three main concerns are about the policies and human participation for enabling different degrees of service automation. Since policies and input from human actors can both be considered as data, we decided to use a **data view** to address the concerns. Hence the automation decision view, degree of service automation view and service automation related data view illustrate the service automation aspect for the architecting of SDCA.

The mapping between the elicited concerns, synthesized concerns and views for addressing these concern is presented in Table 9.4.

Table 9.4. Mapping between the Elicited Concerns, Synthesized Concerns and Views

View	Main concern	Concerns in the SDCA	Concerns in BankFutura	Concerns in HomeFutura
Automation decision view	Decision on the degree of automation	SDCACon1, SDCACon4	BankCon1, BankCon4	HomeCon1, HomeCon5
	Reconfigurability in terms of automation	SDCACon2, SDCACon6	BankCon3	HomeCon3
	Impact of architecture constraints on the degree of automation	SDCACon7	BankCon2	HomeCon2
Degree of service automation view	Degree of automation	SDCACon5	BankCon5	HomeCon4, HomeCon6
	Accountability	-	BankCon6, BankCon11, BankCon14	HomeCon10
Service automation related data view	The preparation of policies	-	BankCon10, BankCon7	HomeCon7, HomeCon9
	The specification of policies	SDCACon3	BankCon8, BankCon9, BankCon11, BankCon13	HomeCon8
	Human participation	-	BankCon11, BankCon14	HomeCon10

9.4 The Automation Decision View

The automation decision view is designed to illustrate all the decisions that have been made on the degree of service automation, the rationale behind them, and the impact of domain specific constraints on the decisions. With these requirements in mind, we created a set of graphic notations for constructing the automation decision view, as no other notation in the literature fits to our purposes. These graphic notations are presented in Figure 9.2.

In this figure, services are represented by ovals; the three ones in the first column represent the three different degrees of service automation that have

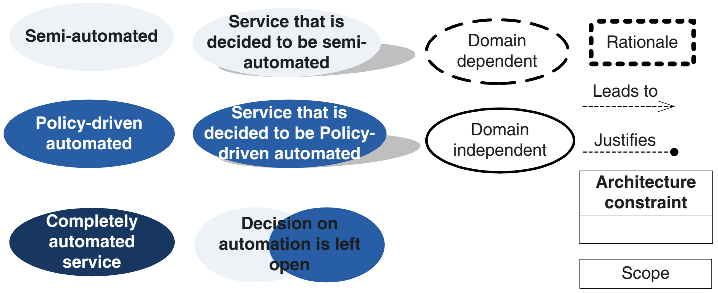

Semi-automated
Service that is decided to be semi-automated
Domain dependent
Rationale
Leads to
Policy-driven automated
Service that is decided to be Policy-driven automated
Domain independent
Justifies
Architecture constraint
Completely automated service
Decision on automation is left open
Scope

Fig. 9.2. The Graphic Notations for the Automation Decision View

been decided. Moreover, they also indicate that alternative degrees of service automation are not feasible or reasonable. The services in the second column represent a decision has been made or left open among alternative degrees of service automation. These services indicate that they can be re-configured to an alternative degree of service automation if necessary. These two sets of notations are meant to address the concerns of *decision on the degree of automation* and *reconfigurability in terms of automation*.

The two notations in the third column indicate the dependency between a degree of service automation and a specific domain. The notations in the last column are used to illustrate the relation between decisions, architecture constraints, and associated rationale, as well as the scope of services where architecture constraints may have impact on. These two set of notations are meant to address the concerns of *the impact of architecture constraints on the degree of automation*.

9.4.1 The Automation Decision View for the SDCA

The automation decision view for the SDCA is presented in Figure 9.3. This view aids the SOA architect in taking design *decisions on service automation* by making explicit which architecture constraints may impact the degrees of automation, and which services are affected by each constraint.

This view differentiates the degrees of service automation that are domain-dependent from the ones that are domain-independent. More specifically, the four services that are completely automated are circled with a straight edge line, indicating that they are domain-independent in terms of automation (SDCACon7). The other five services, however, are designed being both semi-automated and policy-automated. As such, the SDCA offers its users the flexibility to decide on which degree of service automation to be configured for specific domains, based on domain-specific constraints, such as quality attributes or characteristics of the execution environment (SDCACon7).

As an example of domain-independent decision, the deployment service Generate Plan automatically sorts a list of operations, ensuring they are executed in a correct order. The execution of this service only requires the input provided by the previous deployment service Evaluate Required Changes. Hence, the deployment service Generate Plan can be completely automated, independent with any domain specific constraints.

As an example of domain-dependent decision, the service Map Units to Nodes decides the physical distribution of the participating services, among a list of potential mappings provided by the service Obtain Possible Mappings. Depending on the specific domain characteristics, the criteria for making those decisions will be different, as well as the relevance of this decision (ranging from any solution is acceptable to only one distribution is correct). Because of those factors, this service has been designed with flexibility on its degree of automation.

From these examples, we can see that the SOA architect can use this view to explain why some of the services have been designed to be completely automated while others have been designed for both semi-automated services and policy-driven automated services (SDCACon1) and the SOA manager is able to use this view to trace, verify and control these decisions (SDCACon4).

Highlighting the links between the architecture constraints and the decisions, this view facilitates the SOA architect to show the flexibility of adapting the SDCA to specific domain applications. It is obvious from the view that the services whose decisions on service automation are left open, require further re-configuration when architecture constraints become specific (SDCACon2, SDCACon6). The users of the SDCA also benefit from this view by being aware of the impact of certain architecture constraints on the degree of service automation.

For instance, the deployment service Select Unit aims at selecting the main service to be deployed to the environment (and the enclosing deployment unit). The decision on the degree of automation is mainly driven by quality attributes of the deployed system and the technical capabilities of the deployment actor. An automated solution results in less human control; whereas a non-automated solution requires certain skills from a deployment actor.

To give another example, the deployment service Map Units To Nodes decides which physical node among the candidates will host each participating unit. Whether such a decision can be obtained from a certain policy or has to be controlled by a human actor is influenced by the characteristics of the environment and the domain quality attributes. Strictly defined environments will generally impose stricter distribution requirements, which need to be provided by a human actor, whereas the environments defined on-the-fly generally lead towards programmatic distribution policies such as round robin or even load balancing.

9.4.2 The Automation Decision View for BankFutura

When applying the SDCA to BankFutura, the four completely-automated services whose automation state is domain-independent require no further decisions and hence remain being completely-automated. On the other hand, the five services implemented as both semi-automated and policy-driven automated in the SDCA require further decisions on re-configuration based on the BankFutura specific architecture constraints. The outcome of those decisions is illustrated in the automation decision view for BankFutura, presented in Figure 9.4.

