

# Software Service Composition in Next Generation Networking Environments

vorgelegt von  
Diplom-Wirtschaftsingenieur  
Carsten Bether  
aus Elsterheide

Von der Fakultät IV – Elektrotechnik und Informatik  
der Technischen Universität Berlin  
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften  
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr.-Ing. Uwe Nestmann (TU-Berlin)  
Berichter: Prof. Dr.-Ing. habil. Sahin Albayrak (TU-Berlin)  
Berichter: Prof. Dr. Nicholas Bambos (Stanford University)  
Berichter: Prof. Dr. rer. nat. habil. Dr. h. c Alexander Schill (TU-Dresden)

Tag der wissenschaftlichen Aussprache: 11.11.2008

Berlin 2008

D 83



# Acknowledgments

I would like to thank the initiators and shepherds of this project, Prof. Sahin Albayrak, Prof. Nicholas Bambos, Dr. Klaus Radermacher and Falk Henkel. Their encouragement and constant feedback made it possible to write this dissertation from multiple locations (Dresden, Berlin, and San Francisco) with the participation of professors at multiple universities (TU-Berlin, TU-Dresden, Stanford University, and Virginia Tech) and of major market players (Deutsche Telekom AG and Bea Systems Inc.). I further deeply appreciate the contributions of Prof. Alexander Schill and Prof. Athman Bouguettaya, who not only shaped my understanding of service-oriented software systems, but also helped me create a systematic way to approach, structure and formulate scientific work.

I would also like to express my appreciation to Ursula Wahls, who helped me obtain a research fellowship at Deutsche Telekom AG. Franziska Vogel and Katrin Zobel-Lu supported me with long hours of corrections to improve my English grammar and writing. Sylvia Allen gave the thesis its final polishing and structure in endless discussions about specific aspects of the German and English languages.

Last, but definitely not least, I would like to thank my family, my mother who persuaded me to attend a university, my father who taught me to appreciate the simple things in life and my brother, one of the strongest pillars I have.



# Preface

After a decade of rapid changes caused by digitalization, privatization and globalization, the telecommunications market is steadily continuing to diversify. Situated between customer demands and the technological revolution, today's service providers must meet many challenges. Four trends speed up the evolution of telecommunication:

- *Customer Expectations*: Increasing customer demand and changing customer expectations of telecommunication services
- *Convergence*: The independent use of telecommunication services, without regard to devices, networks and location [Mil02]
- *IP centrality*: The tendency to handle network transport via the well-known packet-based Internet Protocol (IP)
- *Software focus*: The shift from hardware solutions to a focus on the use of software components

Present providers maintain infrastructures which have grown through technological evolution into huge, heterogeneous and complex systems. Following the design strategy of one network per service, silo architectures have emerged as redundant backbones of information transmission capabilities. At the same time, the community of users expects converged services for less cost.

The answer to this challenge facing telecommunication providers may exist in the formation of a mediating abstract software layer, a layer that organizes telecommunication services to be agile, flexible and quickly responsive to customer needs while still integrating legacy transport facilities. To design and implement such an approach, the software service paradigm can deliver

a constellation of concepts, values, perceptions, and practices shared by the software engineering community.

Based on this observable complementarity between telecommunication and software services, this thesis demonstrates that software can harness agility and flexibility for the telecommunication domain by using the composition of service contracts and implementations, while remaining semantically rich and loosely coupled.

**Keywords:** Next Generation Network, Service Oriented Architecture, Service Composition, Telecommunication Service Framework

# Zusammenfassung

Gegenwaertige Veraenderungen in der Telekommunikationsindustrie werden durch verringerte Markteintrittsbarrieren, generell verfuegbare Flatrate Zugaeuge sowie frei verfuegbare, ausgereifte Softwaretechnologien vorangetrieben. Diese Veraenderungen inspirieren derzeit sowohl Marktteilnehmer als auch Forscher dazu, nach neuen Moeglichkeiten zur flexibleren Bereitstellung von Telekommunikationsdiensten zu suchen. Eine im Rahmen dieser Arbeit durchgefuehrte Umfrage zeigt, dass Inadequitaet, Inflexibilitaet und zu hohe Kosten die am haeufigsten genannten Gruende fuer die Unzufriedenheit mit vorhandenen Infrastrukturen sind. Gegenwaertige Trends zeigen, dass sich Telekommunikationsinfrastrukturen mehr und mehr zu einer IP-basierten Zukunftsarchitektur entwickeln, dem Next Generation Network (NGN). Gleichzeitig zeichnen sich im Software Engineering Tendenzen ab, flexible Softwareschichten mittels einer Service-Orientierten Architektur (SOA) zu realisieren. Die vorliegende Dissertation eroertert die Komplementaritaet beider Konzepte, um die genannten Herausforderungen derzeitiger Infrastrukturen aufzuweichen und moegliche Loesungen speziell fuer die benoetigte Agilitaet und Flexibilitaet bei der Bereitstellung von Telekommunikationsdiensten aufzuzeigen.



# Abstract

Recent and continuing changes in the telecommunication market, which have been driven by lowered barriers to market, affordable flatrate broadband access, and freely available, mature software environments, have inspired researchers and industry to search for new capabilities to provide telecom services in a more flexible manner. A survey conducted in the context of this thesis found that inadequacy, inflexibility and immense costs are the the issues most mentioned in association with existing infrastructures. Current tendencies show that telecommunication is moving toward a generic IP-based architecture, the Next Generation Network (NGN). At the same time, software engineering tends to implement flexible software solutions using Service-Oriented Architectures (SOA). Bringing telecommunication and software engineering together, this dissertation uses the complementarity of these two concepts to resolve known problems and propose solutions that emphasize the flexibility and agility needed in the provision of telecommunication services.

# Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>1</b>	<b>Basics</b>	<b>3</b>
1.1	Synopsis . . . . .	3
1.2	Domain Definition . . . . .	3
1.3	Problem Statements . . . . .	8
1.4	Research Questions . . . . .	9
1.5	Objectives and Contributions . . . . .	11
1.5.1	Objectives . . . . .	11
1.5.2	Contributions . . . . .	12
1.6	Solution Approach . . . . .	12
1.6.1	Examining Underlying Concepts . . . . .	12
1.6.2	Evaluating Related Work . . . . .	13
1.6.3	Postulates . . . . .	14
1.6.4	The SOTF Framework . . . . .	14
1.6.5	Prototype Scenario . . . . .	14
1.7	Structure of the Thesis . . . . .	15
<b>2</b>	<b>Telecommunication Networks</b>	<b>17</b>
2.1	Synopsis . . . . .	17

2.2	Definition of Telecommunication Network . . . . .	18
2.3	Evolution of Networks . . . . .	18
2.3.1	Network Convergence . . . . .	19
2.3.2	Service-Specific Networks . . . . .	20
2.3.3	Service-Integrated Networks . . . . .	20
2.3.4	Intelligent Networks . . . . .	21
2.3.5	Distributed, Overlay Networks . . . . .	23
2.4	Next Generation Networks . . . . .	25
2.4.1	Definition of NGN . . . . .	25
2.4.2	NGN Requirements . . . . .	26
2.4.3	Development of NGN . . . . .	27
2.4.4	Reference Model . . . . .	28
2.5	Quality of Service . . . . .	32
2.5.1	QoS Parameters . . . . .	33
2.6	Drivers and Requirements of the NGN Concept . . . . .	34
2.7	Summary . . . . .	34
<b>3</b>	<b>Services</b>	<b>37</b>
3.1	Synopsis . . . . .	37
3.2	Evolution of Services . . . . .	38
3.2.1	Business Services . . . . .	38
3.2.2	Telecommunication Services . . . . .	39
3.2.3	Software Services . . . . .	41
3.3	Service Paradigm Entities and Principles . . . . .	43
3.4	Service-Oriented Architecture . . . . .	45
3.4.1	Definition of SOA . . . . .	46
3.4.2	SOA's Entities and Activities . . . . .	47
3.4.3	Implications of the SOA Architectural Style . . . . .	49

3.5	Lifecycle and Scope . . . . .	53
3.5.1	Lifecycle . . . . .	53
3.5.2	Distribution Scope . . . . .	54
3.6	Summary . . . . .	55
<b>II</b>	<b>Composition Model and Related Work</b>	<b>57</b>
<b>4</b>	<b>Service Composition</b>	<b>59</b>
4.1	Synopsis . . . . .	59
4.2	Definition of Composition . . . . .	60
4.2.1	An Overview of Service Composition . . . . .	61
4.3	A Composition Machine . . . . .	63
4.4	Elements . . . . .	65
4.5	Behavior . . . . .	65
4.5.1	Conditions . . . . .	66
4.5.2	Behavior Model . . . . .	68
4.6	Patterns . . . . .	69
4.6.1	Sequence . . . . .	69
4.6.2	Parallelism . . . . .	71
4.6.3	Exclusive Decision . . . . .	72
4.6.4	Multiple Decision . . . . .	74
4.6.5	Looping . . . . .	76
4.7	Non-Functional Properties . . . . .	77
4.7.1	State Management . . . . .	78
4.7.2	Transaction Handling . . . . .	79
4.7.3	Security and Trust . . . . .	80
4.7.4	Quality Measurement . . . . .	81
4.7.5	Errors . . . . .	82

4.7.6	Invocation . . . . .	83
4.7.7	Communication . . . . .	84
4.7.8	Summary of Composition Requirements and Implications	85
4.8	Syntax Formalization of Service Composition . . . . .	87
4.9	Summary . . . . .	89
<b>5</b>	<b>Research Scope and Related Work</b>	<b>91</b>
5.1	Synopsis . . . . .	91
5.2	Research Scope . . . . .	92
5.2.1	Domain Boundaries . . . . .	92
5.2.2	Composition Focus . . . . .	95
5.3	Composition Essentials . . . . .	97
5.3.1	Formalism-Based Approaches . . . . .	97
5.3.2	Software Engineering Entities . . . . .	99
5.3.3	Research Relevance of Formalism Approaches . . . . .	100
5.3.4	Research Relevance of Software Engineering Approaches	102
5.4	Telecommunication Network Frameworks . . . . .	104
5.4.1	IP Multimedia Subsystem (IMS) . . . . .	105
5.4.2	Research Relevance . . . . .	110
5.5	Telecommunication Software Frameworks . . . . .	111
5.5.1	Open Mobile Alliance . . . . .	112
5.5.2	Open System Architecture (OSA) . . . . .	114
5.5.3	Research Relevance . . . . .	115
5.6	Software Frameworks . . . . .	116
5.6.1	Web Services . . . . .	116
5.6.2	Ontologies . . . . .	121
5.6.3	Agents . . . . .	125
5.7	Comparative Analysis . . . . .	126
5.8	Summary . . . . .	127

<b>III</b>	<b>Service Frameworking in the NGN</b>	<b>129</b>
<b>6</b>	<b>Services in Next Generation Networks</b>	<b>131</b>
6.1	Synopsis . . . . .	131
6.2	Terminology . . . . .	133
6.3	A Telecommunication Service Ontology . . . . .	134
6.3.1	Semantic Perspectives . . . . .	134
6.4	Service Cluster Semantics . . . . .	138
6.4.1	Community Services . . . . .	139
6.4.2	Content Services . . . . .	143
6.4.3	Communication Services . . . . .	145
6.4.4	Control Services . . . . .	148
6.5	Summary . . . . .	150
<b>7</b>	<b>SOTF: A Service-Oriented Telecommunication Framework</b>	<b>153</b>
7.1	Synopsis . . . . .	153
7.2	The TOGAF Approach . . . . .	155
7.3	A Service-Oriented Telecommunication Framework (SOTF) . . . . .	157
7.3.1	Framework and Principles . . . . .	158
7.4	Architectural Vision . . . . .	159
7.4.1	General Vision . . . . .	159
7.4.2	Composition Vision . . . . .	160
7.4.3	General Principles and Requirements . . . . .	162
7.5	Business Architecture . . . . .	162
7.5.1	Input and Requirements . . . . .	163
7.5.2	Approach . . . . .	165
7.5.3	Result . . . . .	166
7.6	Data Architecture . . . . .	174
7.6.1	Input and Requirements . . . . .	175

7.6.2	Approach	176
7.6.3	Result	177
7.7	Application Architecture	179
7.7.1	Input and Requirements	180
7.7.2	Approach	180
7.7.3	Result	182
7.8	Composition, Composeability and Matchability	185
7.8.1	Degree of Composeability	185
7.8.2	SOTF Service Composition	186
7.9	Summary	187

## **IV Applied Service Composition in the NGN 191**

<b>8</b>	<b>Prototype Implementation</b>	<b>193</b>
8.1	Synopsis	193
8.2	Prototype Specification	194
8.2.1	Project Background	194
8.2.2	MMCCS Software Components	195
8.3	Prototype Scenario	203
8.3.1	Scenario Introduction	203
8.3.2	Scenario Assumptions	204
8.3.3	Scenario Use Cases	205
8.4	ConnectME, Service Division	206
8.4.1	Overview	207
8.4.2	Use Cases	207
8.5	ConnectME, Application Division	223
8.5.1	Overview	224
8.5.2	Use Cases	224
8.6	Summary	230

<b>9</b>	<b>Prototype Validation and Discussion</b>	<b>231</b>
9.1	Synopsis . . . . .	231
9.2	Non-functional Service Heterogeneity . . . . .	232
9.2.1	State Management . . . . .	233
9.2.2	Transaction Handling . . . . .	234
9.2.3	Communication . . . . .	236
9.3	Inter-service, Circular, and Automatic Invocation . . . . .	237
9.3.1	Inter-service Invocation and Circularity . . . . .	237
9.3.2	Automatic Invocation . . . . .	240
9.4	Quality Awareness . . . . .	242
9.5	Prototype Enhancements . . . . .	248
9.6	Summary . . . . .	250
<b>V</b>	<b>Conclusion</b>	<b>251</b>
<b>10</b>	<b>Contributions, Future Research and Outlook</b>	<b>253</b>
10.1	Synopsis . . . . .	253
10.2	Answers to Research Questions . . . . .	254
10.3	Thesis Contributions . . . . .	257
10.4	Suggestions for Future Research . . . . .	259
10.5	Outlook . . . . .	261
<b>VI</b>	<b>Appendices</b>	<b>263</b>
<b>A</b>	<b>A. Appendix of Theoretical Background</b>	<b>265</b>
A.1	BNF Notation . . . . .	265
<b>B</b>	<b>B. Appendix of SOTF Framework Conceptualization</b>	<b>267</b>
B.1	Specification of Application Architecture Components . . . . .	267

<b>C C. Appendix of Prototype Implementation</b>	<b>271</b>
C.1 Definition of NGN Service Types . . . . .	271
<b>Glossary</b>	<b>275</b>
<b>Bibliography</b>	<b>300</b>

# List of Figures

1.1	User Expectations of Telecommunication Services [uSSG06] . . .	5
1.2	Research Approach of the Thesis . . . . .	15
1.3	Structure of the Thesis . . . . .	15
2.1	Network Evolution Milestones . . . . .	19
2.2	IN Conceptual model . . . . .	23
2.3	Reference Model of NGN . . . . .	29
3.1	Business Service Entities and Activities . . . . .	39
3.2	Telecommunication Service Entities and Activities . . . . .	41
3.3	Software Service Entities and Activities . . . . .	44
3.4	SOA Service Lifecycle . . . . .	54
3.5	SOA Distribution Scopes . . . . .	55
4.1	Orchestration and Composition . . . . .	62
4.2	A Composition Machine . . . . .	63
4.3	Composition Behavior . . . . .	68
4.4	Structure of Sequence Composition Pattern . . . . .	70
4.5	Structure of Parallel Composition Pattern . . . . .	71
4.6	Structure of Exclusive Decision Composition Pattern . . . . .	73
4.7	Structure of Parallel Composition Pattern . . . . .	75

4.8	Structure of Loop Composition Pattern . . . . .	76
4.9	Syntax Formalization of Service Composition . . . . .	87
4.10	Syntax Formalization of a Composition Member Service . . . . .	88
4.11	Summary of Notation for Composition Behavior . . . . .	89
5.1	Research Scope . . . . .	93
5.2	Basic Internet Protocol (IP) Stack . . . . .	102
5.3	IP Multimedia Subsystem (IMS) . . . . .	106
5.4	Open Mobile Alliance Service Environment . . . . .	113
6.1	A Telecommunication Service Ontology . . . . .	135
6.2	Semantic Specification of a Service Cluster . . . . .	139
6.3	Semantic Specification of the Community Service Cluster . . . . .	143
6.4	Semantic Specification of the Content Service Cluster . . . . .	146
6.5	Semantic Specification of the Communication Service Cluster . . . . .	148
6.6	Semantic Specification of the Control Service Cluster . . . . .	150
6.7	NGN Service Cluster: Cohesion and Clauses . . . . .	151
7.1	TOGAF 8.1.1 Architecture Development Cycle (ADM) . . . . .	156
7.2	SOTF: Architectural Vision . . . . .	161
7.3	SOTF: Principles and Requirements . . . . .	163
7.4	SOTF Business Architecture, Services, Clusters and Operations . . . . .	173
7.5	SOTF Data Architecture, Basic Service Data Entities . . . . .	177
7.6	SOTF Data Architecture, Service Data Entities . . . . .	179
7.7	SOTF Technology Architecture, Software Components . . . . .	184
8.1	Components of the MMCCS architecture . . . . .	197
8.2	Scenario Use Cases . . . . .	206
8.3	Sequence Diagram of Lookup Service Use Case . . . . .	226
8.4	Sequence Diagram of Invoke Service Use Case . . . . .	229

9.1	Quality of Service Lookup by Service Specification Type . . .	245
9.2	Quality of Service Invocation by Service Specification Type . .	247
9.3	Quality of Service Processing by Service Specification Type . .	248



# List of Tables

2.1	Evolution of Networks	24
2.2	Drivers and Requirements of the NGN Concept	36
3.1	Evolution of the Software Engineering Paradigm	43
3.2	Design Principles and Implications of the SOA Architectural Style	53
4.1	Composition Environment Requirements and Issues	86
5.1	Efficient Use of Service Composition in Telecom Infrastructures	96
5.2	Comparative Analysis of Related Work	126
6.1	Quality Model of Software Services	137
7.1	Community Cluster Business Operations	166
7.2	Content Cluster Business Operations	167
7.3	Communication Cluster Business Operations	169
7.4	Control Cluster Business Operations	170
7.5	Composition Cluster Business Operations	171
7.6	System Cluster Business Operations	172
7.7	SOTF Component Specification: Design and Execution	189
7.8	SOTF Component Specification: Control and Persistence	190



# Part I

## Introduction



# Chapter 1

## Basics

*“Mozart, you are not the only composer...”*  
*Count von Strack, Amadeus [1984]*

### 1.1 Synopsis

---

This chapter describes the structure of the thesis in detail. It broadly defines the telecommunication domain, exposes questions to be researched, and proposes a potential solution to the crippling rigidity of current telecommunication infrastructures. The chapter closes with a structural overview of the thesis.

### 1.2 Domain Definition

---

We can observe three primary interrelated perspectives of telecommunication: the economic, the operational, and the technical.

## Economic Perspective

The telecommunication domain has undergone rapid changes over the last few years. Economically, telecommunication now earns three percent of the world's revenue, and the market volume is increasing. The market fluctuates constantly. Players in the market face stiff competition.

Identifiable roles of those players, in reference to their service offerings in the new, evolving telecommunication market, are listed below.

- The *Sales Provider* connects services to the user but does not have an infrastructure.<sup>1</sup>
- The *Service Provider* connects services to the user using its own infrastructure.
- The *Network Provider* offers pure network transport capacities.

Depending on their strategy, providers can take on one or more roles. For example, Deutsche Telekom AG functions as a network, service and sales provider, while Ebay/Skype Inc. acts as both service provider and sales provider. The current market situation requires providers to engage in a continuous search for new business models and services while maintaining existing services.

Users of telecom services have various expectations which must be satisfied by telecom providers: [uSSG06]

- *Individuality*: desire to stand apart, status
- *Convenience*: simplicity, ease of use, accessibility
- *Information*: desire to stay informed, desire to inform
- *Control*: security, privacy, safety
- *Belonging*: desire to belong to a group, peer groups, communities
- *Productivity*: speed, time efficiency, general efficiency experienced while communicating

---

<sup>1</sup>The infrastructure is the equipment a provider maintains to operate its network. A network is the combination of links and nodes that enable the transmission of information from one part of the network to another.