The view shows how the domain non-functional requirements such as criticality or reliability limit the degrees of service automation that BankFutura actually can operate with, in spite of the advantages of a completely automated service execution flow. As a result, the SOA architect decided to semi-automate most of the services to guarantee a certain degree of control over the deployment process (`BankCon1, BankCon2`). The only service that was decided be policy-driven automated is the deployment service Resolve Unit. An exception was made in that case because BankFutura services are developed and provided by the internal IT department, satisfying the business needs of the organization and are developed according to internal policies. This suggests a simpler and predictable dependency resolution activity and hence it was decided to be automatically driven by policies rather than human actors.

As we can see the requirements of criticality or reliability as well as their impact on the degree of service automation are highlighted in the view (`BankCon2`). Not only the SOA architect can use this view to justify their decisions on the degree of service automation satisfying the requirements of criticality or reliability, but also the deployment manager can use it to trace, verify and control the decisions that the SOA architect made (`BankCon4`).

Explicitly documenting the rationale for the decisions on the degrees of service automation also enables the analysis of the possible alternatives on the degree of service automation, allowing to evaluate how the deployment service flow can react to changing requirements or constraints (`BankCon3`). If BankFutura intends to reconfigure the degrees of service automation, it will be useful to know the automation alternatives and the trade-off among them. For instance, the services presented in Figure 9.4 are marked with a shadow if they can be either semi-automated or policy-driven. Although the services have been decided to implemented with either of the degrees of service automation, they could be reconfigured to another degree if the organization deemed it necessary (e.g., after a time of operation the organization increased its trust in the automation capabilities of the BankFutura, and opted to increase the degree of automation for a more efficient operation).

9.4.3 The Automation Decision View for HomeFutura

Specialized from the degree of automation view for SDCA (presented in Figure 9.5), this view presents the implemented degree of automation for HomeFutura services.

Similar to BankFutura, the degree of automation view for HomeFutura (presented in Figure 9.5) does not display the four completely automated services that are domain independent and need no further decisions. This view emphasizes the decisions on the degree of automation for the other five services that are domain-dependent, as well as the domain specific constraints that lead to these decisions.

Whereas in BankFutura environment stability is the dominant constraint, HomeFutura aims at providing end users the flexibility to experience new services available on the network. Consequently, HomeFutura opted for a deployment solution with higher degree of automation, not only because end users are not capable of providing technical input to the deployment process but also because agility in the execution is required. The view presents this rationale and shows its link with the architecture constraints(`HomeCon2` influencing the decisions.

As a result, only two services that require input from end users are semi-automated. The service Select Unit requires the end users to select the services that they would like to experience; and the other service Validate Plan requires the end users to approve the execution of the deployment plan (e.g., the cost of news service, the changes to multimedia player device). The rest of the services in the deployment process are all automated and do not require human intervention . Similar with the decision view for BankFutura, this view enables the SOA architect to justify the decisions and supports the deployment manager in governing the decisions (`HomeCon1, HomeCon5`).

It is worth to note that three services are marked with a shadow, which indicates that a choice has been made between semi-automation and policy-driven automation for these services. Although policy-driven automation has been chosen for the current deployment solution for HomeFutura, the shadow reminds the architect that this service could be re-configured to be semi-automated if needed (`HomeCon3`). For instance, after a time of operation it turns out the service Map Units To Nodes does not provide the most optimal mapping of the selected services to devices (due to e.g., too complicated dependency graph that Map Units To Nodes cannot interpret correctly or incomplete policy that is not able to provide sufficient information for the mapping), the architect could decide to let a technical expert decides the mapping and hence make Map Units To Nodes to be semi-automated. However, involving another deployment actor (other than the end user) in the process would cause that the user cannot instantly start to experience the new service, having to wait for the required input from the technical expert. This way, usability and agility of HomeFutura would be challenged.

9.5 The Degree of Service Automation View

Whereas the automation decision view emphasises the domain-specific constraints that lead to the decisions on the degree of service automation, the degree view focuses on the decided *degrees of service automation* for the service execution flow. As a result, only the degree of service automation that the service flow is expected to achieve is relevant in this view, while the details of how the decisions are made are irrelevant and should not be presented in this view.

As denoted by the graphic notations presented in Figure 9.6, the degrees of automation are graphically rendered by the darkness of the color assigned to each service: the darker is the color, the higher is the degree of automation. In addition, human actors are associated to semi-automated services with the purpose of highlighting who are expected to provide input to which services. The sequence between services indicates the order in which the deployment services are invoked. With this additional information, the period during which human intervention is (and is not) required becomes explicit. By illustrating that external inputs are expected to be provided by *whom* and *when*, this view also addresses the *accountability*.

9.5.1 The Degree of Service Automation View for the SDCA

Applying the graphic notations presented in Figure 9.6, we constructed the degree of service automation view for the SDCA shown in Figure 9.7.

Using this view, the SOA manager can gain an overview of the degree of service automation supported by the SDCA, as a result of the decisions illustrated in Figure 9.3. More specifically, two different degrees of automation are designed for the SDCA (`SDCACon5`). Deployment services that do not require additional information are completely automated, while those needing external information are designed to retrieve the external information either from human actors or from policies.

9.5.2 The Degree of Automation View for BankFutura

Analogous to the SDCA, we also used the described notation to construct the degree of service automation view for BankFutura, as it can be seen in Figure 9.8.

The deployment manager can see from this view the three degrees of service automation designed for the BankFutura (`BankCon5`). More specifically, four services are completely automated, one is policy-driven automated and four are semi-automated. In other words, this view shows that nearly half of the services require human intervention, meaning that the automation degree of the deployment process for BankFutura is relatively low.

In addition, as the semi-automated services are associated with a banking deployment plan creator and an environment administrator, the manager may use this view to trace the responsibility of these two human actors who are expected to provide input during the services execution (`BankCon6`).

Similarly, this view points out directly for the banking deployment plan creator and environment administrator which services are expecting their input. As shown in Figure 9.8, the role of the banking deployment plan creator is associated to the deployment services Select Unit and Map Units To Nodes, indicating that as soon as the deployment service flow is initiated and the banking deployment creator should be prepared to first make a selection on the available units and later on to establish mapping between selected units and physical nodes (BankCon14). On the other hand, the environment administrator is only involved in the validation and execution of the deployment plan (BankCon11).

9.5.3 The Degree of Automation View for HomeFutura

Similar with the degree of automation view created for BankFutura, Figure 9.9 shows the designed degree of automation for each service in the deployment process of HomeFutura.

The view visually highlights the fact that a higher degree of automation has been designed for HomeFutura as compared to BankFutura (HomeCon4, HomeCon6). As shown in Figure 9.9, most of the services are illustrated with dark color (indicating that the services can execute without any human intervention); while two services are in light color (indicating that human intervention is needed).