- *Pleasure*: escapism, leisure, infotainment, fun
- *Independence*: self-determination

Figure 1.1 demonstrates how user expectations changed over time.

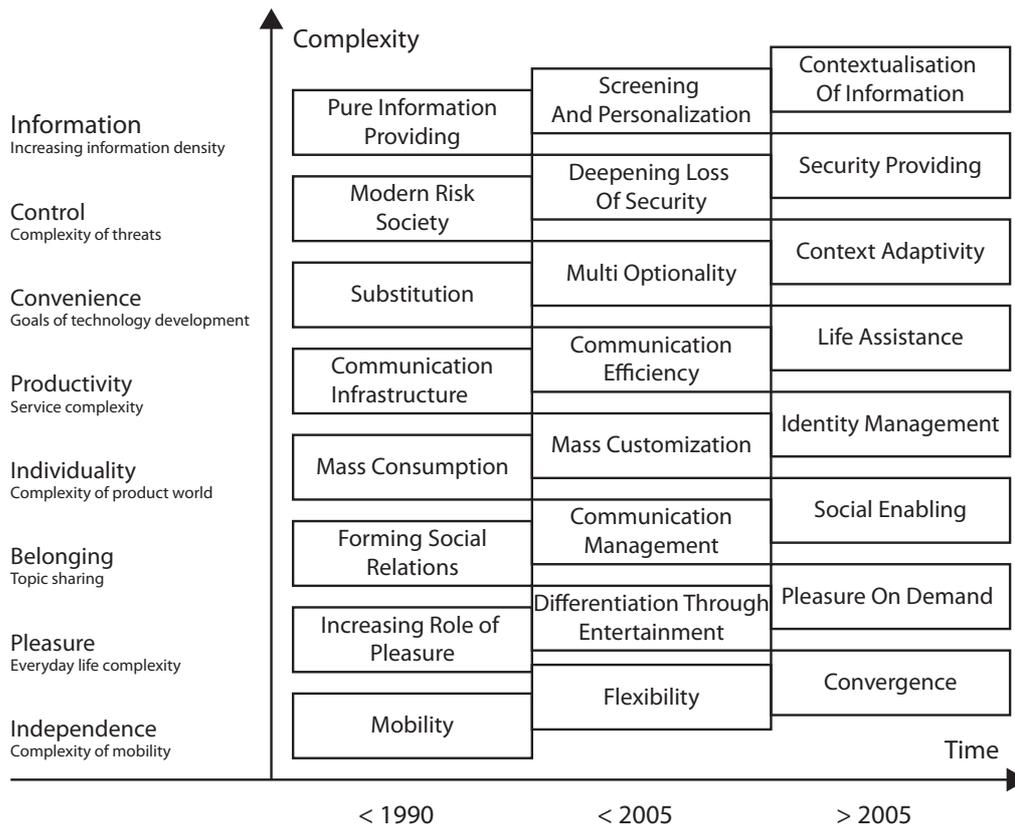


Figure 1.1: User Expectations of Telecommunication Services [uSSG06]

”Everybody, anytime and everywhere” [Tec99] is the motto of today’s telecom services. The proliferation of provider roles and user expectations raises interesting operational and technological challenges for telecom service providers.

## Operational Perspective

Apart from the economic aspects of the telecom domain, a telecommunication network must be operated. Typical operational challenges are customer provisioning and service activation, Quality of Service (QoS) through Service Level Agreements (SLA) and security. [Bag04] In general, these operations have been executed through hardware and software architectures that are home-grown, proprietary and rigid. Providers have both maintained and upgraded huge infrastructures at the same time. In addition, telecommunication companies are burdened with heterogeneous environments which stand in the way of efficient operation.

To succeed in the current economic environment, telecom providers need to establish agility: their systems must be able to adapt new services quickly, and most importantly, without sacrificing reliability. In place of the current pattern of large, rigid infrastructures, telecom systems should operate as a collection of interconnected functions. There is a need for infrastructure transparency, which would give providers the freedom to decide whether to build new functions or update existing ones, and to decide which of these functions are part of their business core competencies, providing opportunities for them to distinguish themselves from competitors, and which can be effectively outsourced. [NB05]

## Technical Perspective

A third aspect of telecommunications is technology management—the management of a network’s technical components and their realization and interconnection.

With the rapid changes in technology and in the marketplace, providers must constantly consider the possibilities of new tools and new uses for tools. User devices (PCs, mobile phones, PDAs, TVs), networks (GPRS, UMTS, CDMA), and software frameworks (Java EE, .NET) are only a few current examples of technologies that can be evaluated and possibly integrated. However, a decision to use a technology requires a huge investment and cannot be rolled back without additional costs.

Telecommunication services must be designed, implemented, controlled and delivered while remaining reliable, robust and secure. [Bag04] Providers must respond quickly to developing business situations and minimize costs through reduction of time to market. The technological infrastructure should reflect and implement that requirement as well.

In responding to the issues outlined above, it is crucial to leverage agility throughout the telecom organization. Agility appears as a critical challenge in the current state of telecom technology, where huge legacy landscapes dominate. The ideal scenario would be a service architecture that can support multiple types of services and management applications over multiple types of transport.

There is currently no standardized process to design and implement the agile, flexible architectures that the emerging telecommunication industry requires. In general, two approaches are being used: the current telecommunication approach, driven by telecom standardization organizations, and the software engineering approach, driven by software framework distributors and software vendors. Unfortunately, the approaches are not in sync, and as a consequence, a broad and heterogeneous field of architectures, standards, and implementation methods exist. This technological heterogeneity presents major obstacles not only to efficient operation but also to the development of larger, yet still efficient, telecom networks.

**Definition of Telecommunication** *Telecommunication<sup>2</sup> is the electronic transmission of information over significant distances.* [Uni]

The word *Telecommunication* is a synthesis of the two elements *tele* and *communication*. *Communication* is an exchange of information. The prefix *tele*, adding the aspect of long distance transmission, raises a technological challenge.

Telecommunication serves as a business market, encompasses some general roles and interactions, and has a considerable impact on the development of technology.

**Definition of Telecom Services** *The manifestation of telecommunication in the marketplace is a telecommunication (telecom) service. A telecom service is a packaged set of capabilities that is perceived by a human user when interacting with a telecommunications network or a service provider and for which separate billing can be arranged.* [Con97]

In this thesis the term *service* will cover both business and technical aspects, further illustrated in Chapter 3. Chapter 6 gives an elaborated overview of services in telecommunication networks.

---

<sup>2</sup>The abbreviation *telecom* is used interchangeably with *telecommunication*.

---

## 1.3 Problem Statements

---

In general, this thesis is a response to the existing mismatch, indicated by the challenges discussed in the previous section, of the approaches of IP-based telecommunication networks and software engineering.

**Problem Statement for the Telecommunication Domain** The telecommunication domain is huge, complex and (so far) semantically undefined. There is no common understanding of how to structure, organize and share the large amount of knowledge about services that exists in the domain. There is, further, no clarity as to which services must be provided in a telecommunication infrastructure and which are optional. Considering the current tendency to implement such services using software, new ways must be found to model and define services in a flexible manner, so that the business, data and technology perspectives can transform one another, and so that participating stakeholders have no need to be aware of which services are provided using which software components. There is an unfilled need for a rigorous analysis of existing scientific and industry work with the aim of building an integrated, implementable model of interworking services. There is, in addition, a need for a unified methodology to transform such a model into computable software artifacts that are extensible and flexible to use.

**Problem Statement for the Software Service Paradigm** Relatively few researchers have studied the potential of an SOA approach within an NGN infrastructure. Existing discussions are held in a limited context, driven either by technology (such as Web Services or Java EE) or by software components (such as runtime environment, service contract, or message bus). Framework approaches, which would include domain specifics as a bridge between building blocks and encompassing environments, are missing. In particular, the identification and semantic clustering of telecom services have yet to be elaborated. Semantic descriptions that would provide standardized sets of services have not been established. The industry approach has been the introduction of Service Delivery Platforms (SDP), which unfortunately do not provide either an architecture or a service design methodology with standardized interfaces.

**Problem Statement for Service Composition** Composition solutions, following the service paradigm principles, are either algebraically formulated (as in Finite State Approach and  $\pi$ -Calculus) or graphically modeled (as in Colored Petri Nets and Hierarchical Task Network (HTN) Planning) as standalone domain-independent systems. However, service composition, especially in the field of telecommunication, is particularly challenging because telecom services are coarse-grained, real-time, and subject to frequent changes while requiring the maintenance of high reliability. [CKMRM03] The seamless integration of synchronous and asynchronous services in a composition scenario is in a very early stage. [Sch05] Emerging technologies such as the Web Service framework seem to fit the basic telecom business interface, but they are not designed for composition in a high-performance real-time telecommunication scenario. There is an unrealized potential for the discussion and design of composition solutions within the requirements of the telecom domain.

## 1.4 Research Questions

---

The main goal of this thesis is to deliver a satisfactory answer to the question, "How can the software service paradigm supplement the emerging IP-based telecommunication infrastructures to improve agility through reuse?" In that context, the following research questions will be addressed:

### **Conceptual Complementarity of Telecommunication Networks and the Software Service Paradigm**

- In a discussion of software services in a telecommunication network, which definition context, entities and activities should be reflected?
- What should a software service lifecycle in a telecom network look like? What are the specifics of services in the telecom domain?
- What role does composition play in the service lifecycle?

### **Telecommunication Service Framework Development**

- Which services must be provided and which are optional, in setting up a reliable business and utility service cluster for telecommunication?
- What data structure will serve to enable both manual and dynamic composition of services?
- Which software components must be assembled, to compute those services using that data structure?

### **Implementation of Service Composition in the Telecommunication Domain**

- How can we integrate the research findings with available standards and technologies to enable the universal computation and composition of services in a software architecture?
- How can we overcome the heterogeneity of telecommunication services (as manifested in stateful vs. stateless invocation, transactive vs. non-transactive processing, diversity in quality requirements, and mixed synchronous and asynchronous communication handling) in such a universally applied service lifecycle?
- How can we dynamize the creation of compositions and automate the invocation of services in a composition?

### **Conclusion**

- How valid is the solution introduced in this thesis?
- What are its boundaries, side effects and obstacles?

---

## 1.5 Objectives and Contributions

---

### 1.5.1 Objectives

The goal of this thesis is to develop a technically neutral service-oriented framework, compliant with the principles of the service paradigm, which covers the specific requirements of telecommunication services, and which promotes reliable service reuse via composition. The framework description will define components on the application, service, and control layers of the network and explain the relations between them.<sup>3</sup> To develop an adequate approach, the following sub-objectives are deducible:

- Conducting an analysis of how well the service paradigm and the SOA architectural style adapt to an IP-based telecommunication architecture. This investigation will require an independent analysis of both constructs. Based on this analysis, we will compile a concept-overlapping requirement evaluation, to identify the specific, non-functional, and environmental properties of telecommunication services.
- Development of a framework, designed as a system architecture, reflecting the identified adaptivity and service composition capabilities. This framework will consist of a static system component structure and will demonstrate system behavior within the service lifecycle. Pillars of the development of this framework will be the identification of an architectural vision, the engineering of all identified requirements and the derivation of three architectural descriptions: business, information and technology.
- Prototyping of the framework for validation and demonstration purposes. The prototype will include a scenario-based implementation of framework parts and an embedded composition solution. It will highlight the usable implementation standards and illustrate efficient design idioms and patterns.

---

<sup>3</sup>Such a project is not without precedent. The project Multi-Modal Communication and Collaboration Services (MMCCS) is a spin-off from the DTAG divisions of T-Laboratories, T-Systems and T-Com. The MMCCS prototyped a seamless telecom infrastructure on a Web Service framework.

## 1.5.2 Contributions

To answer the research questions that have been posed, and achieve the objectives listed above, this thesis will develop two core concepts:

- A *Service-Oriented Telecommunication Framework* (SOTF), a software engineering model for designing a telecommunication infrastructure. It will need to be integrated using software components, technology standards, and design methodologies. This will be discussed in the conclusion of the thesis, in relation to the scenario developed in the prototype.
- A *composition prototype* that enables the manual and dynamic assembling of telecommunication services. The prototype finally represents a high level programming model for telecommunication service composition.

## 1.6 Solution Approach

---

### 1.6.1 Examining Underlying Concepts

Our approach to the telecom technology problems stated above starts with the investigation of three underlying concepts. We will analyze the characteristics of each of these concepts independently, and we will look back at their histories to gain an understanding of their natures and their potential intersection.

#### NGN

Approaching from the evolution of telecommunication networks, we present *Next Generation Networking* (NGN) as a core overlay network infrastructure able to carry any type of IP-based service. As a logical extension of concepts like Intelligent Networking (IN), NGN includes facilities and requirements which serve as input for the design of the framework. A reference model is proposed to structure NGN functionally and analyze the dependencies between the layers.

## SOA

Analogously, *Service-Oriented Architecture* (SOA) is introduced in the context of its historical evolution. Service orientation resulted in the formation of the software service design paradigm, and it is commonly regarded as an evolutionary step up, building on other paradigms such as components and object orientation. This relationship between the paradigms facilitates the identification of the principles of the SOA service paradigm and its implications, as reflected in the SOA architectural style.

### Service Composition

Within the service lifecycle, composition acts as an activity, enabling service creation through assembly and disassembly of atomic and composed services. Composition can be demonstrated by member services interacting on a machine-processable composition path in order to achieve a service consumer goal. Composition manifests certain characteristic patterns of behavior and non-functional properties, which will serve, here, as the foundation for a basic composition model, the *composition machine*. This model will explain the relationship between the primary entities that interact during composition: the service, the service composition and the surrounding software system.

### 1.6.2 Evaluating Related Work

Once we have introduced the three pillars of the research work, NGN, SOA and service composition, we will go on to evaluate the relevant current scientific and industry-based work. A matrix, or comparison table, of the characteristics of these pillars will be used to compare the strengths and weaknesses of existing approaches. Since the analysis is focused on existing frameworks, the comparison also assesses the potential for integration of solutions or parts of solutions into the framework development.

### 1.6.3 Postulates

After identifying the characteristics of the underlying concepts and the gaps in existing approaches, the framework development presents the following postulates:

- NGN is an telecommunication network architecture with an inherent service character formed on the basis of a concrete catalogue of design principles. It accounts for a set of core functionalities and is implemented using hardware and software components.
- The service paradigm and SOA are software design mechanisms. SOA gathers implementational rule sets arising from the fundamental principals of the service paradigm.
- Service composition is an activity of the service lifecycle, which behaves according to specific patterns and includes specific functional and non-functional properties.

### 1.6.4 The SOTF Framework

Our framework approach incorporates three perspectives: business, informational and technical. The outcome of this approach is threefold: a business architecture to account for the main interacting business services, an information architecture to structure underlying data and components and a technical architecture to specify the technical implementation details. We will use the term *Service-Oriented Telecommunication Framework* (SOTF) to describe the resulting structures.

### 1.6.5 Prototype Scenario

Our structures, once created, will need to be verified and tested. To that end, the thesis develops a prototype scenario of a seamless, roamable conference, enabling audio, video and data communication among several participants. The scenario assumes the participants to be highly mobile, which requires switching between devices and services during the conference without losing sessions or user state information. The special focus of this scenario lies in the specification of services, using the developed syntax, its compilation into a service contract and semi-automatic to automatic composition.

The service implementation of the scenario will be based on commonly accepted industry standards, which include the work of organizations such as 3GPP, 3GPP2, OMA, ParlayGroup, Java Community Process (JCP) and the Internet Engineering Task Force (IETF). The framework and the prototype will be documented using OMG’s Unified Modeling Language (UML).

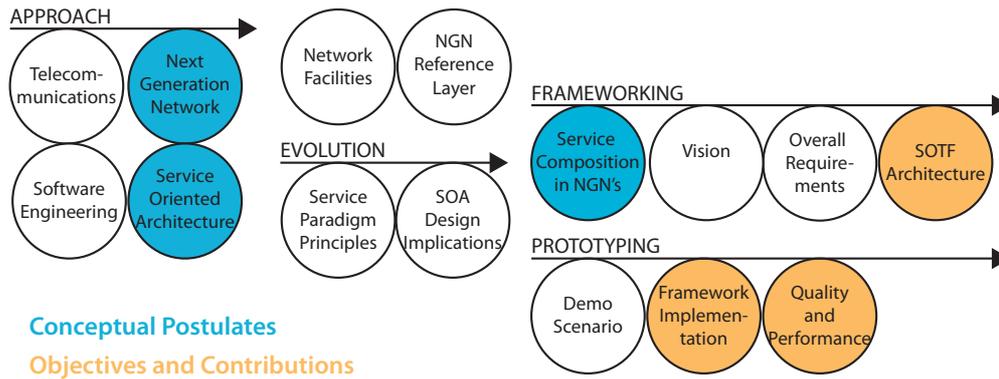


Figure 1.2: Research Approach of the Thesis

## 1.7 Structure of the Thesis

The thesis is organized into five major parts and an appendix:

	I: Introduction	II: Composition and Related Work	III: Frameworking	IV: Applied Service Composition	V: Conclusion
Dedication	1) Basics	4) Service Composition	6) Services in Next Generation Networks	8) Prototype Implementation	10) Contributions, Future Research and Outlook
Statement of Originality					
Preface	2) Telecommunication Networks	5) Research Scope and Related Work	7) SOTF: A Service-Oriented Telecommunication Framework	9) Prototype Validation and Discussion	
Abstract					
Listings (Contents, Figures, Tables)	3) Services				
Appendices					

Figure 1.3: Structure of the Thesis

Part **I** introduces the theoretical work and places it within the fields of telecommunication networks (Chapter **2**) and service engineering (Chapter

3). The characteristics and requirements of the two fields are deduced from their evolutionary milestones. These chapters then introduce a new, complementary step: the intersection of IP-based networks and software service engineering. This concept is based on three entities (consumer, provider and broker) interacting in a lifecycle of dependent service activities, from specification to binding of the service to the consumer.

Part II introduces composition as an activity of the service lifecycle and elaborates the behavioral patterns and properties of composed services (Chapter 4). Related work, which deals with the framework-based service composition of telecommunication services, is investigated in Chapter 5, moving from abstract formalisms to approaches driven by industry.

Part III develops a semantic understanding of NGN Services in Chapter 6. The chapter proposes an ontology as a semantic specification of the core functionality of the telecommunication domain. Based on the cluster structure of this ontology, Chapter 7 constructs a framework that encompasses business, informational and technological considerations.

Part IV specifies a service scenario which consists of interacting domain and utility services. The composing possibilities are discussed in a business and technical context in Chapter 8. Strengths and weaknesses are pointed out and quality issues are considered, including an analytical and experimental performance investigation in Chapter 9.

In Part V, Chapter 10 summarizes the achievements of this thesis. Future research and a future outlook are discussed. The chapter lays out the potential effects of the proposed research approach.

Part VI contains the evaluating work, the software code and documentation of the implementation.

# Chapter 2

## Telecommunication Networks

*“Communication. It’s the first thing we really learn in life. Funny thing is, once we grow up, learn our words and really start talking the harder it becomes to know what to say. Or how to ask for what we really need. ”*  
Dr. Meredith Grey, *Grey’s Anatomy* [2005]

### 2.1 Synopsis

---

Networks are the foundation of all telecommunication, the ecosystem in which telecom exists. The chapter offers a detailed description of the latest step in that evolution, *Next Generation Networking* (NGN). The purpose of the chapter is to place this work in the context of network evolution and NGN. The characteristics of the milestones in this evolution have a direct impact on the framework development in Chapter 7.

Historically, processing and design have been increasingly separated from one another, and so have logic and transport. These separations heavily affected the architecture of network facilities and the structure of the telecommunication infrastructure. In this chapter, these historically significant separations flow naturally into a reference model that illustrates the interacting components and dependencies in the NGN environment. To this end a new

plane is introduced, the *service plane*, where logical software blocks (services) encapsulate functionality and become invoked by user applications. The chapter introduces this new layer and shows how it interacts with other layers. Starting from a black-box perspective, Section 2.4.4 elaborates the new layer's dependencies to the surrounding layers and the requirements of the components to be constructed. The layer specification will take into consideration the applicability of the service paradigm and the software-realized composition of services.

## 2.2 Definition of Telecommunication Network

---

In general, a network is defined as an interconnected system of things or people. [Uni] *A telecommunication network is an interconnected system of electronically exchanged information.* The network-connected roles of transmitter (the user who sends information) and receiver (the user who retrieves the information) interact as the consumers of services obtained from a telecom service provider.

## 2.3 Evolution of Networks

---

The International Telecommunication Union (ITU) extends the term *service* to include two concepts, a specific objective on the part of the user, and the coupling of that objective with an appropriate access. [IT93] The user connects to the network with the objective of exchanging information. The process of connecting users to each other consists of two independent steps: *signaling*, the initialization of the communication, followed by *delivery*, the transmission of the information. [Pro98]

Historically, the telecom network's exchange of information and its handling of signaling and delivery were improved in four evolutionary steps:

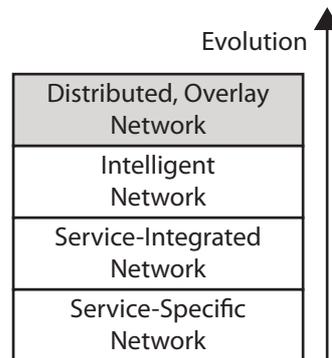


Figure 2.1: Network Evolution Milestones

### 2.3.1 Network Convergence

In discussing the evolution of telecom networks it is worth our while to give a sidelong glance at *network convergence*, a concept that mirrors, in the user's experience, a core tendency of the technology of network evolution. Network convergence is the independent use of services without regard to devices, networks, and locations.<sup>1</sup> The benefits of a converged network include: [Mil02]

- A reduction in the overall complexity of the networking infrastructure (for example, fewer protocols and operating systems)
- Synergies between carrier circuits supporting voice and data, and possible elimination of circuit redundancy
- Possible reduction in carrier circuit charges
- Instead of separate management systems, integrated management systems and strategies that can support both voice and data networks
- More consistent user interfaces
- Access to enhanced web applications (such as integrated messaging, voice-enabled websites, and desktop video conferencing)

Convergence reflects the user's experience of the changes in telecom networks, while the four milestones illustrated in Figure 2.1 reflect the actual technical changes that have been made.

<sup>1</sup>Convergently operated services are also called *seamless* services.

### 2.3.2 Service-Specific Networks

The history of telecommunication networks begins with the development of voice transmission, which was accomplished by using a circuit-switched infrastructure. A circuit-switching network is one that establishes a dedicated circuit (or channel) between nodes and terminals before the users can communicate. [Pro98] The signaling and delivery intelligence is centralized in the service-specific core of the network. The circuit-switched infrastructure of voice transmission is known as a Public Switched Telephone Network (PSTN) and is designed for just one dedicated service, the Plain Old Telephone Service (POTS).<sup>2</sup>

This service-specific pattern shaped the early development of network design. Networks were built to support one dedicated service and were not designed to be used by other services, a limitation that resulted in high maintenance costs. Other disadvantages became evident: [Kel98a]

- Monolithic architectures built on existing open interfaces between components that were not expandable or usable for other services, thus providing no potential for resource scaling (one service, one operator, one provisioning)
- Increased effort in planning, installation, implementation, and operation
- No support for service creation and service execution
- Less transport availability through dedicated channels, resulting in the danger of connection breakdowns

### 2.3.3 Service-Integrated Networks

The next step in the evolution of telecom networks can be seen in service-integrated networks. Services such as voice call, fax or data transfer, which had been implemented in service-specific devices, were now integrated as capabilities of the network. An example of such development is Integrated Services Digital Networks (ISDN). Core signaling and delivery capabilities were implemented in network protocols, and common access functionality was swapped into open interfaces to be used by multiple services. [AZV]

The main advantages delivered by service integrated networks were:

---

<sup>2</sup>More examples of service-specific networks are described by Kellerer in [Kel02].

- Open monolithic architecture, with an implementation of intelligent signaling and delivery protocols, which produced:
  - A first, if limited, separation of service connectivity (signaling and delivery) from service control<sup>3</sup>
  - Open access through access protocols
- Through network intelligence, a reduction in maintenance effort, because changes were accomplished using routers and switches, thus allowing greater agility

Further in the development of service-integrated networks, there were attempts to overcome the limitations in separation of connectivity from control by shifting the excessive transport functions from the nodes inside the network to the edges of the network. The result of this shift was an asynchronous transfer mode (ATM) network architecture, which was part of the evolution from N-ISDN (narrowband) to B-ISDN (broadband).<sup>4</sup> [Pry89] An important step forward with B-ISDN was that it supported Quality of Service (QoS), which allowed major improvements in voice, data, and video services.

For this evolutionary step, unification or integration of services was key, while the underlying network technology assumed secondary importance. [AA02]

### 2.3.4 Intelligent Networks

Intelligent Networks (IN) followed service-integrated networks in network evolution. The ITU's recommendations concerning IN were realized in phases. These phases were based on capability sets (CS), an analogy the ITU started using in the development of services in general, to the point that it became part of the ITU's definition of the term *service*. The ITU created a first set of capabilities (CS-1) in 1993, and others were released in the last few years. [Hua97] CS-1 specifies the single-ended and single-point-of-control services called Type A services.

Capability sets were defined in service feature sets. These could be aligned to functional entities (FE) and physical entities (PE) depending on the IN view. Sample service features (SF) in CS-1 are Call Forwarding (CF), Customer

---

<sup>3</sup>Service control was still supported only for a dedicated service.

<sup>4</sup>More information can be found in materials relating to the RACE project. [Kel98b]

Profile Management (CPM), and Personal Numbering (PS). The IN standardization also specifies a framework to describe and design IN structures outlined in the IN Conceptual Model (INCM). [Hua97]

The following enhancements were delivered by the IN concept: [SC94]

- Layered architecture, which included among its advantages:
  - A decoupling of the service environment (delivery and control) from the network, replacing the limited separation achieved in service-integrated networks
  - The Unified Functional Methodology (UFM), a service creation methodology
  - Service Independent Building Blocks (SIBs), a defined architectural presentation of service, which was intended to encapsulate reusable blocks, possibly chained to provide service processes [Acc99]
- Separation of service control logic from service switching
- Logical separation of applications from underlying processing platforms
- Support of scaling mechanisms for multiple operators, services, access networks, and devices
- Support of multiple service applications on a standardized service logic execution environment (SLEE)
- The use of service creation environments to create, simulate and test new services
- A paradigm shift in transport: the coexistence of circuit-switched and packet-switched mechanisms

Later developments of IN were the Advanced Intelligent Network (AIN) and the Wireless Intelligent Network (WIN).<sup>5</sup>

---

<sup>5</sup>These are described in various sources. [Ker96], [SC94], [Rob91]

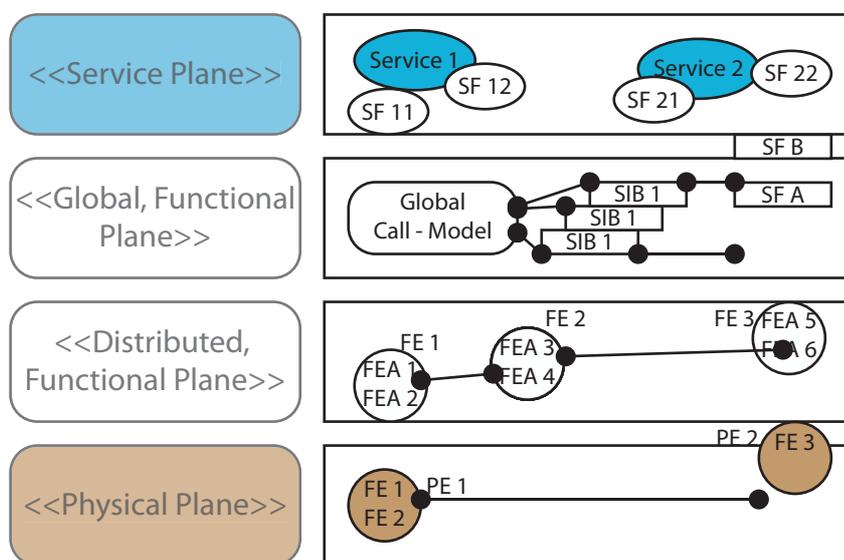


Figure 2.2: IN Conceptual model

### 2.3.5 Distributed, Overlay Networks

A more recent stage of network evolution produced what are sometimes called *distributed, overlay networks*. The attribute *distributed* refers to the wrapping of existing networks, while *overlay* addresses the adding of an additional layer above the existing infrastructure. These networks introduced the ability to implement a network on top of previously implemented networks.

Two architectural approaches of distributed networks are observable: One with a centralized network, and the other with a decentralized provisioning of core functionality. A typical example of a decentralized network is the peer-to-peer network, where the intelligence is located in the distributed nodes of the network. Peer-to-peer networks represent a successful overlay approach, and have been described as very scalable and high-performance structures.<sup>6</sup> [Zha04]

Centralized core networks usually consist of a network core which is completely independent from the underlying network transport capabilities. The core can transparently interface with all types of transport networks. The core is generally homogeneous and built on the TCP/IP protocol suite. The architecture of a centralized core network is influenced by the IN vision and

<sup>6</sup>Further research on new applications (such as BitTorrent, FastTrack and Gnutella) can be found in Pouwelse et al [Ios06], and on research frameworks (such as MULTIPROBE) in Rhea et al [Zha04] and in Katz and Brewer [KB96].

refined through the following points:

- A strongly layered architecture, which includes:
  - Defined components and interfaces to other layers
  - A centralized or decentralized logic approach
  - A strong decoupling between service lifecycle and network transport, differentiating among service creation, execution, and control, with a clear interface between them
- Scaling capabilities for operators, services, access networks and devices
- Quality of service support
- Open interfaces to other networks (e.g. mobile networks).

Table 2.1 provides more detailed information about the milestones in the evolution of networks.

Networks	Service specific	Service integrated	Intelligent	Distributed
Architecture	Monolithic	Monolithic	Layered	Layered
Scalability	n/a	Services, access	Operators, services, access, devices	Operators, services, access, devices
Transport	Circuit switched	Circuit switched	Mixed	Core IP, access mixed
QoS	Guaranteed	Guaranteed	QoS implemented	End to end
Service Environment	n/a	Service execution environments, hardware-based (routers, switches)	Service creation, service execution environments, based on hardware components	Service creation, service execution environments, based on software components

Table 2.1: Evolution of Networks

## 2.4 Next Generation Networks

---

The most interesting of the core centralized networks is the *Next Generation Network* (NGN). Its development efforts were driven by standardization organizations, vendors, and telecommunication providers.

### 2.4.1 Definition of NGN

Current literature defines NGN in various ways. A working definition can be derived by looking at existing definitions, starting with the specification of the ITU, which was one of the main driving bodies of the NGN initiative:<sup>7</sup>

”A Next Generation Network (NGN) is a packet-based network able to provide services including telecommunication services and able to make use of multiple broadband, QoS-enabled transport technologies and in which service-related functions are independent from underlying transport-related technologies. It offers unrestricted access by users to different service providers. It supports generalized mobility which will allow consistent and ubiquitous provision of services to users.” [IT04b]

**Working Definition of NGN** Another, more general definition describes NGN as an umbrella term for mixed voice and data services, running over the IP protocol: [PCM06]

*NGN is an IP-based network with an inherent service character. The NGN infrastructure offers design and execution capabilities for all kinds of functionality by decoupling network transport and service lifecycle. It further supports control of services and quality of service mechanisms.*

---

<sup>7</sup>NGN is used for a variety of future architectures. Our definition is characterized by a strong software service orientation and influenced by the ITU standards body perspective.

## 2.4.2 NGN Requirements

The NGN is characterized by the following fundamental requirements: [IT04b]

- Packet-based transfer
- Separation of control functions among bearer capabilities, between call and session, application and service
- Decoupling of service provision from the network
- Provision of open interfaces
- Support for a wide range of services, applications and mechanisms based on service building blocks, including real-time streaming, non-real time services and multi-media
- Broadband capabilities with end-to-end QoS and transparency
- Interworking with legacy networks via open interfaces
- Generalized mobility
- Unrestricted access by users to different service providers
- A variety of identification schemes, which can be resolved to IP addresses for the purposes of routing in IP networks
- Unified service characteristics for what the user perceives as the same service
- Converged services between fixed and mobile networks
- Independence of service-related functions from underlying transport technologies
- Compliance with all regulatory requirements, such as requirements for emergency communications, security and privacy

### 2.4.3 Development of NGN

The ITU assigned the development of NGN to various working groups to address research areas such as Service Capabilities, Service Architecture and Service Platforms. Among the results of this effort were the recommendations Y.2001 [IT04a] and Y.2011 [Gro04], which delivered a general overview, principles and models on which our proposed reference model for NGN is based.<sup>8</sup>

Another, parallel working standards body is the European Telecommunications Standards Institute (ETSI). It was inspired by and is well-known for the development of Global Systems for Mobile Communications (GSM). ETSI combined its TIPHON, responsible for the development of interconnecting voice over IP (VoIP) and PSTN, with the Signaling Protocols and Networks group, which was developing a European variety of telecommunications standards. [CSLK22] The combination of TIPHON with SPAN became the TISPAN group, which is dedicated to the following tasks:

- Provide to broadband customers all of the services enabled by the Third Generation Partnership Project IP multimedia subsystem (IMS), and provide selected IMS services to PSTN/ISDN customers connected to an NGN.
- Provide most of the PSTN/ISDN services now offered by network operators using legacy equipment and interfaces, to support PSTN/ISDN replacement scenarios.
- Extend the 3GPP IMS to cover those regulatory areas that 3GPP may not have covered: emergency calling, Lawful Intercept (LI), and possibly Malicious Call Indication (MCI).

The 3GPP IMS was selected because it provides several of the fundamental characteristics of an NGN.<sup>9</sup>

Beyond its IMS support, ETSI's TISPAN defined another NGN architecture called TISPAN NGN Release 1, which includes the following elements: [CSLK22]

---

<sup>8</sup>Further details about NGN standardization in the ITU are provided by Cochenec. [Coc02]

<sup>9</sup>IMS is described in Section 5.4.1.

- A service control plane supporting diverse service subsystems (initially PSTN emulation and an adapted IMS)
- A separate application plane
- A core transport plane based on IP technology
- Integration with existing broadband access networks
- Security, QoS and network management

#### 2.4.4 Reference Model

Focusing on the intersecting points in both approaches, ITU and ETSI TISPAN, in this chapter we derive and introduce an NGN reference model that arranges functionality and facilities with a focus on the human end user and software application service access.

The NGN reference model consists of five planes, exposing several top-down, bottom-up, and plane-plane dependencies. The planes are arranged top-down as seen from a software application point of view, and bottom-up in terms of access by an end user. Software services are requested from the application plane, while human end users request network capabilities via the underlying customer equipment.

In the following sections, each plane is defined and outlined by its functions, components, and dependencies, beginning with the customer perspective (bottom-up).

##### Customer Equipment Plane

Customer equipment (CE)<sup>10</sup> consists of all telecommunication access equipment in the customer's residence [Eld97]. CE changed dramatically over the course of network evolution. Functionally, the CE plane builds the customer's interface with telecommunication services. [Woe06] For this reason, it should support the expectations and behavior of the targeted customer group.

The CE plane consists of various hardware devices, though it usually also involves software components. The network revolution has bred large numbers of CE devices, such as residential gateways (RG), set top boxes and mobile

---

<sup>10</sup>CE is also known as customer premises equipment (CPE).

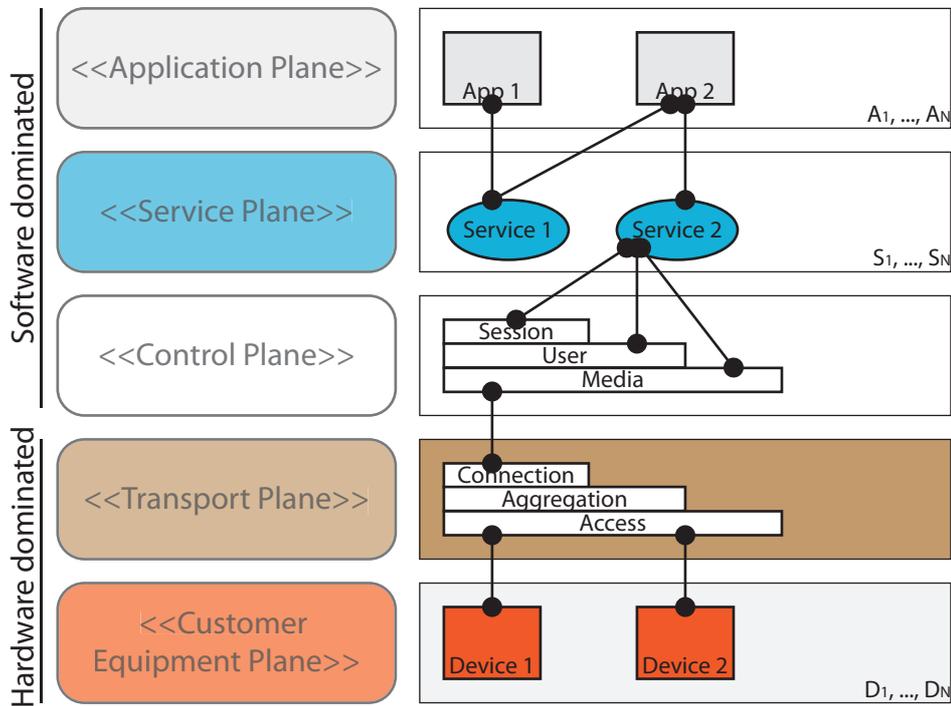


Figure 2.3: Reference Model of NGN

phones. Although some research ignores the software element of CE, here we will treat software as an important part of the CE plane.

The CE plane abuts on the transport plane, using its access capability. There is a strict dependence between a CE device and its transport access, which will become more open because of the generic use of IP and the acceptance of more than one access network seamlessly served by the device.

### Transport Plane

The transport plane manages the transfer of packets between the physical resources of a network. This is reflected in two basic functions:

1. *Access*: enabling access to the CE device
2. *Aggregation*: grouping the incoming packets (from access) and transmitting them over a backbone

Within the network, we distinguish between *core*, which refers to the interaction point of the provider, and *edge*, that of the customer. The transport

carries packets from edge to edge through the core. To perform this task it uses a variety of accesses, both fixed and mobile, and aggregation networks (backbone). The transport plane is very restrictive to the underlying CE, and the CE has to accept its policies. In an NGN the transport layer simply routes the packets, without implementing much logic. This routing includes switching, partial signaling and media delivery.

An opportunity exists to establish powerful functionality at this plane. In some cases it may be appropriate to do this. However, functional solutions applied in the transport plane tend to be inflexible and rigid, which is contrary to the provider's interests. Business and generic service functionality are more appropriately implemented on the upper, software-dominated layers, and the transport layer can be downgraded almost to the status of a simple bit transmission pipe.

### Control Plane

The control plane works as a central management hub, supervising the interacting services as they perform the functions of session, user and media handling. These three functions are swapped out of the usual functionality-handling service layer into this new control layer, where they provide signaling and delivery capabilities.

The control layer consists of points, mainly divided into signaling and resource functions, that maintain communication agreements (protocols) for cooperation. The tasks of session, user, and media handling are built around interacting software nodes, communicating through IP-based protocols. This implies central processing of at least the CRUDF<sup>11</sup> operations of these services.

The control plane is the border between the hardware- and software-dominated worlds. The NGN control layer provides software abstraction mechanisms to encapsulate transport specifics. Another dependency exists from the control layer to the upper service layer; this dependency is characterized by a frequent exchange of information. Service and control components must interact smoothly. For this reason it is recommended that the design of the control plane and the service plane be architecturally homogeneous.

---

<sup>11</sup>The term CRUDF originated in database access, where it is used to refer to the basic object operations Create, Read, Update, Delete and Find.

## Service Plane

The service plane works as the functional enabler of the user's perceived telecommunication objectives. It leverages the design and processing of services as reusable building blocks inside the NGN. The purpose of the (emerging but still optional) service plane is to encapsulate telecommunication services to reuse them in the context of business applications. Harnessing the power of reusability, this plane is functionally the most apt for the implementation of agility, and consequently should be the primary focus when discussing software service approaches.

It is important to note that the abstract service creation activity of composition could be implemented in the service plane and thus serve as a distributor of atomic and composed services to the application plane.