The degree of service automation view for HomeFutura can also be used to explain to the end users how they are expected to participate in the deployment process. End users can see hwo their participation consists of selecting the home services that they would like to experience using service Select Units and eventually to agree on the corresponding costs of consuming these services by using service Validate Plan. This way, it can be seen how the end users are presented only the relevant information for them, while the low level, technical details are hidden(HomeCon10).

9.6 The Automation-Related Data View

While the *degree of service automation view* highlights the degree of service automation designed for the deployment service flow, the *automation-related data view* details the design from the data perspective. This way, the questions related to the generation, management and provision of additional input (from either human actors or policies) can be answered by the automation-related data view.

The graphic notations that we created to construct the automation-related data view is presented in Figure 9.10. Besides the notations for the three degrees of service automation, we distinguish the guidelines/rules from formalized policies. While both guidelines/rules and formalized policies are relevant to service automation, the former are used by the deployment actors to drive the decision and the latter are directly accessed by deployment services to achieve policy-driven automation.

The most right hand side of Figure 9.10 shows the graphic notation denoting the relationships between elements in the automation-related data view.

More specifically, a deployment actor *is responsible for* providing an input; such input *assists the execution* of semi-automated services; guidelines / rules guide the provision of such an input; formalized policies directly *assists the execution* of policy-driven automated services; and *sequence between services* is also denoted. Given these details on the relationships between policies, deployment actors, and deployment services, the deployment actors can tell which services are expecting what information from them. Moreover, it explicitly points out which organizational guidelines or rules should this information comply with. In this way, the deployment actors can be prepared to transform this organizational knowledge to their input to services, hence facilitating *human participation.*

In addition to the graphic notation, we also constructed a table template (shown in Table 9.5) for listing the policies that are relevant to service automation in the deployment service flow. This table aids *the specification of policies* in presenting all the information relevant to the policies in a structured manner. As such, this table also aids the *preparation of the policies.*

Table 9.5. The Template for Automation-Related Policy Table

Policy ID	Policy name	Policy Description	Associated service	Controlled by	Type of format

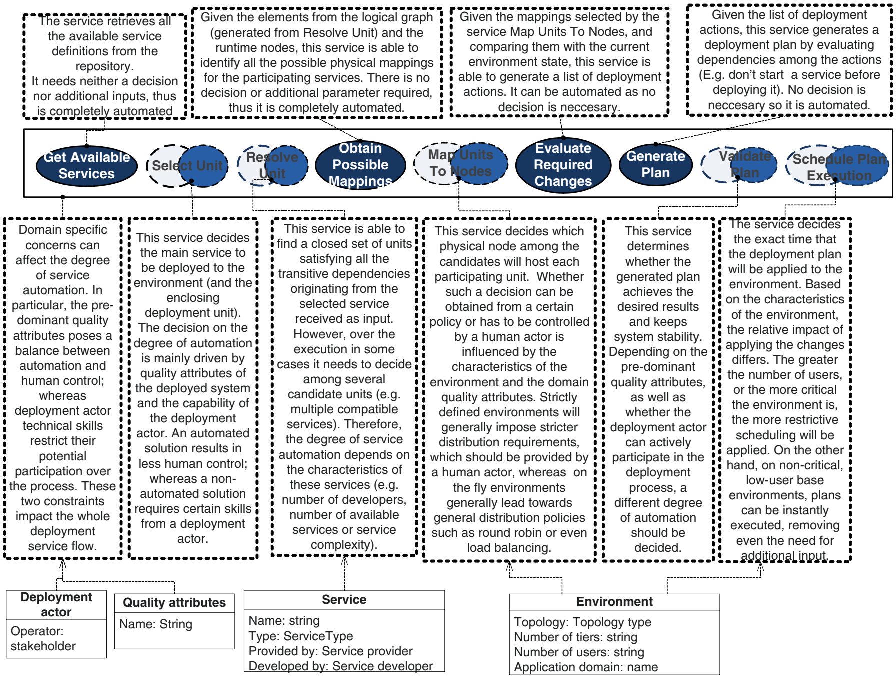
9.6.1 The Automation-Related Data View for the SDCA

As the SDCA is a reference deployment management system, it does not define concrete deployment actors or policies, as they will be specialized in specific applications. However, in order to guide the application of the SDCA, the SOA architect is concerned about identifying the required policies for providing the additional input to the deployment service flow. For this reason, we constructed the automation-related policy table, using the template presented in Table 9.5.

The *automation-related policy table* (presented in Table 9.6) provides detailed information about all the policies, including the ones for guiding human actors and the ones for policy-driven automated services. Using this table, the SOA architects in specific domains can gain an overview on what type of policies might be relevant and should be prepared, as well as which format should they be expressed when applied to the SDCA (`SDCACon3`).

9.6.2 The Automation-Related Data View for BankFutura

Applying the graphic notation presented in Figure 9.10, we constructed the automation-related data view for BankFutura (shown in Figure 9.11). This information is complemented with the BankFutura policy table, presented in 9.7)



The service retrieves all the available service definitions from the repository. It needs neither a decision nor additional inputs, thus is completely automated

Given the elements from the logical graph (generated from Resolve Unit) and the runtime nodes, this service is able to identify all the possible physical mappings for the participating services. There is no decision or additional parameter required, thus it is completely automated.

Given the mappings selected by the service Map Units To Nodes, and comparing them with the current environment state, this service is able to generate a list of deployment actions. It can be automated as no decision is neccesary.

Given the list of deployment actions, this service generates a deployment plan by evaluating dependencies among the actions (E.g. don't start a service before deploying it). No decision is neccesary so it is automated.

Domain specific concerns can affect the degree of service automation. In particular, the pre-dominant quality attributes poses a balance between automation and human control; whereas deployment actor technical skills restrict their potential participation over the process. These two constraints impact the whole deployment service flow.

This service decides the main service to be deployed to the environment (and the enclosing deployment unit). The decision on the degree of automation is mainly driven by quality attributes of the deployed system and the capability of the deployment actor. An automated solution results in less human control; whereas a non-automated solution requires certain skills from a deployment actor.

This service is able to find a closed set of units satisfying all the transitive dependencies originating from the selected service received as input. However, over the execution in some cases it needs to decide among several candidate units (e.g. multiple compatible services). Therefore, the degree of service automation depends on the characteristics of these services (e.g. number of developers, number of available services or service complexity).

This service decides which physical node among the candidates will host each participating unit. Whether such a decision can be obtained from a certain policy or has to be controlled by a human actor is influenced by the characteristics of the environment and the domain quality attributes. Strictly defined environments will generally impose stricter distribution requirements, which should be provided by a human actor, whereas on the fly environments generally lead towards general distribution policies such as round robin or even load balancing.