NGN-realized services, with few exceptions, are software-based. Typical software constructs such as frameworks, components, and application programming interfaces (APIs) can be used to implement a telecommunication service infrastructure in the form of an NGN service layer. As in the control layer, the main interacting entities are represented by service nodes which communicate through IP-based protocols and software-negotiated agreements.

The service plane is integrated seamlessly into the underlying control layer. Above the service plane, the application plane reuses the services to present them in a user-perceived context.<sup>12</sup>

## Application Plane

The upper layer of our model facilitates the application environment. In this thesis, applications represent rich functionality boxes that can invoke multiple services simultaneously. Like software-implemented services, applications maintain a runtime environment which facilitates their processing. Inherent properties such as state and transaction management are initiated in the application plane and delegated to the service plane. The interface between these two layers is seamless, and from the technical perspective they can be implemented in the same runtime environment. The architecture must find an appropriate way to integrate service design and runtime as well application design and runtime.

---

<sup>12</sup>To guarantee usability in such an arrangement, services must accommodate certain requirements, as discussed in Chapter 6.

Applications have a strong dependency to the CE devices because the applications' client-side software complement is running above the CE. Consequently, the application provider must ensure mechanical compatibility between CE and applications.

## 2.5 Quality of Service

---

In the evolution of networks, a concept known as Quality of Service (QoS) has become increasingly important, as high traffic and high demands for bandwidth have produced an increasing need for predictability and efficient use of resources.

In a basic telecommunication scenario, two prerequisites must be fulfilled to achieve successful communication. First, users must be understood. This primarily requires that transmitter and receiver speak the same language and, secondarily, that sufficient technical capabilities are available. The second prerequisite is that there must be a certain degree of reactivity in case of error occurrence. [LG06] For instance, when a transmission error occurs, the network must react agnostically and initiate retransmission.<sup>13</sup>

**Working Definition of Quality of Service** *Quality of service (QoS), in a telecommunication context, is the ability to ensure specific, quantifiable levels of performance in a network. These levels are service-specific and assignable, and can be guaranteed in advance. QoS allows revenue-generated service levels to be implemented and service level agreements (SLAs) to be negotiated. [LG06]*

---

<sup>13</sup>More in detail, depending on the underlying transmission protocol, TCP retransmissions were initiated by the network, while UDP retransmissions have to be handled by the application.

### 2.5.1 QoS Parameters

The evolution of telecommunication toward IP-based networks is accompanied by a trend toward convergence, which leads to the diversification of applications and services while all contend for bandwidth. A lack of bandwidth is the issue behind most quality problems. [Wal06] When identifying bandwidth issues, the following three parameters should be considered:

- *Delay*:<sup>14</sup> the time elapsed between the transmitter's sending and the receiver's receiving, the time during which a packet is traveling [Wal06]
- *Jitter*: the interval between transmission signal arrivals, which should be stable
- *Loss*: the complete dropping of packets, which happens when components are congested or overflowing

We can distinguish between the generic and service-specific use of QoS parameters. Generic application of these parameters occurs on the bottom layer of the reference model, where transport capabilities are common. Their service-specific use is for the specific quantification of QoS parameters in a Service-Level Agreement (SLA), a quantification requested by the service consumer and provided by the infrastructure. The infrastructure layers involved in providing service-specific QoS are the control and service planes, which implement service-specific functionality.

There are three mechanisms applicable to services that handle QoS definitions:

- *Best Effort*: the basic mechanism that represents the inherent first-in and first-out character of packet traffic
- *IntServ*: the direct definition of parameters within network components [RB94], [SS97b], [SS97a]
- *DiffServ*: the indirect definition of parameters applied to protocol sections to mark for special treatment [Gro02], [Fau02], [JB06]

In an NGN context, the main challenge in implementing QoS is the transformation of the quality parameters into the software world by defining appropriate attributes and metrics. This challenge implies the necessity of

---

<sup>14</sup>The terms *delay* and *latency* are used interchangeably.

defining appropriate thresholds and metrics that are machine-readable and processable.

QoS is a fundamental requirement of NGN. In NGN, agility is key, but is also meaningless without quality of service. The software approach of this thesis enables the definition and processing of QoS-required parameters and mechanisms. We will illustrate how to design a QoS solution in Chapter 7 and implement it in Chapter 8.

## 2.6 Drivers and Requirements of the NGN Concept

---

NGN was an integral part of the evolution of telecommunication networks, and coherences are shared between certain key milestones in the evolutions of both NGN and telecommunication. In this thesis, these coherences, listed in Table 2.2, serve as the fundamental rules for the design of a software-based infrastructure.<sup>15</sup>

## 2.7 Summary

---

In this chapter we examined the historical evolution of telecom networks in terms of the milestones in that evolution, particularly Next Generation Networking (NGN). We looked at NGN in some detail, considering most closely the evolutionary coherences shared by NGN and telecommunication, and discussed the role played by Quality of Service (QoS) in network evolution.

We considered in what manner this work, and its framework development, will reflect the milestones in that evolution. We introduced the service plane, a new layer for a telecommunication network model, where services encapsulate functionality and become invoked by user applications. We explored how this new layer interacts with other layers and discussed its dependencies and component requirements.

---

<sup>15</sup>The format POSTULATE.CONCEPT.NUMBER is used throughout the thesis as an identification label for specifications. Table 2.2 uses this format to identify facility requirements: FR.NGN.NUMBER.

---

In the next chapter we will investigate services, in particular the Service-Oriented Architecture (SOA) concept of software service behavior, by exposing SOA's service entities and their interactions.

Specification	Name	Derivation	Description
<b>FR.NGN.1</b>	<i>IP Founda- tion</i>	Data Networks	Basis for the transport is the IP protocol stack.
<b>FR.NGN.2</b>	<i>Transport Abstrac- tion</i>	Intelligent Net- works (IN)	Decoupling of service en- vironment and network through open interfacing.
<b>FR.NGN.3</b>	<i>Services as Inde- pendent Building Blocks (SIB)</i>	Intelligent Net- works (IN)	Appropriate, visible and independent presentation of telecommunication services which refers to Service-Independent Building Blocks (SIB) in the IN concept.
<b>FR.NGN.4</b>	<i>Design / Processing Separa- tion)</i>	Intelligent Net- works (IN)	Separation of service exe- cution (SLEE) and service creation.
<b>FR.NGN.5</b>	<i>Signaling / Delivery Processing)</i>	Control Plane	Separated but high- performance handling of service signaling and media delivery flow.
<b>FR.NGN.6</b>	<i>Service Layering (Appli- cation, Service and Control)</i>	Intelligent Net- works (IN)	Separation of application, service and control logic, which had been consoli- dated into a homogeneous cluster.
<b>FR.NGN.7</b>	<i>Quality of Service Support</i>	Distributed Networks	Service-specific perfor- mance coverage.
<b>FR.NGN.8</b>	<i>Unrestricted Access</i>	NGN	Converged access, inde- pendent of location or de- vice.
<b>FR.NGN.9</b>	<i>Regulation Compli- ance</i>	NGN	Solutions restricted to support regulated mechanisms, rules and guidelines, such as emer- gency calling and lawful interception.

Table 2.2: Drivers and Requirements of the NGN Concept

# Chapter 3

## Services

*“I want perfect meals and perfect service. ”*  
*Anita Smythe, Bright Eyes [1934]*

### 3.1 Synopsis

---

Today’s service economy is surpassing industrial production and the sales of physical goods. We have progressed from an economy dominated in the 19th century by agriculture and extractive industries, through industrialization, to an economy now based on services. [SDfNGNGBY06] In the telecommunication domain we have experienced a similar, if more compressed, evolution from a hardware-based industry to software-based services.

To establish a common understanding of services and their focus in this thesis, we will start by offering an overview of the term *service*, its evolution and some other contexts in which the word is used. This chapter explains how the term is understood in software engineering, especially in a new architectural style called Service-Oriented Architecture (SOA). Analogously to the previous chapter, we identify the milestones that have been the drivers of software service evolution, so as to extract core concepts for inclusion in the framework development in Chapter 7.

In previous chapters we studied the six-layer aesthetic model of Next Generation Networking (NGN). In this chapter we discuss software services, in

particular the SOA model of how software services interact and behave. SOA brings a wealth of dynamics that potentially enrich NGN structures. In this chapter we expose SOA dynamics by specifying SOA's service entities and their interactions, all of which are technologically independent.

Following our consideration of SOA entities and interactions, we take a look at reuse, which is one of the main goals of the service paradigm. Focusing on reuse, SOA introduces the opportunity to compose structures. A discussion of the lifecycle and distribution scope of SOA completes the introduction and characterization of telecommunication and software-service engineering.

## 3.2 Evolution of Services

---

Services were presented in the previous chapter as a common capability set. Historically, they have been used in various research fields such as business administration, telecommunication, and software engineering. Regardless of the fields in which they are used, services share common elements such as entities and activities which were changed and refined during service evolution. The software service concept has evolved toward the service paradigm, a constellation of concepts, values, perceptions and practices shared by the software engineering community to reduce complexity and leverage reuse.

This section shows the evolution of services in the telecommunication domain, where software paradigms are of increasing influence.

To reflect the distinction between business and technical service perspectives, we introduce an extended terminology that differentiates between levels of service development.

### 3.2.1 Business Services

The term *service* is widely used in various domains, and is often defined ambiguously even within a single domain. [SDfNGNGBY06]

Services are basic to the study of economics. Within that field, *service* has been defined in various ways, from "deeds, processes and performances" [ZB96] to "any act or performance that one party can offer to another that is essentially intangible." [Kot88] Furthermore, a service "takes place in

interactions between the customer and service employees and/or physical resources or goods and/or systems of the service provider which are provided as solutions to customer problems.” [Gro01]

In Johne and Storey’s definition [AJ98], a service is characterized by the following four attributes:

- *Intangibility*: Services tend to be intangible, hence physical objects are not necessarily exchanged.
- *Simultaneity*: Provision and consumption of services are contemporaneous, so the provider must work out interaction modes and adequate organizational structures apart from technological capabilities.
- *Perishability*: Services cannot be provided on stock and are inherently different from each other.
- *Heterogeneity*: Services vary in quality.

Figure 3.1 illustrates a simple business service relationship.

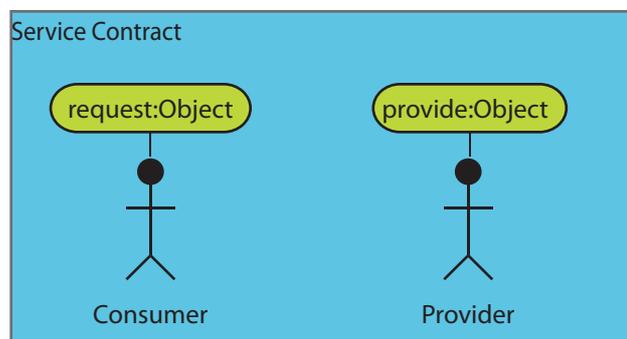


Figure 3.1: Business Service Entities and Activities

## 3.2.2 Telecommunication Services

**Definition, Telecommunication Services** As a type of business service, telecommunication service is characterized in Section 2.3 as *a packaged set of capabilities for exchanging information over distances, perceived by a human end user*. The Telemanagement Forum (TMF) extends this concept as follows: "The telecommunications service provider has the responsibility for the acceptance, transmission and delivery of the message." [For06] The message can be seen as "a set of independent functions that are an integral part of one or more business processes. This functional set consists of the hardware and software components as well as the underlying communications medium. The customer sees all of these components as an amalgamated network. Eventually, a service can be a service component of another service." [For06]

### Perspectives of Telecommunication Services

In this context, we can see that there is a mismatch between the business and technical aspects of the term *service*. From the business (consumer) perspective, a service is a packaged set of functions, billable with its own access. From the technical (provider/developer) perspective, a service is a set of hardware and software components in its infrastructure. [Woe06] It is observable that the number of components on a layer increases from the business perspective to the more detailed technical view, and that granularity is reduced in the same direction. In this thesis the technical perspective is dominant, with a focus on enhancing the provider's platform.

Services can be distinguished by domain relevance into two types:

- *Domain services*, which deliver direct business value
- *Utility services*, which promote the domain services and encapsulate basic infrastructural functionality

Another, more telecom-oriented classification is the differentiation between operational and business services. The telecom community collected all services together, handling provisioning services centrally under the term Operational Support Systems (OSS). The counterpart of this classification is Business Support Systems (BSS), which process consumer-related data (such as orders and user data). In the light of this classification, OSS services provide

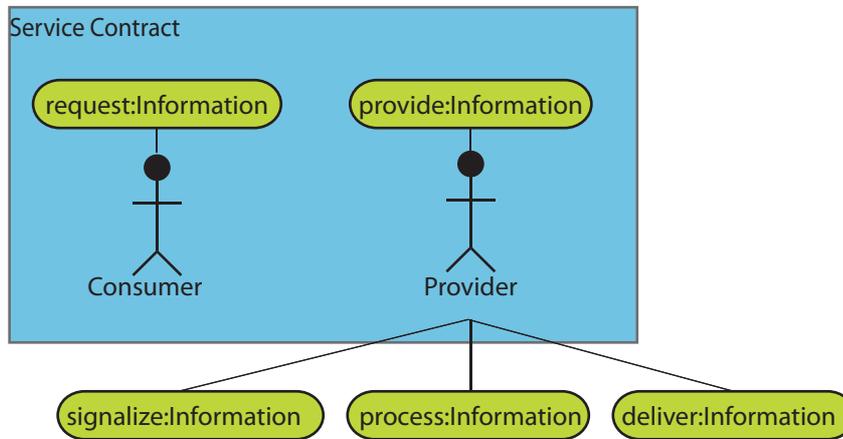


Figure 3.2: Telecommunication Service Entities and Activities

a BSS-independent layer to decouple network operations from the business aspects of the telecom infrastructure.

Technically, service components were realized using software and hardware building blocks in a network infrastructure.

### 3.2.3 Software Services

Software services are a part of the service-engineering discipline, which is a research field of software engineering. Analogous to the divide between business and technical services, two completely different interpretations of software services are currently observable:

- *Software services as a distribution model* addresses the business perspective and is defined as "a mechanism by which users subscribe to some of the software they use and select how much or how little of the full application they want access to."<sup>1</sup> [Elf02]
- *Software services as a software paradigm* focuses on the technical perspective and is defined as "ongoing research with an aim to investigate the requirements, implications, implementation, negotiation and maintenance for providing the users with a service."<sup>2</sup> [Sae05]

<sup>1</sup>Related terminology is *Software as a Service* (SaaS).

<sup>2</sup>Related terminology is *Service-Oriented Computing* (SOC) and *Service-Oriented Development* (SOD).

Here, we will focus on the software paradigm view of software services. The term *software paradigm* may be defined as "a constellation of concepts, values, perceptions and practices shared by a community which forms a particular vision of reality that is the basis of the way a community organizes itself." [Kuh96]

### Relationship of Service Paradigm to Components and OOP

The software service paradigm is very young compared with the previous approaches on which it is built, such as procedures, components, and objects. [EG05] Although services, as a software paradigm, have traceable roots in the component and object-oriented paradigms (OOP), each paradigm has its own principles and implications.<sup>3</sup>

The main assumption of OOP is that real-world issues can be mapped onto object structures and reused on the programming level. From this assumption the OOP formed practical design implications: inheritance, encapsulation, and polymorphism. [GB01] The service paradigm embraces OOP in that it aggregates objects into services. As a result, there is an intensive use of OOP principles and design implications in service-oriented software systems.

The component model describes the solving of programming tasks using *components*, which have been defined as non-trivial, nearly independent, and replaceable parts of a system, which fulfill a clear function in the context of a well-defined architecture. [Kru98] As such, components expose a clearly-defined set of interfaces for component access. In a service context, components can be used to aggregate and bundle service functionally.

---

<sup>3</sup>A *paradigm principle* is a fundamental design rule that is mandatory when applying the paradigm. Further mechanisms that arose through frequent use of the principles were called *design implications*. Unlike principles and implications, which serve to support the coarse design of the system, a *design pattern* concentrates on the building blocks of the software system.

## Reuse

One of the main accelerators in the evolution of the service paradigm is *reuse*, which is the reusing of software building blocks within a software system. Reuse can be seen as an engineering activity that focuses on the recognition of commonalities of systems within and across domains. Various concepts related to reuse have made a strong impact on the evolution of the software paradigm toward services. Among these key concepts is the creation of models with varying abstraction levels<sup>4</sup> and their use during the engineering of an application. [Uni06]

To optimize the reuse of system elements, the service paradigm must be applied on each architectural layer by refactoring redundant capabilities, exposing and promoting common functionality, and composing new business-relevant functions as reusable services. [NB05]

Services represented a foreseeable new stage in a paradigm evolution driven by familiar evolution accelerators:

	Object Orientation	Components	Services
Reuse	Low	Medium	High
Granularity	Fine Grain	Medium Grain	Coarse Grain
Coupling	Tight	Tight	Loose
Communication Scope	Building blocks are individual classes	Building blocks consist of several classes (components)	Building blocks consist of services

Table 3.1: Evolution of the Software Engineering Paradigm

<sup>4</sup>Abstraction levels help to hide implementational details to facilitate the access to certain functionality, e.g. via programming interfaces.

## 3.3 Service Paradigm Entities and Principles

### Entities and Their Relationships

The service paradigm organizes the various players that it involves into three independent but collaborative entities: the service consumer<sup>5</sup>, the service broker<sup>6</sup>, and the service provider.<sup>7</sup> [Tsa05] Service providers develop software services, usually autonomously. After development, the provider registers the service with the broker. The service broker publishes or markets the registered services. [Tsa05] Finally, the consumer discovers the service through the broker and binds the provider's service capabilities. Figure 3.3 illustrates this domain-independent relationship.

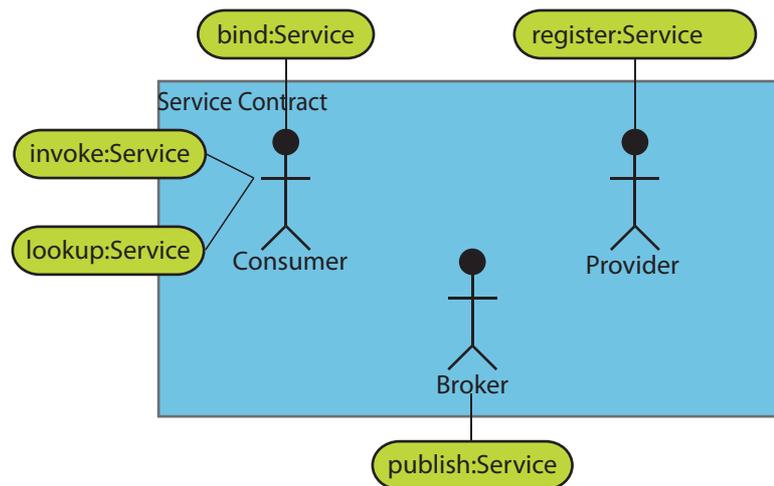


Figure 3.3: Software Service Entities and Activities

<sup>5</sup>In the literature, both *service user* and *service requester* are used synonymously.

<sup>6</sup>The service broker is also known as the *service publisher*.

<sup>7</sup>The service provider is also known as the *service developer*.

## Design Principles of the Service Paradigm

The preceding model offers a dynamic view of the service paradigm, based on activities. Behind this concept of dynamic interaction, the service paradigm evolved its own design principles which are fundamental to the design of service-oriented software systems. From the prototypes of industry frameworks such as Sun's Jini framework [Sun99], the principles were brought to technological independence. The core principles of the paradigm are listed below:

- *Loose Coupling*: Coupling is a measure of the strength of association between two dependents. In the service paradigm, services are coupled loosely to minimize dependencies. They require an awareness of each other that supports high dynamics and flexibility even during runtime. [GA006]
- *Granularity*: Granularity is a measure of the size of the services that define the level of abstraction. In the service paradigm software services are coarse-grained, or at any rate more so than in related paradigms such as components and objects. [Szy] Granularity is one of the main design challenges of the paradigm and involves reconciling providers and consumers, their interfaces, interactions and collaboration agreements, and their internal business processes, data, and (legacy) applications. [DD04] In terms of reuse, the business functionality of a service that is exposed to the user is coarse-grained and can match to real-life activities, while the service representation on more technical-oriented planes has a finer, sometimes even the finest (object-level), granularity.
- *Modularity and Self-Containment*: Services are maintained as separate and distinct units of software that can be used as components in the context of a system. [VID03] In addition, they can consist of one or more other services. The service can even be a part of its own service functionality.
- *Autonomy*: It is expected that services will be developed by autonomous teams. [DD04] This autonomy is a requirement for parallel development and execution. It implies that services have control over the logic they encapsulate. [VID03]
- *Context Awareness*: Though this may seem to contradict the principles of autonomy and modularity, because of the need to support state

information services should be designed to be applicable in a specific context.

The implications of these principles in the SOA architectural style are discussed in the next section.

## 3.4 Service-Oriented Architecture

---

Behind the principles of the software service paradigm, researchers and practitioners discovered rule sets, manifested in a Service-Oriented Architecture (SOA), for applying the paradigm. SOA extends the service paradigm with methodologies and strategies intended to aid in the development of sophisticated applications and information systems that are compliant to the paradigm.

### 3.4.1 Definition of SOA

Software architecture has been defined as a "blueprint" for certain mainly collaborative software development efforts. It is a set of technical, social, and managerial considerations that are "used to guide and control all aspects of software development." [KvDK06] An architecture's specification facilitates communication between all systems stakeholders, including customers, management, and programmers. Today's large-scale, multi-tier architectures are especially in need of guidelines and rules to maintain the overview and orientation of the system.

Unfortunately, multiple and sometimes competing definitions of SOA exist. In the literature, SOA is generally seen as a technical collection of components, "a software architecture that is based on the key concepts of an application frontend, service, service repository, and service bus." At other times it is considered in the light of a theoretical design style, defined as an "architectural style which allows interaction of diverse applications regardless of their platform, implementation languages, and locations by utilizing generic and reliable services that can be used as application building blocks." [Gn05]

Rather than introducing yet another definition, we derive one by combining aspects of existing definitions:

**Working Definition, SOA** *SOA is a software architectural style that, based on the principles of the software service paradigm, gathers considerations of the way services interact in a software environment. The interactions are accomplished among the software entities consumer, provider and broker, and can be defined as activities in a service lifecycle.*

The building blocks of SOA are software services, whose principles are central to the entire software lifecycle. The purpose of SOA is to build a software system using loosely coupled distributed components with minimal dependencies and maximal reuse.

In this thesis, SOA is analyzed in the context of its conceptual static and dynamic perspectives. The static perspective is represented by the service paradigm principles, which lead to specific design implications when using the SOA style. The dynamic consideration is modeled as an entity-activity presentation, evolved from the basics of service relations and ultimately generating an iterative lifecycle process of software services.

SOA adopts all of the principles of the service paradigm. The SOA style takes these principles into account and integrates practical, mature methodologies and patterns. The following section elaborates these methodologies and patterns to provide a comprehensive exposition of the SOA style.

### 3.4.2 SOA's Entities and Activities

Our analysis begins with an overview of SOA's three contracted entities, including a close look at their encompassing contract. It is important to note that these entities serve as logical constructs. Each service can assume one or more entity roles, sometimes even simultaneously. The interactions of the entities are independent of location and loosely coupled.

#### SOA Entities: Consumer, Provider, Broker

**Service Consumer** The service consumer is an application, a service or some other type of software that uses a service. [JM03] The consumer starts a lookup for a service.<sup>8</sup> The lookup is followed by a negotiation in accordance with the service contract. When a service consumption is successfully negotiated, the service is invoked. Part of the negotiation is the service lease, which specifies the amount of time the contract is valid. [WA01] The

---

<sup>8</sup>Lookup is also referred as *discovery* of a service.

consumer obtains a lease from the provider and starts the consumption by binding the service for the time of the lease. The lease is helpful in supporting stated information and reducing coupling between consumer and provider. [JM03] However, it is an optional part of the contract; services can be bound indefinitely.

**Service Provider** From the customer's perspective, the service provider is the service provisioning and maintaining entity. The provider can also act as a service consumer when requesting other services to fulfill the functionality requested by the preceding consumer. The provider is network-addressable and accepts requests from the consumer entity. [WA01] It specifies services, within a given set of rules, in its contract, and registers (publishes) the contract with the broker. In addition to these activities, the provider has opportunities to compose new services by instantiating existing ones, an approach that enhances service reuse, and can support both atomic<sup>9</sup> and composed services.

**Service Broker** The service broker is an optional entity, mainly intended as a registry, facilitating search capabilities to registered services from a specific set of providers. The central task of the broker is to publish the provider's contract and provide negotiation capabilities in accordance with the contract. It serves to decouple provider from consumer by initializing their communication through stating the terms of the contract.

### The Service Contract

The service contract is a collection of metadata and a set of conditions and policies [SR06] that state the guidelines for communication between the consumer and the provider. Constructed by the provider, it is intended to match consumer goals. A basic requirement for this matching process is the discoverability and accessibility of the service. To this end, two attributes are common to service contracts:

- The encapsulation of functionality and the invisibility to the user of implementation details [NB05]
- The degree of cohesion, which refers to the nature of the relationships between the operations in a contract

---

<sup>9</sup>The term *atomic service* refers to a service that is in a non-decomposable state.

Cohesion in a contract can be weak or strong. The stronger the cohesion, the more resilience the contract provides. A contract can maintain consistency only by strictly separating contractual metadata, conditions and policies:

- *Metadata* is macro-level information.
- *Conditions* are parameter-based constraints processed during runtime that control the state of services.
- *Policies* address service-specific arrangements between consumer and provider.

These characteristic elements are specified when designing a service contract. The contract is realized as an interface, machine-readable and processable. The service contract is the central entity of the service lifecycle. For that reason, contract design is central to the SOA style.

### 3.4.3 Implications of the SOA Architectural Style

The service paradigm is characterized by a set of principles (loose coupling, granularity, modularity, autonomy, and context awareness) which evolved from other paradigms (Object-Oriented and Components) by sharing the same basic drivers (reuse, granularity, and coupling). As we have observed, adapted from the service paradigm and formed through scientific and practical work, the SOA concept emerged as a collection of approaches and strategies for realizing the service paradigm principles. Specific mechanisms can be identified to support that transformation. Here we will consider the implications of the service paradigm's design principles in the SOA architectural style.

#### Loose Coupling

Coupling, the strength of association between two dependents, can be described using a scale of two major grades: tight and loose. Tight coupling implies an uncounted number of unknown dependencies. Loose coupling indicates a counted number of known dependencies. Tight coupling in a system raises the specter of spaghetti structures and the necessity of accommodating entire system structures when changing single components. For this reason, SOA generally uses loose coupling. In some cases tight coupling is unavoidable, for instance when performance is a critical issue in binding services.

SOA generally applies loose coupling in two different dimensions, which can be distinguished by their level of abstraction. The first dimension is the location of a service, where the service consumer should be kept aware of any relevant information. The second dimension addresses technology specifics which may not appear in a service definition. Services are defined on a programming meta-level, transparent to implementation languages, frameworks or locations. They are applicable to various technologies and should be technologically agnostic.

Both dimensions affect the design of the service contract to keep a *transparency of service use*, from the consumer's perspective. A logical implication of this principle is the provider's freedom to change these transparent dimensions at any time without letting the customer know. However, there is still a mechanism in place to retain time-based binding: the service lease.

Other mechanisms are applied to support loose coupling:

- Strict separation between contract and implementation
- Hiding of the implementation from the consumer, technological transparency from the consumer's perspective
- Intermediation of a broker as a decoupling entity between consumer and provider
- Definition of a service lease, the amount of time a consumer is permitted to bind the service
- Discoverability, meaning contract-based advertising which makes the services findable and negotiable

### **Granularity**

Granularity, or the size of the services that define the level of abstraction, is applied to the building blocks of software and is declared in the contract. Three levels of granularity can be distinguished: coarse, medium, and fine. Fine granulation describes small functionalities, flexible and agile. A coarse-grained block offers more functionality with fewer external dependencies and forms a more independent unit of specification, design, implementation, deployment, testing, execution, management, maintenance, and enhancement. [GS03]

In an SOA, granularity decisions are made at design time and influence the entire development process. The granularity differs depending on the direction of analysis (business to technology or vice versa). Starting with a top-down (technology to business) approach, developers use coarse-grained services to nurture a domain-driven analysis. Stepping deeper into the design process, services are broken into smaller pieces by specifying more technical details. There is also a strong dependency between granularity and the level of interpretation and abstraction.

There is no one way to implement granularity, but rather an intuitive use which often leads to redefinition and continuous change in the development cycle. Specifying services with respect to granularity also influences the principles of modularity, process awareness and autonomy.

### Modularity and Self-Containment

Modularity describes how units are aggregated with respect to reuse, clarity, elegance, maintainability, and flexibility. Self-containment is a special case of modularization in which a unit can contain itself, which results in a kind of recursion.

Like granularity, modularity and self-containment influence the design process and support opportunities for service aggregation and composition. These concepts introduce a set of criteria for assessing a given service design: [Mey88]

- *Composability*: the ability to aggregate services into parent services
- *Decomposability*: the ability to split a service into subservices
- *Understandability*: the human- and machine-readability of services, addressed in the service contract
- *Continuity*: the minimizing of dependencies with other services
- *Protection*: the restriction of reactions to the service itself, when abnormal behavior occurs (e.g. failure, exceptions)

## Autonomy

By definition, services facilitate the exposition of a set of capabilities. This set is autonomous and can be perceived by a service consumer. This means that the set of functionalities exposed by a service should be unique; the same service should not be exposed twice. SOA supports autonomy through specific mechanisms:

- *Statelessness*: States are not shared between services, and services are primarily stateless.
- *Accessibility*: Services are accessed only by well-defined access points (endpoints) which are also part of the contract definition.

## Context Awareness

Context awareness means that services are designed to be applicable in a specific context, to facilitate processing and provide the consumer with an appropriate, customized behavior. [DS05] The context of a service defines its potential use in a consumer-determined application.<sup>10</sup> Ideally, during creation a service should be autonomous and modular, but during processing it should be context-aware.

Context awareness has a direct relationship with granularity: A service that is not used in an application context is probably not designed with an appropriate granularity. Context should be applicable to services without changing the contract. For this reason, the service contract should be designed to create an equilibrium between autonomy and domain awareness.

---

<sup>10</sup>The term *context* itself has a very wide definition and has inspired many research discussions. There is no need, here, to add another definition. We will focus on the characteristic of context that it exposes the ability to add state information to a service.

### Overview of Design Principles and Implications in SOA

The following table lists the principles of the service paradigm and some of the implications that have been derived from those principles in the SOA architectural style.<sup>11</sup>

Specification	Name	Derivation	Description
DM.SOA.1	<i>Decoupling of Service Contract and Implementation</i>	Loose Coupling Principle	Implies the separation of interface (contract description) and implementation (logic) in reference to the dimensions of location and technology.
DM.SOA.2	<i>Lifecycle Support</i>	General Principles	Covers the main phases of the service lifecycle.
DM.SOA.3	<i>Statelessness</i>	Modularity Principle	Inherently, standalone services share and contain no state information.
DM.SOA.4	<i>Brokerage</i>	Loose Coupling Principle	Covers optional integration of a broker.
DM.SOA.5	<i>Service Leasing</i>	Loose Coupling Principle	Covers optional time-based service binding.

Table 3.2: Design Principles and Implications of the SOA Architectural Style

<sup>11</sup>For the specification, Table 3.2 uses the identification format DesignMechanism[DM].SOA.NUMBER.

## 3.5 Lifecycle and Scope

### 3.5.1 Lifecycle

Beyond the basics of the service paradigm, its principles and implications, services step through a set of lifecycle phases. The development lifecycle of a software service starts with specification and traverses further stages until it reaches binding to the consumer. The steps in the service lifecycle are fundamentally embedded in the service paradigm and are implemented in the SOA style.

Figure 3.4 is an illustration of the service development lifecycle.<sup>12</sup>

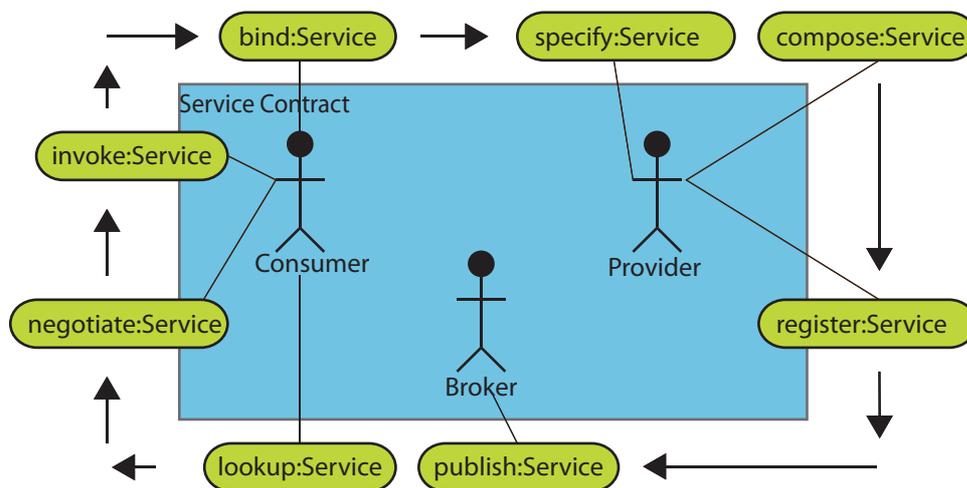


Figure 3.4: SOA Service Lifecycle

In the context of this thesis, it is important to recognize, especially, the role of service composition, which occurs between specification and registration, both carried out by the provider. Iteratively following specification, composition creates services by reusing existing ones. Chapter 4 provides an in-depth consideration of service composition.

<sup>12</sup>This model focuses on the development of services and does not include considerations of the governance of services.

### 3.5.2 Distribution Scope

Having reviewed the entities, relationships, and principles of SOA, it may be useful to mention the scope where SOA is applied. SOA is designed as a distributed concept with boundless use. Unfortunately, so far its use has been largely restricted to internal use in companies, with no communication to the world outside. This would seem to be a waste, given the potential of SOA to connect highly diversified and heterogeneous environments beyond company boundaries. Common reasons for this restricted use are security and trust problems, which are, of course, worth consideration. SOA addresses these issues through policy and trust management, established as part of the service contract among the entities of consumer, broker and provider. Policies, established as part of the service contract, are negotiable, depending on the required distribution scope of the service architecture.

Distribution scopes, seen from a service provider perspective, may be organized into three categories: intra-, inter- and extra-company.<sup>13</sup>

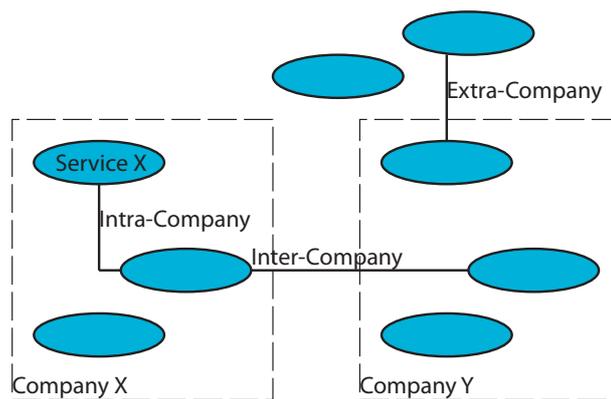


Figure 3.5: SOA Distribution Scopes

<sup>13</sup> *Extra-company* refers to consumer relations.

## 3.6 Summary

---

In this chapter, we have seen how the software service concept has evolved toward the service paradigm, a constellation of concepts, values, perceptions and practices shared by the software engineering community. We exposed SOA dynamics by specifying SOA's interacting service entities and their interactions, and considered the implications of the service paradigm's design principles in the SOA architectural style.

This concludes the Part I of the thesis. In Part II, we introduce composition as an activity of the service lifecycle and elaborate the behavioral patterns and properties of composed services. We then go on to discuss the framework-based service composition of telecommunication services.

## Part II

# Composition Model and Related Work



# Chapter 4

## Service Composition

*“Every living thing has a chemical composition, and anything that is added to it changes it. ”*

*Bonanza Jellybean, Even Cowgirls Get the Blues [1993]*

### 4.1 Synopsis

---

We have previously discussed composition, in the context of service engineering, as a part of the service lifecycle. Generated by the provider and transparent to the actual service consumer, composition is accompanied by the activities of specification and registration. A provider develops services by either creating new ones from scratch or composing and decomposing existing ones. In other words, to develop a given service the provider must choose either creation or composition. From already existing services, composition produces more complex, highly functional ones, by means of reuse, extension, restriction, parameterization, or specialization.

The aim of this chapter is to postulate a formal approach to composition realization within the telecommunication domain. The constructs *behavior*, *patterns*, and *non-functional properties* are used to build a syntax of

composition. Since the intended formal approach is expressive and dialectic, the descriptive concepts of syntax and semantics are well suited for this analysis. While syntax expresses the form or structure of composition, semantics completes it with the meanings of the expressions. [Seb89] Hence, the final outcome of this chapter is a syntactic specification of service composition, compilable into language constructs, technology-independent but implementable and cut to fit services as building blocks. As in the service contract, this specification expresses the objectives of a composition in a technology-independent description, which serves as a template for the implementation in Chapter 8.

The syntactic specification consists of the language specification and a basic notation to model service behavior with respect to the encompassing rely conditions, also seen as the non-functional properties. The complete system will be modeled as a *composition machine* so as to concentrate on the system's core entities and reduce its complexity.

The significant achievement of this chapter is to build a foundation for the composition of software services, consisting of a syntactical specification and its accompanying properties, usable in a state-of-the-art benchmark analysis, and also of use in the initialization and reproduction of a telecom domain-specific composition within the framework in Chapter 7. In addition, the patterns proposed here serve as a basic template to demonstrate the service composition behavior embedded in the prototyped scenario in Chapter 8.

The syntactic specification developed here is compatible with the grammar of extended BNF.<sup>1</sup>

---

<sup>1</sup>Extended BNF is specified in ISO Technical Report 14977 [ISO96] and elaborated in Appendix A.1.

## 4.2 Definition of Composition

---

In taking a closer look at the composition of software services, it will be helpful to define and characterize some key concepts.

**Definition of Composition** In the literature, certain terms tend to be defined synonymously but are actually used very differently. In particular, in this work we distinguish between *orchestration* and *composition*. The primary difference is that orchestration focuses on the design of a single-service model in which the service becomes orchestrated, while composition creates a completely new service by focusing on transition properties.

*Service orchestration is the coordination of the sequence and conditions in which one service invokes other services to realize some useful function.*<sup>2</sup> [Con04]

The focus of orchestration is the single member service and its behavior, which are described in properties dependent on a user defined goal. This is generally achieved by coordinating the member service using flow commands in an orchestration topology.

*Service composition is the set of activities and conditions required to combine and sequence multiple cooperating independent entities which exchange messages so as to perform a task and achieve a goal.* [Con04]

The focus of composition is its result, which is a new service, the goal service. In composition, which is on a higher level of abstraction than orchestration, the functional and non-functional properties of member services are transformed into the properties of the goal service. The circumstances of this transformation are the object of composition research.

---

<sup>2</sup>The term *coordination* is used similarly.

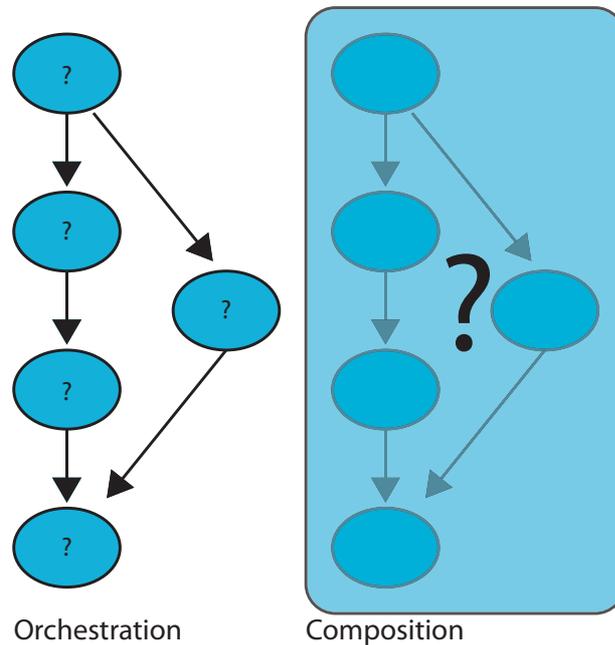


Figure 4.1: Orchestration and Composition

### 4.2.1 An Overview of Service Composition

In general, service composition consists of two successively timed steps, design time and runtime. A service composition is created at design time and processed at runtime. The design time focuses mainly on the arrangement of the control and data flow of the services, to define the order and constraints of the composed elements. At runtime, a service undergoes from one to three stages: initialization (preprocessing), execution (processing) and termination (postprocessing). Initialization and termination are both optional processes.