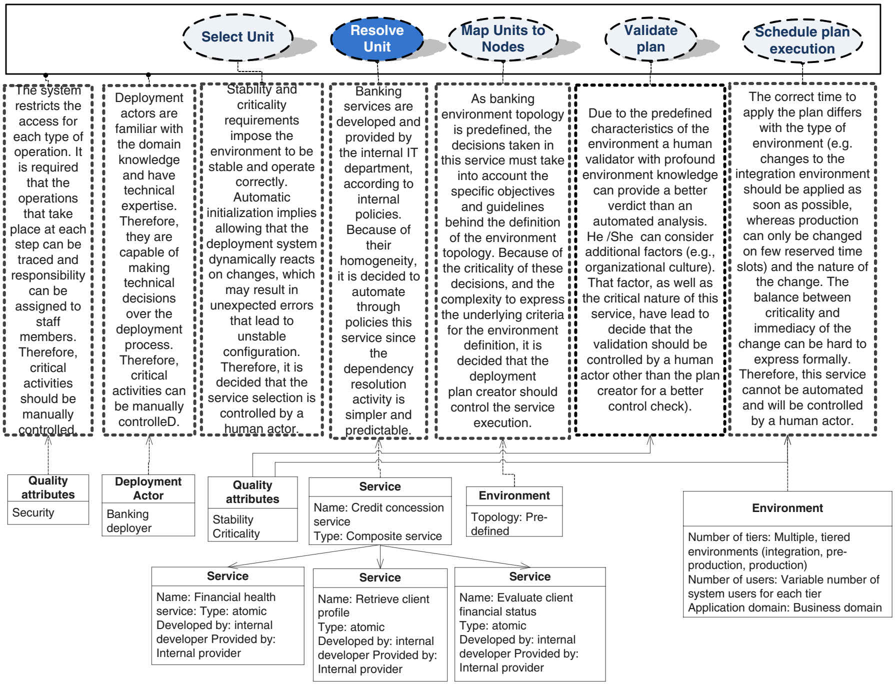
This service determines whether the generated plan achieves the desired results and keeps system stability. Depending on the pre-dominant quality attributes, as well as whether the deployment actor can actively participate in the deployment process, a different degree of automation should be decided.

The service decides the exact time that the deployment plan will be applied to the environment. Based on the characteristics of the environment, the relative impact of applying the changes differs. The greater the number of users, or the more critical the environment is, the more restrictive scheduling will be applied. On the other hand, on non-critical, low-user base environments, plans can be instantly executed, removing even the need for additional input.

Fig. 9.3. The Automation Decision View for the SDCA

Table 9.6. The Automation-Related Policy Model for the SDCA

Policy ID	Policy name	Policy Description	Associated service	Controlled by	Type of format
P01	System requirements	Describes the functionality (services) that the target environment must provide	Select Unit	-	Formal/Text
P02	Unit selection policy	Provides criteria for selecting among multiple candidate units that satisfy the same dependency	Resolve Unit	-	Formal/Text
P03	Unit distribution policy	Provides criteria for selecting the physical place of the environment where each deployment unit will be installed	Map Units To Nodes	-	Formal/Text
P04	Plan Validation Rules	Defines a set of checks that can identify fatal errors in the plan definition or potential risks expressed as warnings	Validate Plan	-	Formal/Text
P05	Environment update policy	Controls at what periods of time plans can be applied to the environment	Schedule Plan Execution	-	Formal/Text


Select Unit
Resolve Unit
Map Units to Nodes
Validate plan
Schedule plan execution
Quality attributes
Security
Deployment Actor
Banking deployer
Quality attributes
Stability
Criticality
Service
Name: Credit concession service
Type: Composite service
Environment
Topology: Pre-defined
Environment
Number of tiers: Multiple, tiered environments (integration, pre-production, production)
Number of users: Variable number of system users for each tier
Application domain: Business domain
Service
Name: Financial health service: Type: atomic
Developed by: internal developer Provided by: Internal provider
Service
Name: Retrieve client profile
Type: atomic
Developed by: internal developer Provided by: Internal provider
Service
Name: Evaluate client financial status
Type: atomic
Developed by: internal developer Provided by: Internal provider

The system restricts the access for each type of operation. It is required that the operations that take place at each step can be traced and responsibility can be assigned to staff members. Therefore, critical activities should be manually controlled.

Deployment actors are familiar with the domain knowledge and have technical expertise. Therefore, they are capable of making technical decisions over the deployment process. Therefore, critical activities can be manually controlleD.

Stability and criticality requirements impose the environment to be stable and operate correctly. Automatic initialization implies allowing that the deployment system dynamically reacts on changes, which may result in unexpected errors that lead to unstable configuration. Therefore, it is decided that the service selection is controlled by a human actor.

Banking services are developed and provided by the internal IT department, according to internal policies. Because of their homogeneity, it is decided to automate through policies this service since the dependency resolution activity is simpler and predictable.

As banking environment topology is predefined, the decisions taken in this service must take into account the specific objectives and guidelines behind the definition of the environment topology. Because of the criticality of these decisions, and the complexity to express the underlying criteria for the environment definition, it is decided that the deployment plan creator should control the service execution.

Due to the predefined characteristics of the environment a human validator with profound environment knowledge can provide a better verdict than an automated analysis. He /She can consider additional factors (e.g., organizational culture). That factor, as well as the critical nature of this service, have lead to decide that the validation should be controlled by a human actor other than the plan creator for a better control check).

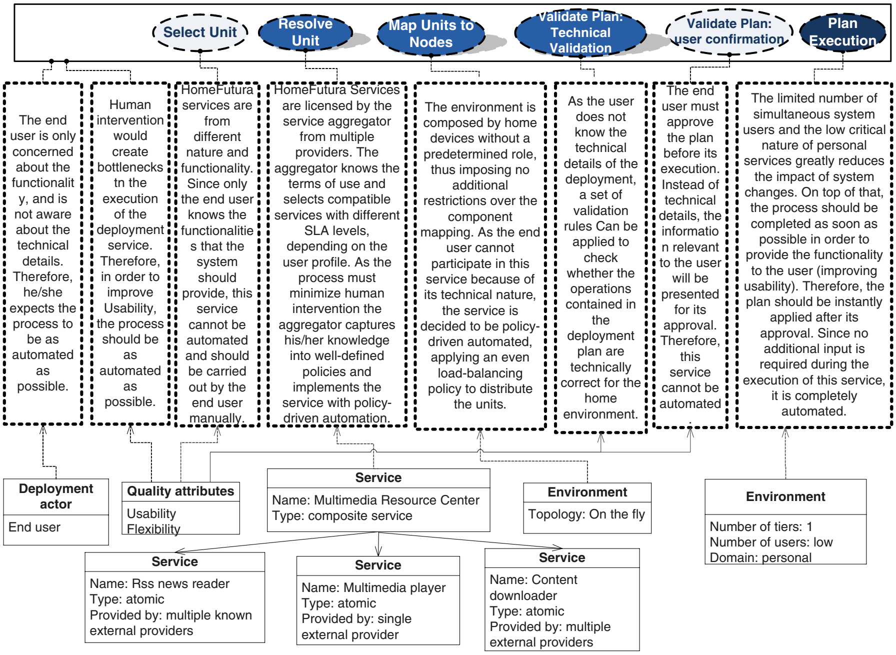
The correct time to apply the plan differs with the type of environment (e.g. changes to the integration environment should be applied as soon as possible, whereas production can only be changed on few reserved time slots) and the nature of the change. The balance between criticality and immediacy of the change can be hard to express formally. Therefore, this service cannot be automated and will be controlled by a human actor.

Fig. 9.4. The Automation Decision View for BankFutura

From this view the two human actors, the *banking deployment plan creator* and the *environment administrator*, can see what types of information they are expected to provide to which services during the service execution flow (BankCon11, BankCon14). In addition, they can see which organization policies (or guidancerules) can be referred in order to provide the required information (BankCon9, BankCon11, BankCon12, BankCon13).

We will use an example scenario to illustrate how BankCon10 is supported by the data view, guiding the participation of the *deployment plan creator* in the service *Map Units to Nodes*. In an specific environment, the environment design document mentioned in the data view informs that only one server from the environment infrastructure is configured in the network firewall to be remotely accessible from the outside; the remaining elements being protected from outer clients. In this case, by analyzing that information, the human will decide to assign units containing final services (which must be remotely accessible) to the visible server, whereas the remaining elements will be distributed over the other elements, regardless of whether those servers might also be technically capable of hosting the same types of services.

Figure 9.11 shows that the policy-driven automated service is explicitly linked to the corresponding policy. For instance, service Resolve Unit is linked


Select Unit
Resolve Unit
Map Units to Nodes
Validate Plan: Technical Validation
Validate Plan: user confirmation
Plan Execution

The end user is only concerned about the functionality, and is not aware about the technical details. Therefore, he/she expects the process to be as automated as possible.

Human intervention would create bottlenecks tn the execution of the deployment service. Therefore, in order to improve Usability, the process should be as automated as possible.

HomeFutura services are from different nature and functionality. Since only the end user knows the functionalities that the system should provide, this service cannot be automated and should be carried out by the end user manually.

HomeFutura Services are licensed by the service aggregator from multiple providers. The aggregator knows the terms of use and selects compatible services with different SLA levels, depending on the user profile. As the process must minimize human intervention the aggregator captures his/her knowledge into well-defined policies and implements the service with policy-driven automation.

The environment is composed by home devices without a predetermined role, thus imposing no additional restrictions over the component mapping. As the end user cannot participate in this service because of its technical nature, the service is decided to be policy-driven automated, applying an even load-balancing policy to distribute the units.

As the user does not know the technical details of the deployment, a set of validation rules Can be applied to check whether the operations contained in the deployment plan are technically correct for the home environment.

The end user must approve the plan before its execution. Instead of technical details, the information relevant to the user will be presented for its approval. Therefore, this service cannot be automated

The limited number of simultaneous system users and the low critical nature of personal services greatly reduces the impact of system changes. On top of that, the process should be completed as soon as possible in order to provide the functionality to the user (improving usability). Therefore, the plan should be instantly applied after its approval. Since no additional input is required during the execution of this service, it is completely automated.

Deployment actor
End user

Quality attributes
Usability
Flexibility

Service
Name: Multimedia Resource Center
Type: composite service

Environment
Topology: On the fly

Environment
Number of tiers: 1
Number of users: low
Domain: personal

Service
Name: Rss news reader
Type: atomic
Provided by: multiple known external providers

Service
Name: Multimedia player
Type: atomic
Provided by: single external provider

Service
Name: Content downloader
Type: atomic
Provided by: multiple external providers

Fig. 9.5. The Automation Decision View for HomeFutura

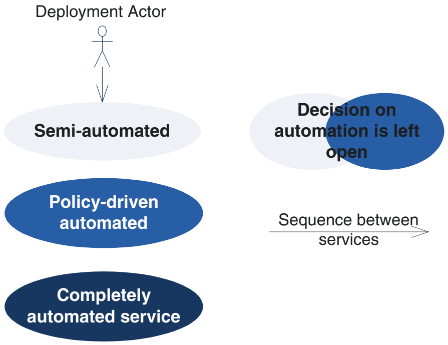

Deployment Actor
Semi-automated
Decision on automation is left open
Policy-driven automated
Sequence between services
Completely automated service