The entity resulting from service composition is a new service, which reuses available ones. This new service is a *goal service*, which describes the consumer's desired composition state. After composition, the service lookup<sup>3</sup> cooperates with the activity of composing. The lookup includes the sub-activity of matchmaking, which consists of a search for appropriate candidates and the verification of them. Both matchmaking and verification must be considered when investigating the potential automation of service invocation.

A central issue of composition is the method and format of composition

<sup>3</sup>lookup is also known as *discovery*.

description, which must be on the one hand machine-readable, but on the other hand open for human developers' intervention, preferably in a visualized manner.<sup>4</sup>

Further analysis of composition issues leads to a set of constructs which enhance the definition of composition behavior:

- **Elements:** the entities involved in the composition process, services and the transitions between services
- **Behavior:** the dynamics of the composition activity
- **Patterns:** templates of composition behavior, scientifically derived and practically approved<sup>5</sup>
- **Non-functional Properties:** secondary effects, not related to a specific service, that must be considered when designing composable environments

## 4.3 A Composition Machine

---

To facilitate the analysis of composition dynamics, we propose the development of a service composition machine. Figure 4.2 illustrates the interacting entities of this composition machine, taking into consideration the SOA entities and focusing on the composition of services.

The service is described in the *service contract*, its technology-independent interface. In 3.4.2 we listed three elements in a service contract: *meta-data*, *conditions*, and *policies*. The contract is centrally involved when new services are specified through composing. Within a composition, multiple *service descriptions* are arranged to fulfill a goal. The entity that includes all composition postulates is the *composition description*, a machine- and human-readable, syntactic and semantic expression of the behavior of the

---

<sup>4</sup>Visualized composition, as a way of developing services, is supported by certain Integrated Development Environments (IDEs) to define the control flow and parameters of processing. Visualized composition is not investigated in this thesis, but could be the objective of closely related future research.

<sup>5</sup>These patterns should not be confused with *design patterns*, an architectural design solution.

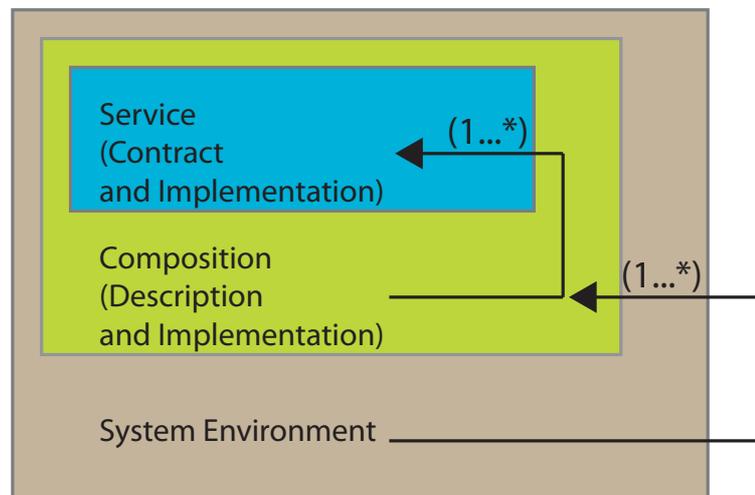


Figure 4.2: A Composition Machine

entire composition and particularly of the interworking services in the composition. The surrounding *system environment*, where the composition description and service descriptions are designed and processed, provides non-functional properties and defines the technical parameters of the design and processing of the composition.

Within the composition machine, we can identify three dependencies when designing composition approaches:

- *Service to composition relationship:* A composition connects one or more services to reach a consumer perceived goal.
- *Composition to system environment relationship:* The system environment includes one or more compositions, supports the composition design and guarantees a reliable and scalable composition processing.
- *Service to system environment relationship:* The system environment provides atomic<sup>6</sup> service descriptions which serve as composable inputs.<sup>7</sup>

It is important to note that service contracts are restricted to the description of atomic services, while a service composition description specifies non-atomic services.

<sup>6</sup>An atomic service is one that is in a non-decomposable state.

<sup>7</sup>*Composability* is the degree to which atomic services have the potential to be used in a composition. [Med04]

The entity on which we focus in this work is the service composition description, and its associated system environment, as it is applicable to the telecommunication domain. A composition description specifies the goal to be achieved, the elements to be arranged, and the constraints to be satisfied. To this end, based on our requirement of machine and human readability, we suggest a descriptive language approach inherited from the service contract. The elements of a service contract, metadata, conditions and policies, are adapted to characterize basic behavior in the composition description. In addition, this chapter introduces attributes for the storage of semantic information in a composition description.

## 4.4 Elements

---

The main components of a composition are interacting elements, which we have previously introduced as software services. These services must be connected with one another to compose member services and achieve a goal. Hence, services must be composable and callable via transitions. There is a need to identify stereotyped connectors between services to influence behavior within the control flow.

Services themselves were described in Chapter 3 as possessing these contract elements:

```
1 Composition := [Metadata], {Condition}-, {Connector}-, {Transition}-,  
2 {Service}-, {Attribute}-;  
3 Service := [Metadata], {Condition}-, {Policy}-, {Attribute}-;
```

## 4.5 Behavior

---

Taking into account the active nature of composition and its elements and interactions, it is important to consider service composition dynamics (behavior). Fundamental to this consideration is the separation of control and data flow, which is crucial for large, scalable systems.

When a service becomes part of a composition, a new service is created.<sup>8</sup> This new service is called the *goal service*. It consists of member services that were composed to reach the goal state.

To differentiate between member services in a composition we establish the roles *caller* and *callee*. In addition, we introduce the roles *start service* and *end service* to identify the first and the last service of a composition. These roles are particularly helpful when analyzing the composition path. In a kind of recursiveness, the goal service, specified by the provider, composes (consumes) member services (start, end, caller, callee) that have been specified by that or some other provider.

Member services are either atomic or (previously) composed. This inclusion of already composed services in a further composition raises the issue of circularity. For this reason we propose the term *composition depth* to identify the recursiveness of a service as used in a given composition.

### 4.5.1 Conditions

The processing of a composition relies on certain set of conditions<sup>9</sup>, listed below. Depending on the interaction stage, a service handles certain processed data which is defined in the assigned condition. When a condition is analyzed by the system, a parameter-based processing takes place, leading to an appropriate reaction at the end of the processing.

In general, conditions may be classified into four types: [Kru92]

- Preconditions, notation PRE: Must be considered before the service is processed. Influences the composition state before processing. Applies to composition member services and to the goal service. Based on input parameters. A reaction occurs in the initialization stage.
- Postconditions, notation POST: Must be considered after service processing. Influences the composition state after processing. Based on output parameters. A reaction occurs in the termination stage.
- Invariants, notation INV: Must be considered throughout service processing. Is constant through all composition states. The reaction occurs during processing, through all stages.

---

<sup>8</sup>Since the focus of the composition lies in the new, composed service, when we use the term *provider* we are referring to the provider of the new service.

<sup>9</sup>Conditions are also termed constraints.

- Rely Conditions, notation REL: Relates to the surrounding system. Influences the system state throughout composition processing. Based on system parameters. Reaction occurs through all stages.

To denote conditions in the syntax we use four clauses, written in uppercase letters:

```
1 Condition := 'PRE' | 'POST' | 'INV' | 'REL';
```

### Service Flow Control

To control the flow of interacting services we need to introduce additional control elements.<sup>10</sup> Control can be declared by behavioral connectors, which manage transitions between services. The purpose of these connectors is to define the order of the composed member services. [vdAADtH04]

We can identify eight basic connectors, denoted in uppercase letters:

```
1 Connector := 'START' | 'END' | 'SEL'(*Selection*) | 'CH'(*Choice*) | 'FORK'
2 | 'MERGE' | 'JOIN' | 'LOOP';
```

### Data Flow Control

Data elements are arithmetic objects, or sets<sup>11</sup> of them, which are parametrized and given direction in a composition. Parameters are explicitly analyzed by conditions, depending on the stage where the service processes the parameters:

- Input Parameter, notation IN: outcome of the termination stage of the previous service on the composition path, processed by the initialization stage of the next service, and directed to the processing stage of that service.
- Output Parameter, notation OUT: outcome of the processing stage of a service, processed by the termination stage of the same service, directed to the next service.

<sup>10</sup>Disciplines of software engineering interested in the management of control flow are workflow [vdAADtH04] and Business-Process Management (BPM). [AHK05]

<sup>11</sup>A set is a collection of objects without any particular order.

- In and Output Parameter, notation `INOUT`: outcome of the termination stage of the previous service, untouched, or read only, by all stages of that service, and directed to the initialization stage of the next service.

When looking at data, compatibility is an issue of considerable interest. Service composition data compatibility issues appear between interacting member services at the moment when the services hand over the data. Especially given the distributed nature of service environments, the format of the data is likely to be different from one service to the other. There are two transformation mechanisms available to adapt data between services:

- *Explicit Transformation*: a transformation that involves a predefined rule set that applies to mappings for all parameters
- *Implicit Transformation*: a transformation that does not have a defined rule set, and that happens automatically (e.g. via introspection) during the composition activity

Explicit transformation can happen within a member service or between two services, but since transformation is not specific to one service but is defined for the entire composition, it is more useful to denote connectors in the composition description:

```

1 Parameter := 'IN' | 'OUT' | 'INOUT';
2 Connector := 'TRANSFORM';

```

Introspection, the ability to analyze data during runtime, is worth considering for data transformation and is demonstrated in the prototype in [Chapter 8](#).

## 4.5.2 Behavior Model

To illustrate the dependencies of the data introduced here, we propose the following reference model. The model shows the goal service and an example of a caller-callee pair, part of a composition and located on the composition path. The focus lies on the dependencies between caller, callee, goal service and their environment.

In the next section, this model is used to analyze the dependencies within a composition and its behavior patterns.

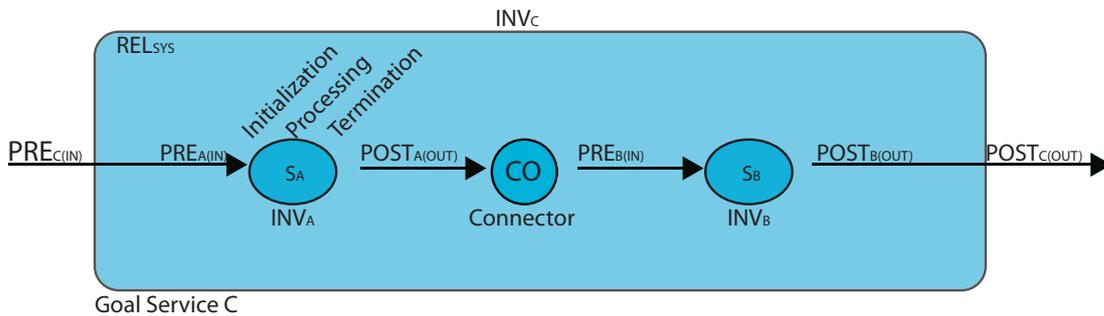


Figure 4.3: Composition Behavior

## 4.6 Patterns

Composition consists of an arrangement of a set of behavior patterns, defining the choreography of a composition. To analyze the choreography within a service composition, we must examine the patterns that are used in composing services. Patterns are constructs that describe a general, repeatable solution to a commonly occurring problem, and are usually used in the design of software engineering. In this section we will introduce the patterns of behavior between member services in a composition. Adapted from the Gang of Four pattern analysis [GHJV00] and tailored to suit our analysis of composition, these patterns will be elaborated using the following criteria:

- **Name and Classification:** identifier and explanation
- **Also Known As:** well-known synonyms
- **Structure:** graphical representation of the pattern
- **Notation:** logical description of the pattern
- **Issues:** potential problems with the pattern

Subsequently, we will describe a set of patterns, *Sequence*, *Parallelism*, *Exclusive Decision*, *Multiple Decision*, and *Looping*. This pattern list is not exhaustive, but mirrors the typical behaviors in a composition path.

### 4.6.1 Sequence

**Name and Classification** Sequence describes a pattern where services are processed exactly in the order in which they were defined, and the output of a caller becomes the input of the following callee. This pattern does not support any kind of parallelism. It is the simplest and most common behavior pattern in composition. In this pattern, conditions cascade.<sup>12</sup>

#### Also Known As

- Routing

**Structure** A sequence is illustrated in Figure 4.4.

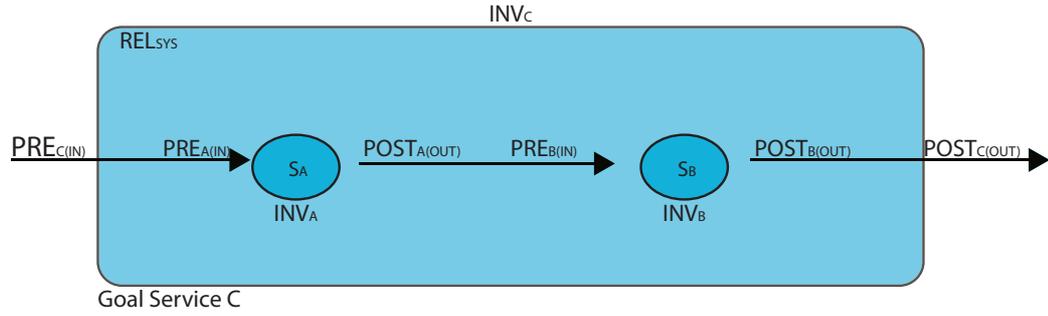


Figure 4.4: Structure of Sequence Composition Pattern

#### Notation

$$C := A \triangleright B$$

Conditioning

$$\begin{aligned} REL_{SYS} \wedge (INV_C \wedge (INV_A \wedge (PRE_A \wedge POST_A)) \wedge \\ (INV_B \wedge (PRE_B \wedge POST_B))) \\ := output_C \end{aligned}$$

Sub-Conditions

$$PRE_C := PRE_A, POST_B := POST_C, output_A := input_B$$

<sup>12</sup>However, rely conditions are valid for the entire system.

**Issues** Sequence is a straightforward composition pattern in which a service is executed only when all conditions of the previous service have been satisfied. It is necessary to plan for alternative handling, consisting of compensation or abortion of the composition path, for the case where these conditions are not met.

## 4.6.2 Parallelism

**Name and Classification** Parallelism describes the processing of multiple services at the same time. In this pattern, a set of two or more composition paths must be processed concurrently. Parallelism is closely related to the Decision patterns in that there is either splitting of the composition path or an aggregation into a single path.<sup>13</sup> With parallelism, the alternative paths are processed concurrently and all affect the composition goal.

### Also Known As

- Concurrency

**Structure** Parallelism is depicted in Figure 4.5.

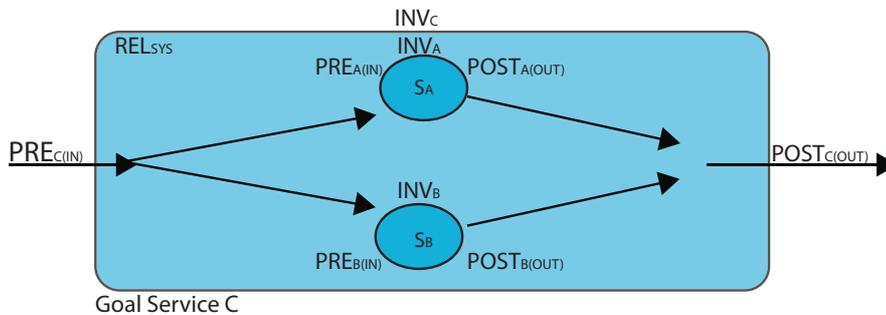


Figure 4.5: Structure of Parallel Composition Pattern

<sup>13</sup>The difference is that in the Parallelism pattern both paths are executed simultaneously, while in the Decision patterns, a decision is made and only one path is executed.

## Notation

$$C := A \parallel B$$

### Conditioning

$$\begin{aligned} REL_{SYS} \wedge \wedge (INV_C \wedge & (((INV_A \wedge (PRE_A \wedge POST_A)) \\ \vee (INV_B \wedge & (PRE_B \wedge POST_B)))))) \\ := output_C \end{aligned}$$

### Sub-Conditions

$$PRE_C := PRE_A \wedge PRE_B, POST_A \wedge POST_B := POST_C$$

**Issues** Whenever processing is done in a parallel manner, the issue of inconsistency arises. This is true in compositions as well, for example when the processing of one path fails and another is processed correctly. This inconsistency must be compensated through error delegation to the composition or the system.

### 4.6.3 Exclusive Decision

**Name and Classification** Exclusive Decision behavior describes the case where, out of a set of alternative paths, only one is selected and processed further. [ADHF06] The decision in favor of one exclusive path is based on constraints ( $SEL, CH(INOUT)$ ) that must be satisfied by the time of processing. Depending on the handling of the decision to be made this deciding happens in two ways:

- Choice: There are multiple incoming paths and one outgoing path.
- Selection: There is one incoming path and multiple alternative outgoing paths, but only one of these outgoing paths is selected and processed.

### Also Known As

- Choice: Switch, Case
- Selection: Multiple Choice

**Structure** Figure 4.6 illustrates the Exclusive Decision pattern.

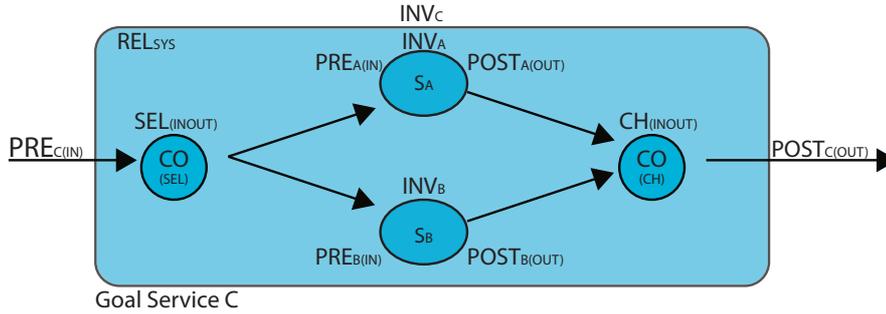


Figure 4.6: Structure of Exclusive Decision Composition Pattern

### Notation

$$C := A \oplus B$$

$$\text{Selection} : \cdot | \oplus$$

$$\text{Choice} : | \oplus \cdot$$

#### Conditioning

$$\begin{aligned} REL_{SYS} \wedge (INV_C \wedge SEL \wedge \\ ((INV_A \wedge (PRE_A \wedge POST_A)) \\ \vee (INV_B \wedge (PRE_B \wedge POST_B)))) \wedge CH \\ := output_C \end{aligned}$$

#### Sub-Conditions

$$\begin{aligned} PRE_C &:= SEL, output(SEL) := input_A \vee input_B \\ CH &:= POST_C, output_A \vee output_B := input(CH) \end{aligned}$$

**Issues** The main difficulty of the Exclusive Decision pattern is to make sure that the constraints are satisfied by exactly one service located on the desired targeted path. If this is not the case, an appropriate reaction (compensation or abortion) must occur on the composition or system level.

#### 4.6.4 Multiple Decision

**Name and Classification** Multiple Decision handling is similar to Exclusive Decision behavior, but includes multiple paths in the outcome. The decision is based on an optional synchronization constraint (*SYNCH(INOUT)*). If no constraint exists, all alternatives are bundled into a single path.

We distinguish three types of Multiple Decision behavior:

- Fork: Splits the composition path into multiple alternative paths that will be processed in a parallel manner, without conditions.
- Join: Brings multiple paths together, including their synchronization.
- Merge: Brings multiple paths together into a single path without synchronization.