Fig. 9.6. The Graphic Notation for Constructing the Degree of Automation View

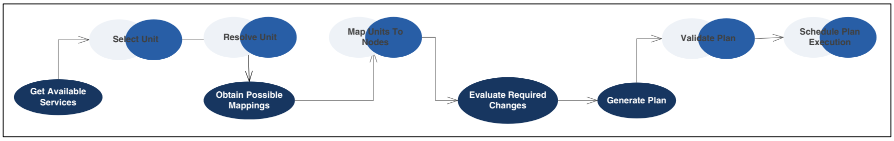


Fig. 9.7. The Degree of Automation View for the SDCA

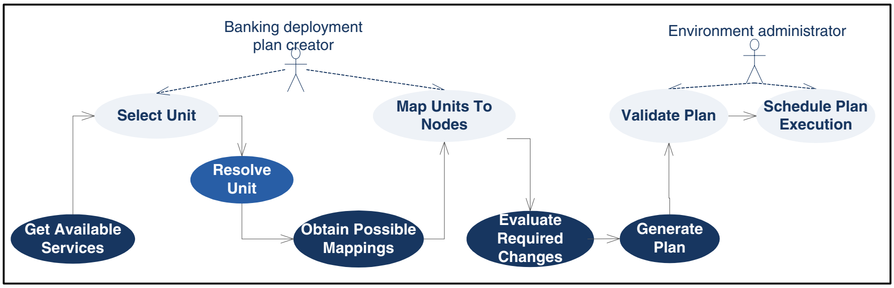


Fig. 9.8. The Degree of Automation View for BankFutura

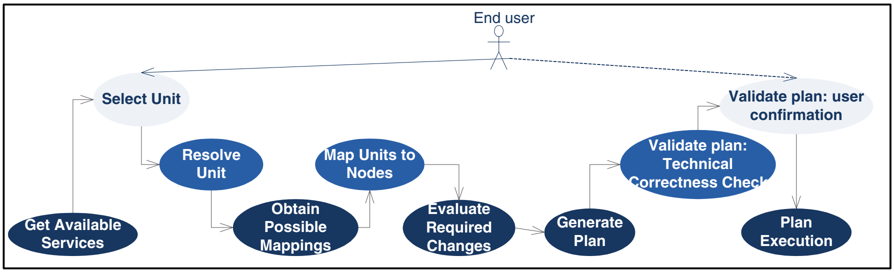


Fig. 9.9. The Degree of Automation View for the HomeFutura

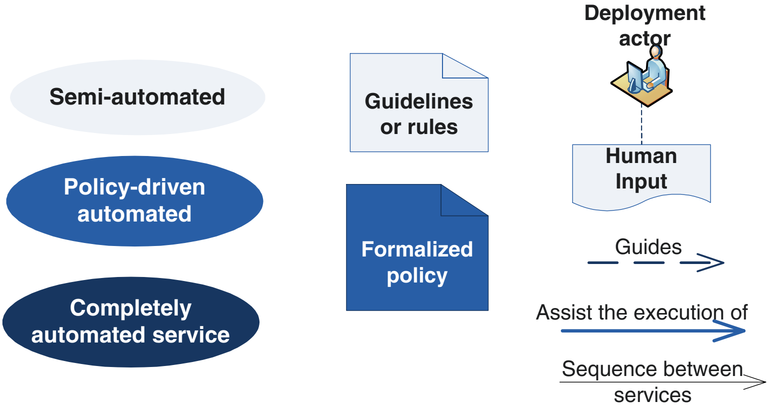


Fig. 9.10. The Graphic Notation for Constructing the Automation-Related Data View

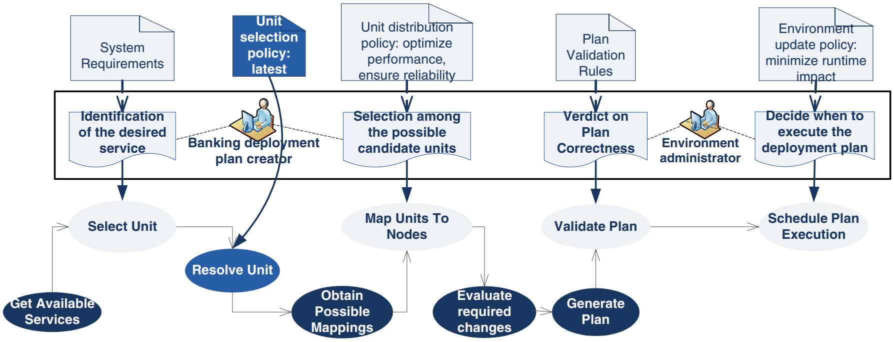

System Requirements
Unit selection policy: latest
Unit distribution policy: optimize performance, ensure reliability
Plan Validation Rules
Environment update policy: minimize runtime impact
Identification of the desired service
Banking deployment plan creator
Selection among the possible candidate units
Verdict on Plan Correctness
Environment administrator
Decide when to execute the deployment plan
Select Unit
Resolve Unit
Map Units To Nodes
Validate Plan
Schedule Plan Execution
Get Available Services
Obtain Possible Mappings
Evaluate required changes
Generate Plan

Fig. 9.11. The Automation-Related Data Flow View for BankFutura

to *Unit selection policy* indicating that the criteria for selecting among multiple candidate units defined by the policy directly influences the output of Resolve Unit. Explicitly visualizing the dependency between the services and their corresponding policies, the automation-related data flow model aids the deployment manager in obtaining an overview on when certain policies are required and which services require them during the deployment service flow (BankCon7).

Complementary to the data view, the automation-related policy table further details each policy presented in Figure 9.11 by noting its description, the role that is in charge of it and the type of format for documenting the policy. This table aids the preparation of policies as it can be used as a check list for the deployment manager to assign tasks to some specific personnel, making sure that the policies are in place and are expressed in the right format before the execution of the deployment process (BankCon7).

9.6.3 The Automation-Related Data View for HomeFutura

Applying the graphic notation shown in Figure 9.10, the automation-related data view for HomeFutura is presented in Figure 9.12. From this view, the end users can see that they are only required to initiate the deployment process when they want to experience new services and confirm the operation when the selected services are ready to be deployed. More importantly, the end users will be confident in providing this information as the data view shows that the former is based on their own functional requirements and the latter on their own non-functional requirements. With limited involvement in the deployment process, the end user has full control over the selection of new home services and the acceptance of any associated cost (HomeCon10).

As the deployment process for HomeFutura has much higher degree of automation than our previous case, it is more critical that the policies are in place before the deployment process starts as compared to the case of BankFutura.

Table 9.7. The Automation-Related Policy Model for BankFutura

Policy ID	Policy name	Policy Description	Associated service	Controlled by	Type of format
PB01	System requirements	Describes the business processes that must be supported by the environment	Select Unit	Managers	Textual Document
PB02	Unit selection policy: Latest version	Selects the unit with the most recent version among the potential candidates	Resolve Unit	SOA Architects	Formal/SQL Sorting Query
PB03	Unit distribution policy	Defines the rationale behind the environment definition and the quality levels to be sustained by the deployed services	Map Units To Nodes	SOA Architect	Textual Environment design / SLA Document
PB04	Plan Validation Rules	Checks whether the plan is coherent with the current state of the environment	Validate Plan	Environment Administrator	Formal/RETE-based rules + Guidelines Textual Document
PB05	Environment update policy: Minimize runtime impact	Controls the reserved time slows for applying changes, and avoids overlap in the execution of concurrent changes	Schedule Plan Execution	Environment Administrator	Environment Operation Guidelines Textual Document + Environment Calendar

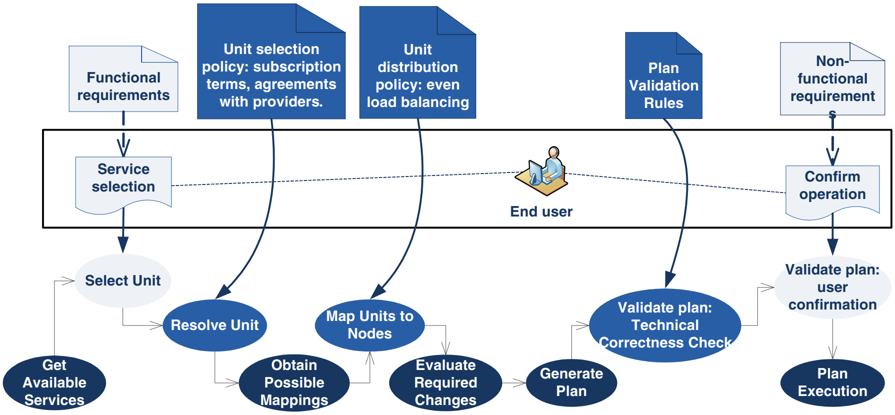

Unit selection policy: subscription terms, agreements with providers.
Unit distribution policy: even load balancing
Functional requirements
Plan Validation Rules
Non-functional requirement s
Service selection
End user
Confirm operation
Select Unit
Resolve Unit
Map Units to Nodes
Validate plan: user confirmation
Validate plan: Technical Correctness Check
Get Available Services
Obtain Possible Mappings
Evaluate Required Changes
Generate Plan
Plan Execution

Fig. 9.12. The Automation-Related Data Flow View for HomeFutura

In BankFutura, there is only one policy-driven service requiring direct access to its associated policy. The other semi-automated services require the associated policies to be well documented enough so that the deployment actors are able to provide the required input. In HomeFutura, as shown in Figure 9.12, there are three policy-driven services, which means that the three associated policies should all be accessible before the deployment process starts (HomeCon7).