#### Also Known As

- Fork: Split, Parallel Routing
- Join: Synchronizer, Rendezvous
- Merge: Aggregator

**Structure** Figure 4.7 illustrates the multiple decision pattern.

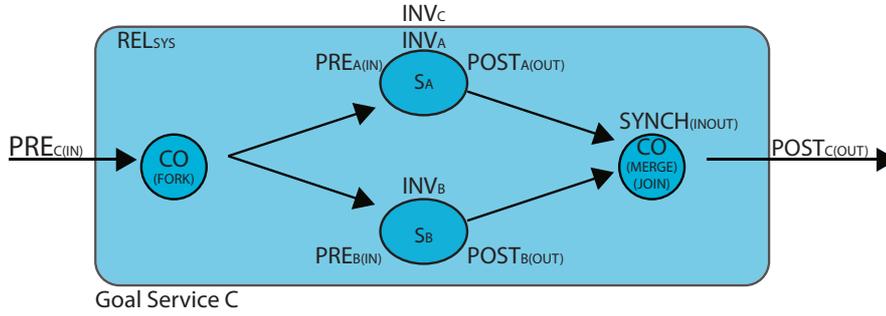


Figure 4.7: Structure of Parallel Composition Pattern

### Notation

$$C := A \otimes B$$

$$Fork : . | \otimes$$

$$Merge, Join : | \otimes .$$

#### Conditioning

$$\begin{aligned} REL_{SYS} \wedge (INV_C \wedge FORK \wedge \\ ((INV_A \wedge (PRE_A \wedge POST_A)) \\ \wedge (INV_B \wedge (PRE_B \wedge POST_B)))) \wedge SYNCH \\ := output_C \end{aligned}$$

#### Sub-Conditions

$$\begin{aligned} PRE_C &:= FORK, output(FORK) := input_A \wedge input_B \\ SYNCH &:= POST_C, output_A \vee output_B := input(SYNCH) \end{aligned}$$

**Issues** Multiple processing inherits the issues of the related patterns parallelism and exclusive decision: inconsistency and constraint availability.

### 4.6.5 Looping

**Name and Classification** The Looping pattern is the construct most commonly used to repeat already defined behavior. The termination of a loop is determined by constraints ( $LOOP(IN)$ ,  $LOOP(OUT)$ ). The Looping pattern is characterized by start and end definitions where the composition path is repeated until the constraints are satisfied.

#### Also Known As

- Iteration, Cycle

**Structure** A loop is illustrated in Figure 4.8.

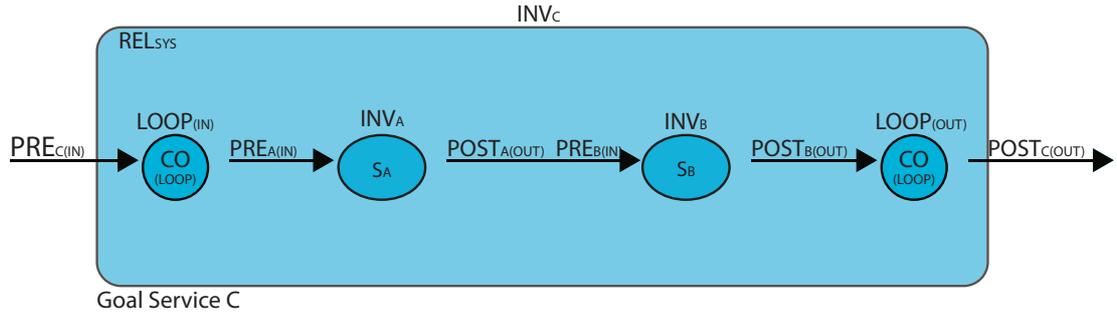


Figure 4.8: Structure of Loop Composition Pattern

#### Notation

$$C := \infty(A, B)$$

Conditioning

$$\begin{aligned} REL_{SYS} \wedge (INV_C \wedge LOOP_{IN} \wedge \\ (INV_A \wedge (PRE_A \wedge POST_A)) \\ \wedge (INV_B \wedge (PRE_B \wedge POST_B)) \wedge LOOP_{OUT}) \\ := output_C \end{aligned}$$

Sub-Conditions

$$PRE_C := LOOP_{IN}, LOOP_{OUT} := POST_C$$

**Issues** The main problem of loops is endlessness or "loopivity", which arises when the loop condition is not available or not meetable. It causes errors in a composition which must be compensated on the system level. For this reason, obviously, it is important not to design loops without loop conditions. Another related issue is the potential cascading of loop constraints, which leads to multi-dimension loops. Both issues imply an immense resource allocation which can ultimately lead to a complete abortion of composition processing.

## 4.7 Non-Functional Properties

---

In general, software solutions are deeply influenced by non-functional properties.<sup>14</sup> These properties provide a way to design, evaluate and redesign a system environment for the targeted building blocks. [Rot07] The specifications of non-functional properties can be divided into those that occur at development time and those that occur at runtime. [RZ03] These properties should be considered even in the predesign stage, because of their enormous effect and also because, due to perpetually arising dependencies, the redesign of non-functional properties is often very complex and time-intensive.

Service composition was introduced in Section 3.5.1 as a phase integrated in the service lifecycle. We also introduced in that chapter the three scopes of distribution. Because of the distributed character of service environments, it is necessary to guarantee that the system is consistent even when services are heavily distributed. To ensure system consistency and quality of services during composing we propose a set of non-programmatic markup demarcations to describe the composition machine entities:

- *Descriptive Attribute*: A general attribute that can be set on a single service. If a service or composition has this attribute, the realization must implement this attribute setting and deal with any side-effects.
- *Descriptive Policy*: A policy with which a service or a composition is marked so as to satisfy non-functional properties.
- *Rely Condition*: A general condition that must be satisfied by the system environment.

---

<sup>14</sup>As mentioned earlier, *non-functional properties* are secondary effects, not related to a specific service, that must be considered when designing composable environments.

In this section, seven non-functional properties are described semantically and evaluated, using criteria based on the work of Dustdar and Schreiner [DS05] and of Sun Microsystems [Sun99], in terms of the properties' implications for composition and their resulting effects.

### 4.7.1 State Management

State relates to a specific point on the composition path. The state can be seen as the aggregation of all data at that point on the path.

To achieve a composition goal, some member services must support the capability of acquiring state information, particularly in the telecommunication domain where we control session, user and media handling. Architecturally, state control requires three stages in detail: initialization, assignment and destruction of involved parameters. [IFT05]

In composition, state is not restricted to member services. It affects the dependencies between members, and thus the entire composition. A failure resulting in the unavailability of one service can result in the unavailability of a large number of other services. For this reason, services should meet the following challenges: [XZH06a]

- *Internal Transparency:* A service should be able to add state information without changing the service contract.
- *External Transparency:* State changes should not affect the external behavior of the composition.
- *State Update and Restoration:* State should have the ability to be saved to a database and then restored after a failure occurrence.

In response to these challenges, certain design implications are evident. Principally, it is important to make sure that services expose their ability to become state-aware. Our solution would be to store this information as a descriptive service feature, in which services that can potentially assume state during composition are tagged `stateful`, while their non-state-capable complements are marked `stateless`.<sup>15</sup> We propose to store this information in the service description, which is the appropriate element for service capability publishing.

---

<sup>15</sup>It is important to note that, to avoid redundancy, the enabling of state information handling should be restricted to atomic services.

## 4.7.2 Transaction Handling

Historically, transactions have been associated mainly with databases, where they are seen as logical operations between participating entities. This characterization of transactions also holds true in service environments, where member services interact to reach goal service state.

It is widely accepted that the basic ACID (Atomicity, Consistency, Isolation and Durability) database concept does not cover the requirements of service-oriented systems. [WY06] Service transactions take place in a more sophisticated environment, involving entities such as the goal service, member services and transitions. With the goal of a composition in mind, certain additional effects of the behavior in a composition must be considered in a discussion of transaction handling: [WY06]

- *Transparency*: As inherited from state management, all composition participants should have a transparent view of the goal of the service.
- *Partiality*: All participants should have the potential to affect transaction state (e.g. partial commit).
- *Long-time Running*: Transactions should be designed to support long-time running compositions without affecting system behavior.
- *Nesting*: Design should include multi-level support for concurrently running transactions.
- *Compatibility*: Transactions must be autonomous and technologically independent.

Transparency and partiality are requirements which have a direct impact on the design of the service itself, within the composition path. Depending on the state of the composition, a member service may act transparently and autonomously. Hence, analogous to state management, we should provide a mechanism to mark atomic services according to their transparency and transaction capability. As suggested in Schlichting et al [XZH06b], we propose to save this information as a declarative attribute in the service contract with the pre-initialized values of `transactive` or `non-transactive`. The reason for this atomic tagging is the potential treatment of transactions in of the machine. A system can guarantee transactional behavior only if it can recognize the transactability of a single service, which therefore must provide this information.

Another aspect of transactions is the consideration of what happens when transaction failure occurs. In conformation with the ACID principles a service state must be saved persistently when successfully committed or, in case of failure, must be restored to a consistent state. During composition, services should move from one consistent state to the other. Failure compensation mechanisms must be handled inside the composition or delegated to the system, because member services can only publish their failure and cannot compensate for it. To detect transaction failures, services that are marked as transactive should also include an attribute for failure handling.

Because transaction handling causes overhead during processing, only transaction-critical services should be marked: services are `non-transactive` by default.

Long-time running and nesting are also designed into the surrounding system and not in the service itself. Long-time running means that transactions take a very long duration time and so involve human attention. [WZZJ05] In case of failure, the system coordinates which participants should perform compensation. Nesting should be compensated similarly.

### 4.7.3 Security and Trust

Security and trust are basic mechanisms to protect service environments and ensure system integrity. We can identify six main categories of service security and trust mechanisms: [Yee06]

- *Authentication*: determines the user's identity
- *Authorization*: determines the user's permission regarding access to resources
- *Access Control*: determines the access of the user depending on his authorization - in other words, the enforcement of authorization
- *Data Confidentiality*: protects the data and makes it accessible only to permitted users
- *Data Integrity*: manages the reliability of data, so that user and provider have the same understanding of the data
- *Non-repudiation*: establishes trusted behavior through predefined parameters of identification defined between user and provider: a "handshake" where the user has to bring proof of identity and the provider has to prove delivery of the service

Trust enables a relationship between interacting services. Services can implement security and trust through policy management. Policies define rule sets to initialize interaction behavior. They have traditionally been used to satisfy security requirements between networks and distributed hardware. Recently, they have begun to enter software-realized layers, including service environments.

Policies can be attached to atomic or composed services. Authentication, authorization and access control are applicable to both constructs. Because of their related behavior, we propose to collect this information in a policy called `Security`, included in the entity description.

#### 4.7.4 Quality Measurement

Services, especially in the context of a composition, are highly abstracted, loosely coupled software building blocks. But every abstraction that hides implementation details must be resolved and translated into underlying implementation specifics that are closer to machine-understandable language. Especially in multi-user environments such as telecommunication, performance suffers from these multi-level abstractions.

To ensure a predictable level of quality in terms of service composition we distinguish two observable characteristics:

- *Performance*: a quantifiable service parameter describing behavior at runtime, short-term focused and in most cases related to latency measurements of atomic and composed services
- *Composition Quality*: a quantifiable and qualifiable parameter that guarantees long-term quality for the whole composition machine

In the context of these characteristics, the following properties should be considered:

- QoS parameters of the underlying network (described in Section 2.5)
- General quality properties of a service environment
- Service-specific properties of atomic and composed services

To facilitate the analysis of performance, we suggest creating an attribute for atomic and composed services, to trace the runtime parameters and react

appropriately. The concrete quantifications should be defined in the framework, where a performance service becomes implemented as a utility, usable for other services.

To assess the quality of a composition we have already suggested three basic network constructs, delay, jitter and loss, which must be transformed into traceable service attributes. Furthermore, we need to define service-specific parameters that can be observed at runtime.

Both characteristics, performance and composition quality, have a profound influence on composition and system reliability. For this reason, we must find quality metrics that will determine the thresholds of runtime-measured attributes. If metric measurements violate thresholds, an appropriate reaction must be specified.

### 4.7.5 Errors

In service composition, errors arise in the entities that are part of the composition machine. The root cause of an error is most likely to occur when processing member services in a composition. Failures should be detected immediately and the system must react appropriately.

We distinguish three main categories of reaction:

- *Compensation*: The error becomes fully compensated and the composition path can continue processing without using alternatives.
- *Alternative*: The error is compensated using an explicit defined alternative path.
- *Termination*: The complete composition execution must be interrupted.

There are two ways to establish error handling in a composition:

- Errors are compensated within the service. The implementation is specified in the service contract.
- Errors are handled for the entire composition, with the option of declaring an alternative path. We propose to specify this option in the composition contract.

### 4.7.6 Invocation

Services, whether atomic or composed, become invoked in the context of a reachable goal: yet another service, which is therefore also invocable. As we noted in our discussion of the service paradigm, services are invoked by service consumers, which are machines or humans. One of the primary potential benefits of composition is its automation. According to the level of automation versus human intervention, we can define the following types of invocation in the service contract:

- *Manual*: Invocations are declared manually and rigidly processed without parameterized alternative handling.
- *Semi-automatic*: Invocations are partly automated: Control flow is created by humans, but intelligently controlled by the composition environment.<sup>16</sup>
- *Automatic*:<sup>17</sup> Control flow is created dynamically based on the parameter of a human-specified goal. As a result, service invocation is fully automatic when processing the created composition.

Only semantically well-described service structures can be invoked. The descriptions require semantics that will be introduced later on, in a discussion of the application of functional service properties in Chapter 6, where we will propose a semantic domain service taxonomy. It is also necessary to design into the system the ability to handle an automatic invocation. This implies automatic matchmaking and lookup in the composition context.

---

<sup>16</sup>For example, automatic alternative handling by increasing thresholds during QoS processing.

<sup>17</sup>Dynamically composed services invoke their member services in a fully automated manner.

### 4.7.7 Communication

Communication is the property that addresses the manner of interaction between service consumer and provider. We distinguish between synchronous and asynchronous communication modes. The two modes have different requirements in terms of timeliness, which requires that we give them an almost completely different technical realization.

Synchronous communication describes a timed dependency in which the processing of one service must be completed before the succeeding service can be processed, and the subsequently processed service relies on the result of the previous one. [TvS02] In contrast to this behavior, an asynchronous communication scenario implies immediate processing without dependency handling between services. The asynchronous mode has no time-critical restrictions, and in this mode processing can be marked (e.g. buffered or queued) for later execution.

Ruh [RMB00] suggests the establishment of technologically adaptable templates or patterns for communication modes:

- *Synchronous Communication*, service-dependent processing:
  - *Request/Reply*: The basic template. The subsequent request relies on the results of the preceding one, allowing for a short idle time that is consumed in processing,
  - *One-Way*: The subsequent service relies on the arrival of a service-processing confirmation.
  - *Polling*: The subsequent service continues processing but uses interval checking (polling) for confirmation control.
- *Asynchronous Communication*: service-independent processing:
  - *Passing*: Communication is addressed (passed) directly.
  - *Publish/Subscribe*: Multiple subscribers receive communication from a publisher. The publisher-subscriber assignment is determined by subscriber preferences (e.g. keywords, time) based on a subscription list of receivers who have subscribed with a broker.
  - *Broadcast*: The communication is addressed to all receivers involved in it.

In the context of composing services, we can determine the appropriate communication mode by considering two factors: the service type, which is a

semantic criterion, and the goal that the composition is intended to achieve. The former criterion is a service attribute, extractable from the quality parameter of a service. For example, for an audio service that contains a quality parameter of maximum 100ms delay and a strong dependency between neighboring services, it is appropriate to communicate using the synchronous request/reply model.

Observing the entire composition path, communication models can be mixed to achieve the goal state. The underlying system should enable the processing of both communication modes. In the case of communication error or unavailability, the system must provide an appropriate reaction.

### 4.7.8 Summary of Composition Requirements and Implications

Table 4.1 assigns the major design implications examined in this section to the entities where they must be considered.<sup>18</sup>

---

<sup>18</sup>The specification in Table 4.1 uses the following system of abbreviation: CompositionEnvironmentRequirement[CER], CompositionEnvironmentIssue[CEI].Composition[Co].NUMBER.

Specification	Name	Machine entity root cause	System Implication
<b>CER.Co.1</b>	<i>State Management</i>	Atomic Service	The system must synchronize and recover state information.
<b>CER.Co.2</b>	<i>Transaction Handling</i>	Atomic Service	The system must enable the handling of transactions.
<b>CER.Co.4</b>	<i>Quality Measurement</i>	Atomic, Composed Service	The system must be able to check and react to performance and quality metrics.
<b>CER.Co.5</b>	<i>Error Handling</i>	Atomic, Composed Service, System	The system should track atomic and composing errors and react appropriately.
<b>CER.Co.6</b>	<i>Automation</i>	Composed Service	The system must be enabled to invoke services automatically and semi-automatically. Enabling of automatic matchmaking.
<b>CER.Co.7</b>	<i>Communication</i>	Atomic, Composed Service	The system must be able to transfer data synchronously and asynchronously among all involved composition parties.
<b>CEI.Co.1</b>	<i>Circularity</i>	Atomic Service	The system should control heavily recursive service calling.
<b>CEI.Co.2</b>	<i>Redundancy</i>	Atomic, Composed Service	The system should track redundant use of functionality.

Table 4.1: Composition Environment Requirements and Issues

## 4.8 Syntax Formalization of Service Composition

Based on the discussions in this chapter of the syntax of a service description, information storage, and service attributes, we can denote a composition as outlined in Figure 4.9.

```
1 SERVICE-COMPOSITION(C)
2 (
3   (*identification and dependencies*)
4   [METADATA];
5   (*global policy agreements*)
6   {Policy}-;
7   (*global processing constraints*)
8   {CONDITIONS
9     [INV],
10    [PRE(IN)],
11    [POST(OUT)]}-;
12  (*global attributes*)
13  {ATTRIBUTES}
14  CONNECTOR('START'); (*start connector*)
15  TRANSITION
16  CONNECTOR
17    ('SEL' | 'CH' | 'FORK' | 'MERGE' | 'JOIN'
18     | 'TRANSFORM' | 'LOOP' | 'ERROR');
19  TRANSITION
20  SERVICE (*sequence of member services, connectors. transitions*)
21    (?...?);
22  TRANSITION;
23  CONNECTOR('END'); (*end connector*)
24 )
```

Figure 4.9: Syntax Formalization of Service Composition

In the preceding composition formalization, a service is expressed by the the special sequence (?...?). A single service description can be formulated as follows:

```
1 SERVICE(S)
2 (
3   (*identification*)
4   [METADATA];
5   (*local policy agreements*)
6   {POLICIES}-;
7   (*local data flow*)
8   {Parameter
9     [IN],
10    [OUT],
11    [INOUT]}-;
12  (*local attributes*)
13  {ATTRIBUTES}-;
14 )
```

Figure 4.10: Syntax Formalization of a Composition Member Service

## 4.9 Summary

---

In this chapter we introduced the pillars of service composition, taking into account the dynamic nature of composition. Composition aggregates certain elements, transitions, services, and connectors, and acts within the composition machine, which consists of service descriptions (or service contract), the composition description (or composition contract), and the encompassing system. The composition description is a human- and machine-readable, understandable and traceable logical construct with strong dependencies within the composition machine. The composition machine includes the relationships of service to composition, composition to system environment, and service to system environment.

To picture the basic behavior during composition, certain significant patterns were outlined and a basic notation for them was introduced, which is summarized in Figure 4.11.

- Sequence:  $\triangleright$
- Parallelism:  $\parallel$
- Exclusive Decision:  $.\mid\oplus$  Selection and  $\mid\oplus.$  Choice
- Multiple Decision:  $.\mid\otimes$  Fork and  $\mid\otimes.$  Merge, Join
- Loop:  $\infty$

where a composition example could be denoted as:

$$output(S_N) := S_1 \triangleright (.\mid\oplus S_2 \triangleright (S_3, S_4), S_5 \triangleright (S_6)) \triangleright (\mid\otimes .S_7)$$

Figure 4.11: Summary of Notation for Composition Behavior

Non-functional properties were described, and their design implications were assigned to the entities where they must be considered. Finally, a notation was introduced for the composition of multiple services.

In the next chapter, we will investigate related work, which deals with the framework-based service composition of telecommunication services, moving from abstract formalism to industry-proposed approaches.



# Chapter 5

## Research Scope and Related Work

*“It’s a very difficult job and the only way to get through it is we all work together as a team. ”*

*Charlie Croker, The Italian Job [1969, 2003]*

### 5.1 Synopsis

---

This chapter investigates current industrial and scientific work relevant to the aims of this thesis.