Complementary to the data view, the automation-related policy table for HomeFutura presented in Table 9.8 provides detailed information about the policies illustrated in the data view. From this table, the service aggregator can see that he or she is the main role who is in charge of the formalized policies (HomeCon9).

Services available to HomeFutura come from different service providers, after being licensed by the service aggregator. The aggregator knows about their use licenses and selects compatible services with different SLA levels, depending on the user profile. With the aggregator defining policies for solving dependency conflicts based on the user profile, the technical knowledge of the aggregator is transferred to formalized documents that the three deployment services can directly access to. This not only enables these services to be policy-driven automated, but also ensures the alignment between them and technical requirements that are specific to HomeFutura (HomeCon8).

Table 9.8. The Automation-Related Policy Model for HomeFutura

Policy ID	Policy name	Policy Description	Associated service	Controlled by	Type of format
PH01	Functional Requirements	Functionality that must be provided from the HomeFutura platform to the end user	Select Unit	End User	Undocumented knowledge
PH02	Unit selection policy	Selects among the candidate units based on the agreements with service providers and the user subscription terms	Resolve Unit	Service aggregator	Subscription Terms Document, Agreements with providers document
PH03	Unit distribution policy: Even load balancing	Distributes the affected units over the home environment attempting to balance the load on the computing nodes	Map Units To Nodes	Service aggregator	Formal: Linear Programming coding
PH04	Plan Validation Rules	Checks whether the plan is coherent with the current state of the environment and the subscription terms of the end user	Validate Plan: Technical Correctness Check	Service aggregator	Subscription Terms + Formal/ RETE-based rules
PH05	Non-functional requirements	Checks whether the consumption of the selected service complies the non-functional requirements of the end user	Validate Plan: User Confirmation	End user	Undocumented knowledge
PH06	Environment update policy: Instant execution	Instantly applies the plan to the home environment	Schedule Plan Execution	Service aggregator	Formal: code

9.7 The Power of Visualization

Visualization is a common technique to convey abstract information in intuitive ways. Representing information in terms of (a set of) graphics often more easily draws readers' attention and improves understandability, as compared to pieces of text. That is why visualization has been considered as one of the most effective ways for communication. In the same vein, effectively applying the technique of visualization in architecture description improves the communication between stakeholders; in our experience the range of stakeholders involved in services engineering is broader than in the development of traditional applications, rendering the usage of visualization techniques as a key to get effective communication. On the other hand, the usage of diagrams (such as UML) well-known in the field of software architecture, while useful for technical stakeholders, results to be difficult to understand and reason with for non-technical ones (such as users in HomeFutura).

When constructing the service automation views, we consciously design the graphic notation to make the views intuitively understandable and to hold readers attention steady. Some of these graphic notation have been commonly used in architecture description, like using a symbol of person to stand for a human actor, or using a symbol of document to stand for a policy.

In addition to these commonly used notation, we created a set of color schema representing the different degrees of service automation. The motivation behind this schema is that from the human perception point of view, objects with dark color often make one feel heavy in weight (and easy to sink), whereas objects with light color make on feel light (as easy to float). As shown in the views, the degree of automation is graphically rendered by the darkness of the color assigned to each service: the darker the color, the higher s the degree of automation. This representation resembles an iceberg immersed in the sea: only the top (white is the lightest color) is visible (i.e., the user is aware of the service and manually participates in its execution), while the deeper the iceberg is sunk in water, the lesser visible it becomes (i.e., accessible by users, or in other words, increasingly automated).

In the same vein, the graphic notation for the policies inherit the same color schema. As shown in Figure 9.10, the policies for providing guidance for deployment actors are in light color; whereas the policies for assisting the execution of policy-driven automated services are in dark color. As such, from the color of the policies shown in Figure 9.11, and 9.12, the reader can have the perception of the correspondence between the degree of service automation and policies. In addition, in both the degree of service automation view (shown in Figure 9.7, 9.8, and 9.9) and automation-related data view (shown in Figure 9.11, and 9.12, the completely-automated services with dark color "sink" at the bottom, implying that they are loosely coupled with human actors. Whereas the semi-automated services with light color "float" at the top, implying that they are tightly coupled with human actors. As such, these visualization techniques enable the views become self-explaining.

9.8 Observation

By studying the SDCA, and in particular its two industrial cases, we have identified a set of concerns that are particularly relevant to service automation. By constructing views to illustrate the service automation aspect, we gained insight into the way in which service automation has been designed under different contexts. In addition, during the course of this work, we made several observations.

First, we noticed that the degree of service automation is also relevant to the level of service granularity, which was not foreseen in this study. In the design of SBAs, the appropriate level of granularity of services is often regarded of great importance and challenging. The alignment between business and IT is often the (only) main driver for service identification due to the benefits it might bring [8], such as the ease of comprehension of the design, the increase of potential reuse, just to name a few. Since service granularity in nature does not share common interests with service automation, it was not identified as one of the concerns to be addressed by the views. However, in this work we noticed that the service granularity, to certain extent, is also influenced by service automation, which again makes the relevance of service automation to SOA more evident.

In the case of HomeFutura, we noticed that the deployment service Validate Plan was decomposed to two services, one is Validate plan: Technical Correctness Check and another is Validate plan: user confirmation. The main driver for this decomposition is that Validate Plan consists of two types of validation that can be implemented with two different degrees of automation. The technical validation aims to check whether the operations contained in the deployment plan are technically correct for the home environment. This can be done automatically, provided that the policy *Plan Validation Rules* is available. Whereas the user validation aims to get approval from the end users if they agree with the non-functional attributes associated to the select home service. Since the end users are the only ones that have the knowledge on their own preferences and these preferences vary from person to person, it is not feasible to embrace this knowledge into a policy and it has to be controlled directly by the end users. As a result, the deployment service Validate plan: Technical Correctness Check validates the deployment plan from a technical perspective and is designed to be policy-driven automated; whereas the deployment service Validate plan: user confirmation validates the deployment plan from a user perspective and is designed to be semi-automated.

In the case of BankFutura, the deployment service Validate Plan was not decomposed although it also consists of the technical validation and user (system) validation. Similar with HomeFutura, *Plan Validation Rules* can be formalized as a policy that Validate Plan can directly access. The difference lies in the fact that the non-functional requirement of the system is known by the environment administrator and hence can also be formalized as a policy. In this way, Validate Plan can be designed as policy-driven automated, accessing

Plan Validation Rules that consists of both the rules for technical validation and the non-functional requirements.

From these two examples, we can see that the identification of the deployment services or the level of service granularity is not only driven by the business functionalities that they represent, but also influenced by the degree of service automation. Despite the benefits that the business-IT alignment may achieve, an architect sometime would decompose a coarse-grained service into multiple fine-grained services due to different service automation requirements. The result of decomposition might lead to a SBA with higher maintainability and adaptability in terms of service automation but tightly-coupled services and decreased reusability. As such, in the design of SBAs, an SOA architect has to make a trade-off between the alignment with business functionalities and the level of service automation.

The second observation we made is on the applicability of the views to architecture descriptions of SBA in general. As explained by Shull et al. [12], the sources of variability may influence the result, such as the types of projects for which a technique is effective. For this reason, we did an analysis on the variability of the domain and the type of architecture that we studied.

More specifically, the design of the SDCA aims at providing a reference architecture for service configuration and management while the same time focuses on its applicability in industrial domains. We also studied the application of the SDCA in an enterprise domain and a personal domain with completely different characteristics. The difference between these domains contributes to the variability of this study. Although the concerns elicited from the SDCA and its two case studies are somehow different and represent domain-specific interests, addressing these concerns in architecture description demands for similar types of information. When illustrating all these types of information in terms of the same set of graphic pictures and tables (or views), we are able to show that all the concerns identified from each case have been addressed in the corresponding architecture design. As a result, we are confirmed that the views can be applied to three different domains.

However, the concerns that we identified are all related to the SDCA, both its own design and its applications in different industrial domains. The lack of variability in terms of the type of architecture that we studied might threat the validity of the views in illustrating the service automation aspect of SOA in general. For this reason, we plan to replicate the study by analyzing the service automation aspect of various types of SOA in our future work.

9.9 Conclusion

In this chapter we have studied the different degrees of automation suitable for services configuration and deployment on different domains (business and home). While the initial goal was just the development of a system able to perform these functions -a Services Deployment and Configuration Architecture or SDCA- we discovered that the architectural concerns were affected by the specific domain of application it was to be used for; in fact there are several quality attributes that must be covered, but the balance between trust and reliability for example, is specific to the domain.

The key contribution of this paper is the identification of three views that structure and ease elicitation and documentation of stakeholders' concerns. The application of these three views onto the bank and the home domain case studies clearly reflects the differences on the degree of automation for a similar set of basic functions (provided by the services); with a lower degree of automation at the bank domain when compared to the home domain. The notation we have used for the description of views and decisions is simplified with respect to available notations. This allows for a better representation of the concepts involved in the architectural decision making by stakeholders, while remaining intuitive even for non-technical ones. The expression of usually implicit architectural knowledge allowed us getting a hint on the relationship between the degree of automation and the granularity of services. Also, the usage of the same description technique across domains revealed commonalities between them.

The results obtained seem promising, but in order to better capture the wide variability of service automation we plan as future work to to apply the same process to additional Service-Based Applications, as well as applying the approach to several more unconnected domains. This way, it would also be interesting to validate whether different SBAs belonging to the same domain share specific constraints, which affect their decisions on the degree of automation the same way.

References

1. Babar, M.A., Dingsoyr, T., Lago, P., van Vliet, H. (eds.): Software Architecture Knowledge Management: Theory and Practice. Springer, Heidelberg (July 2009)
2. Andrikopoulos, V., Bertoli, P., Bindelli, S., Di Nitto, E., Gehlert, A., Germanovich, L., Kazhamiakin, R., Kounkou, A., Pernici, B., Plebani, P., Weyer, T.: State of the art report on software engineering design knowledge and survey of HCI and contextual knowledge. Technical Report PO-JRA-1.1.1, S-Cube Network of Excellence (2008)
3. ANSI/IEEE. Standard glossary of software engineering terminology, std-729-1991. ANSI/IEEE (1991)
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Reading (2003)
5. Emery, D., Hilliard, R.: Updating ieee 1471: architecture frameworks and other topics. In: WICSA '08: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), pp. 303–306. IEEE Computer Society (2008)

6. Gu, Q., Lago, P.: On service-oriented architectural concerns and viewpoints. In: 8th Working IEEE/IFIP Conference on Software Architecture (WICSA), Cambridge, UK, IEEE (2009)
7. Neumair, B., Hegering, H.-G., Abeck, S.: Integrated management of networked systems: concepts, architectures, and their operational application. Morgan Kaufmann, San Francisco (1998)
8. van den Heuvel, W.-J., Yang, J., Papazoglou, M.P.: Service representation, discovery, and composition for E-marketplaces. In: Batini, C., Giunchiglia, F., Giorgini, P., Mecella, M. (eds.) CoopIS 2001. LNCS, vol. 2172, p. 270. Springer, Heidelberg (2001)
9. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer Magazine, IEEE 36(1), 40–49 (2003)
10. Kruchten, P., Lago, P., van Vliet, H.: Building up and reasoning about architectural knowledge. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 43–58. Springer, Heidelberg (2006)
11. Ruiz, J.L., Dueñas, J.C., Cuadrado, F.: Model-based context-aware deployment of distributed systems. Communications Magazine, IEEE 47(6), 164–171 (2009)
12. Shull, F.J., Carver, J.C., Vegas, S., Juristo, N.: The role of replications in empirical software engineering. Empirical Software Engineering 13(2) (2008)

Correction to: Modeling and Negotiating Service Quality

Salima Benbernou, Ivona Brandic, Cinzia Cappiello, Manuel Carro,
Marco Comuzzi, Attila Kertész, Kyriakos Kritikos, Michael Parkin,
Barbara Pernici, and Pierluigi Plebani

Correction to:
Chapter “Modeling and Negotiating Service Quality” in: M. Papazoglou et al. (Eds.): *Service Research Challenges and Solutions for the Future Internet*, LNCS 6500, https://doi.org/10.1007/978-3-642-17599-2_6

In the original version of this chapter, the affiliation of Barbara Pernici was incorrectly indicated as “Tilburg University, The Netherlands”. It was corrected to “Politecnico di Milano, Italy”

The updated online version of this chapter can be found at
https://doi.org/10.1007/978-3-642-17599-2_6

M. Papazoglou et al. (Eds.): Service Research Challenges and Solutions, LNCS 6500, p. E1, 2010.