In the challenge to embed flexible, agile solutions in a highly sophisticated telecommunication domain, we can borrow from several telecommunication standardization bodies as well as from the rapidly growing software community. Both communities introduce theoretical concepts and implementations that can lead toward a homogeneous system architecture based on a frameworking methodology.<sup>1</sup> This chapter’s investigation of existing approaches focuses on composition solutions that can be embedded in a framework to achieve telecom service agility.

To systematize the analysis, we divide the approaches under investigation into four subcategories:

---

<sup>1</sup>According to The Balanced Scorecard Institute, frameworking was established to aggregate “logical structures for classifying and organizing complex information.” [Ins06d]

- *Composition Essentials*: scientific and industrial solutions that do not use frameworks but do integrate service composition
- *Telecommunication Network Frameworks*: framework solutions that are telecommunication community-driven and propagate all planes of the NGN reference model, including transport and aggregation mechanisms
- *Telecommunication Software Frameworks*: framework approaches that are telecommunication community-driven and primary technical, and propagate the three software-dominated NGN layers
- *Software Frameworks*: software engineering solutions, primarily technical, that are functionally completely domain-independent

Business and technical perspectives are discussed here in the light of their relevance to the framework design and prototype implementation presented in subsequent chapters, and this consideration formed the criteria for selecting the work to be investigated.

After presenting the related work, we provide a comparative analysis of the existing approaches and discuss how gaps in these approaches create challenges for our framework design.

## 5.2 Research Scope

---

To establish criteria for evaluating related work, it will be helpful to have in mind the scope, assumptions, and basic composition focus of this thesis, which limit the scope of related research to be considered.

### 5.2.1 Domain Boundaries

As described in Section 1.6, the framework proposed in this thesis adapts the reference model of NGN, which describes five functional planes, their components and inter-plane dependencies.<sup>2</sup> The related work presented in this chapter is considered in the light of that base structure and the assumptions that follow from it.

---

<sup>2</sup>NGN was described in detail in 2.4.

Complementary to the static view of the NGN reference model, the lifecycle model of the software service paradigm describes the core software entities (consumer, broker and provider) and their activities<sup>3</sup> While the NGN model provides the skeleton of an approach, the service paradigm adds the necessary design guidelines and lifecycle phases to describe the main structures.

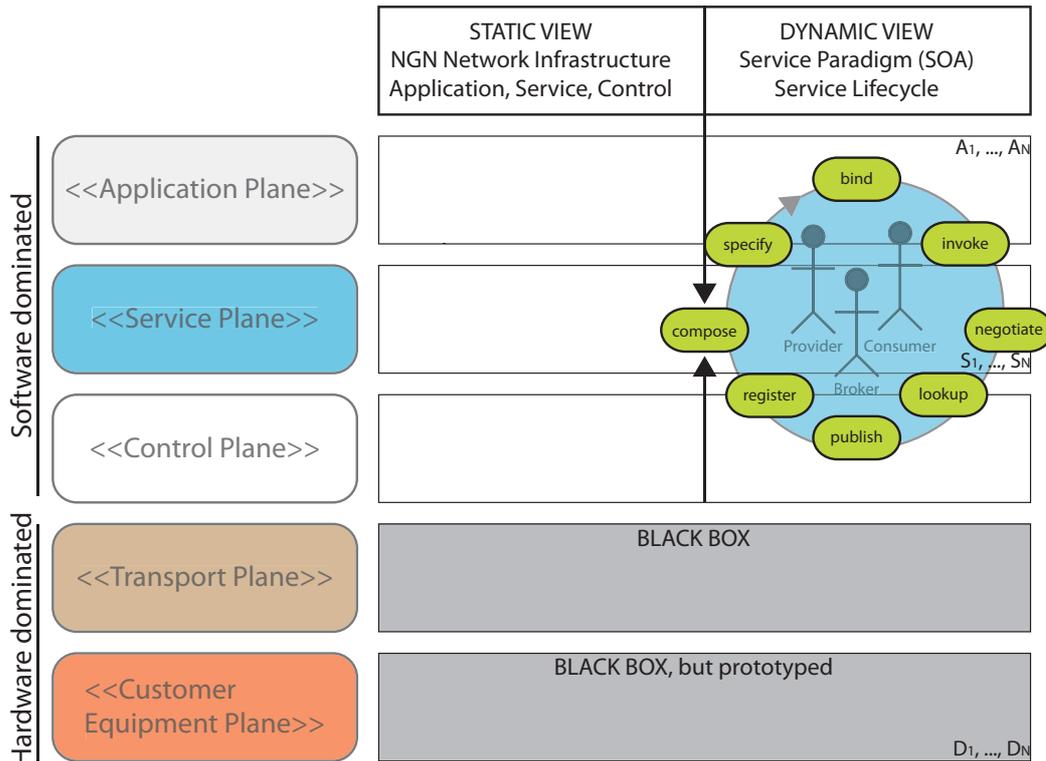


Figure 5.1: Research Scope

Based on the static and dynamic views shown in Figure 5.1, we can define the following research boundaries:

- To apply the service paradigm and SOA to the NGN structures, only software-dominated planes will be considered. In other words, the research will concentrate on the application, service and control layer construction.
- The transport plane will be treated as a black box. We assume that it is possible to wrap all of the transport functionality transparently in a software-based network abstraction layer which can emulate all

<sup>3</sup>These are described in Section 3.2.3.

network-specific transport functions, including signaling and media delivery.

- The CE layer will also be treated as a black box.<sup>4</sup>
- The software service paradigm is seen as a continuous design schema, technically adaptable to the application, service and control layers.
- The lifecycle of services will be considered only in regard to development activities.<sup>5</sup> Related activities such as service governance or maintenance are not part of the analysis.
- The services developed in this research will be based on the following assumptions:
  - Independence: There is no direct dependency between services.
  - Uniformity: Elements are defined in a semantically equal manner. For example, a QoS delay of 5ms defined for a given service will be a constant for all instances of that service.
  - Equipartition: When processing in a parallel manner and without decisions, there is an equal probability of aggregating or splitting services.
  - Security and trust operational readiness: The declarative values of composition elements are correct. The system supports both security and trust with built-in mechanisms.
  - Error handling operational readiness: Errors and exceptions are declared within the composition and compensated appropriately by the system.
- In our prototype, we focus on implementations using Java technology, which are particularly well suited to the requirements of the framework prototyping because of their openness, variety and industry-wide propagation and acceptance.

---

<sup>4</sup>However, it will be prototyped in this thesis.

<sup>5</sup>As described in the concluding model of Section 3.5.1.

## 5.2.2 Composition Focus

As we have seen in previous chapters, service composition is one of the key concepts on which this thesis is based. In the NGN reference model, the three software-dominated upper layers (the application, service, and control layers) are all adaptable to composition activity, but they address different stakeholders and thus generate certain efficiencies during composition. To determine the most efficient applications of service composition, it is useful to consider how each provider's<sup>6</sup> interest affects agility issues.

We can distinguish three roles involved in the development of services within the providers' networks. Each role approaches service composition differently.

- Business analyst: determines the non-technical, business-related service representations, and has a domain process-related background
- Software architect: translates a business analysis into technical requirements using an overview of the infrastructure, and has a combined domain and technical background
- Software developer: technically implements the service functionality, and has a specialized technical background

In the light of these three roles and their interests, we suggest three qualities to use in assessing service composition efficiency:

1. It is important, first, to assess the *type of service*, which will be either a domain or a utility service.
2. Secondly, consideration must be given to the *manner of defining the flow of data and control*. We distinguish three kinds of control flow development, which primarily affect composition declaration. *Imperative* and *declarative* composition handling represent different programming paradigms and each has its own characteristics.<sup>7</sup> Beyond these two programming-derived concepts, a *visualized* composition implies a graphical user interface (GUI) to define the flow of services. Imperative composition usually integrates a proprietary, non-human-readable, machine-interpretable storage format, while declarative and visual composition use a more abstract, human-readable and machine-processable format.

---

<sup>6</sup>Here we are referring to the telecommunication service provider, as defined in Section 1.2, not the logical construct of the software service provider.

<sup>7</sup>The characteristics are elaborated in Lloyd [Llo95] and Broy [Bro91].

3. The third quality we would examine is the *reduction of human intervention* or automation of the composition invocation.<sup>8</sup>

Based on these three criteria, we propose that telecommunication services should be propagated in each of the three planes and implemented using cooperating (i.e. software) components. The design of the components should be carefully planned using quantifiable criteria, to fit requirements and to avoid over-engineering with its inherent performance problems, implementational redundancies, and escalating costs.

We have previously noted<sup>9</sup> that the central service layer is highly adaptable to all kinds of service representations, differentiable into domain and utility types. Surrounding the service layer, each of the two adjacent layers also has an optimally efficient use of service composition techniques. We suggest the following distribution of composition techniques to composition layers:

Layer	Service Types	Composition handling	Automation
<b>Application</b>	Domain Services	Visual and Declarative	Manual and Semi-Automatic
<b>Service</b>	Domain and Utility Services	Declarative and Imperative	Semi-automatic and Automatic
<b>Control</b>	Utility Services	Declarative and Imperative	Automatic

Table 5.1: Efficient Use of Service Composition in Telecom Infrastructures

It is a central proposition of this thesis that the most effective way to harness agility in a telecom infrastructure is to employ a semi-automated, declarative composition of domain services on the service layer. We further argue that the composition of the utility services underlying the domain services must be fully automatic and transparent to the domain composition.

Having narrowed the scope of our research, we can now proceed to examine the published work that most closely fits that scope.

<sup>8</sup>In Section 4.7.6 we defined three levels of composition invocation: manual, semi-automatic and automatic.

<sup>9</sup>In Section 2.4.4.

## 5.3 Composition Essentials

---

Service composition is a broad field encompassing many continuing research activities. Our evaluation begins with approaches that were developed without the integration of a framework infrastructure or a domain-related background. Beyond this starting position we can observe two research directions: first, composition approaches applying a particular scientific formalism<sup>10</sup> to composable structures, and secondly, solutions that were inspired by explicit functional and non-functional software engineering requirements. The latter focus mainly on a particular technical component or implementable aspect. A typical representative of this research is the development of descriptive language dialects, design- or run-time based, and optionally processable in an execution environment. Both discussions enrich our framework development.

### 5.3.1 Formalism-Based Approaches

Formalism-based discussions of composition have been held in the context of software engineering when discussing business process-related software aspects such as workflow or business process management. [Aal03] [LRS02] Emerging from this discussion, the term Process-Aware Information Systems (PAIS) has been termed to describe the group of business application efforts that includes Customer Relationship Management (CRM) or Enterprise Resource Planning (ERP). Recently, researchers have begun to adapt formalistic approaches to leverage service architectures and, in particular, their composition capabilities.

#### Petri Nets

General examples of applied formalisms are apparent, for example, in the work of Hamadi and Benatallah [HB03], where services are composed using Petri nets. Developed by Carl Adam Petri in his 1962 progressive dissertation "Kommunikation mit Automaten" Petri nets offer a connected, bipartite representation of distributed systems in graphical and algebraic notation. Based on a well defined set of elements (mainly places and transitions), Petri nets analyze the structure and connectivity of interacting entities. Their graphic

---

<sup>10</sup>Formalisms are techniques or methodologies that approach the design or improvement of software systems using algebraic or graphical notation.

representation is especially well suited to support the design, specification, simulation and verification of systems. [Jen98] By extending the concept of colored Petri nets, it is possible to outline communication systems and illustrate data flow issues in conversations. [CCF+00]

### $\pi$ -Calculus

Further work has been done to optimize service composition through the use of  $\pi$ -calculus. [PSWW04] The  $\pi$ -calculus methodology suggests a naming format to analyze concurrent computing issues. The main components of  $\pi$ -calculus are tasks, represented as  $\pi$ -calculus processes, and dependencies linking the tasks, represented by  $\pi$ -calculus channels. [PSWW04] Pairs of processes interact by exchanging messages through input and output channels. Beyond its algebraic notation, by facilitating the definition of channel structures  $\pi$ -calculus offers mobility of entities in the context of interacting systems where the user has free choice of communication access. It is possible that NGN could be used as the system environment in a  $\pi$ -calculus approach.<sup>11</sup>

### Finite State Machines

Another example of applied formalisms that merits attention is the Finite State Machine (FSM).<sup>12</sup> The primary strength of FSM is its analysis of abstract software system states and the state-changing transitions between them. FSM can be illustrated both algebraically and graphically. FSM provides a set of pre-defined states which refer to a given system and thus offer the ability to apply dynamics through operations. Both system states (statics) and operations (dynamics) can be formally described using mathematical clauses.

---

<sup>11</sup>The semantic and syntactic concept has been elaborated in Robin Milner's tutorial. [Mil91]

<sup>12</sup>In software engineering, the term Abstract State Machine (ASM) is used synonymously.

## Other Formalisms

There are many other formalism-based approaches that we considered because of their ability to optimize composition computing. A few examples are logic programming, linear logic, AI Planning, and Markov decision processes. A non-exhaustive list of potential formalisms, their methodology and their tool support is published under <http://vl.fmnet.info> and currently contains 93 items. Unfortunately, there is still very little information about the results of applying these formalisms in a composed software service system.

### 5.3.2 Software Engineering Entities

A second direction for basic composition research has been the focus on software implementations through the transformation of functional or non-functional requirements into software building blocks.<sup>13</sup> There are five main types of software entities:

- *Communication Protocols*: conventions or standards that describe potential paths of communication between two endpoints or software components. In this thesis we have focused on IP-based implementations.
- *Middleware*: runtime environments, defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system. An example is application server technology.<sup>14</sup> Middleware uses APIs to provide a higher level of abstraction for programmers. [Bak01]
- *Descriptive Programming Languages*:<sup>15</sup> languages that utilize a human- and machine- readable format, using a tree or key-value structure. Examples are XML and its derivatives. No compilation is necessary for processing in the software environment.
- *Compilable Programming Languages*: languages that perform computing tasks in an imperative manner. Language code is compiled into

---

<sup>13</sup>The primary software building blocks are components, objects and services.

<sup>14</sup>The term *app server* is used synonymously. We view the app server as an architectural software component, functionally referring to the service or control layer.

<sup>15</sup>Although descriptive languages are a type of programming language, we consider them separately so as to focus on their expressing capability.

non-human-readable code<sup>16</sup> and executed in a runtime environment. Java is an example.

- *Application Programming Interface (API)*: a collection of programming code implemented using one or more programming languages and addressing a given functionality. Examples are Java Telephony API (JTAPI) and Java Business Integration (JBI). Commonly, APIs have been based on compilable programming languages, but more recently they tend to be based on the declarative interfaces, as in Parlay X web services. It is possible to distinguish between low-level APIs, which work with underlying abstractions such as strings or char, and high-level APIs, which use more advanced programming abstractions such as objects and services.

### 5.3.3 Research Relevance of Formalism Approaches

The use of formalisms tends to require considerable expertise, and they are generally deficient in the transformation of analyzed notation into practically usable implementations, leaving a gap between theory and implementation that complicates any practical work with formalism. To reduce this gap we might either reduce the complexity of the paths between specification and code or implement the system and verify its correctness against the model. However, the precisely accurate character of formalisms makes them highly appropriate for the description of coherences in an abstract context not focused on implementation. In particular, formalisms simplify the specification and verification of a modeled software situation by reducing complexity and allowing the simulation of behavioral alternatives. Some issues may be clarified through their abstraction into algebraic or graphically represented formalism. This aspect of formalism is potentially useful when designing and verifying software systems.

Petri nets show their strength when examining the flow of services in general and producing a behavioral overview of interaction in distributed systems. Using Petri nets, data and control flow can be illustrated either graphically or algebraically. [HB03] Critics of Petri nets note that they are not valid in terms of the principle of composability, which requires that the system provides recombinant components that are self-contained and stateless. However, Petri nets offer a very mature, integrated simplification of service behavior. Although techniques and tools have been developed to close the gap

---

<sup>16</sup>This is also known as *byte code*.

between formalism and practicability, Petri nets will be used more in simulations than in concrete programming environments. An interesting example of service composition analysis is the Petri net-based pattern notation of the Business Process Execution Language (BPEL).<sup>17</sup> [Sta05]

The strength of  $\pi$ -calculus is in its dynamic, transitional detail. Composed services can be seen as  $\pi$ -calculus processes exchanging messages through channels. The simulation of those processes allows the automatic detection of potential deadlock or livelock situations, and  $\pi$ -calculus's concept of intercommunication channels permits inferences about non-functional properties in relation to system environment design, for example in transaction analysis through  $\pi$ -calculus notation. Wil van der Aalst presents an interesting comparison of the usability of Petri nets and  $\pi$ -calculus. [Aa004]  $\pi$ -calculus influenced the development of declarative composition languages such as the Business Process Modeling Language (BPML) and XLANG by Microsoft, whose concepts were used in the BPEL standardization.

Finite State Machines (FSM) are particularly well adapted for use in clarifying state management. Because of their abstraction of system states, they are an appropriate tool to use in analyzing services (interpreted as machines) and their conditions and related parameters. Influenced by the research of Yuri Gurevich, the concept of FSM was used in the development of the Abstract State Machine Language (ASML). The language is directly integrated into the Microsoft development environment and can be used to verify the design of an application before coding or to generate test cases. FSM can be used for direct simulation of service composition in web service environments and even in implementations of particular telecommunication issues. [GK97]

The three most prominent candidates in composition-related formalism have their primary advantage in an abstract simplification of composition behavior, which reduces complexity when designing and simulating the control and data flow of composed services. They provide the capability to validate the well-formness of a composition and its included member services. Measurable aspects are potential deadlock or livelock situations of members and constraint compatibility of members in respect to the composition description. In this thesis, behavior patterns and data and control flow are derived from results borrowed from formalism-based research.

---

<sup>17</sup>Because of BPEL's origin in the web service framework, WS-BPEL or BPEL4WS are used synonymously.

### 5.3.4 Research Relevance of Software Engineering Approaches

Communication protocols are a technical pillar of telecommunication service development and provide a wealth of approaches. Based on the seven-layer, partitioned Open Systems Interconnection (OSI) reference model, we can posit an IP protocol stack (illustrated in Figure 5.2) formulated to carry the telecommunication traffic in NGN's. Like the NGN model introduced in Chapter 2, the OSI model represents functional planes in a useful manner. The Internet Protocol (IP), because of its ability to carry packets<sup>18</sup> without any individual transmission setup, offers the potential to establish connections statelessly. Request-response messages are exchanged without an exclusive connection between transmitter and receiver, which address each other using unique, device-assignable addresses (IP addresses). The packets used usually consist of two parts, the header and the payload. The header declares meta information in a key-value format, while the payload carries the actual service content.

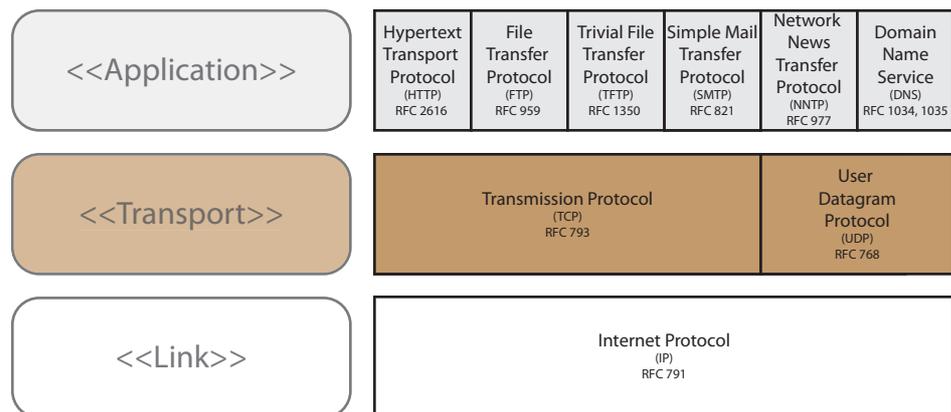


Figure 5.2: Basic Internet Protocol (IP) Stack

Middleware approaches, largely domain-independent, are driven by the software engineering community. From Bakken's middleware classification [Bak01] we have selected two approaches that are of particular interest. The first presents an extended Message Oriented Middleware (MOM) in a solution called the Enterprise Service Bus (ESB). The ESB combines, in a loosely coupled, distributed manner, MOM features such as routing, transaction handling, persistence and message queuing with SOA-derived ideas such as heterogeneous connectivity, service mediation and even service composition.

<sup>18</sup>The term *datagrams* can be used synonymously.

Composition capabilities are mainly restricted to utility functionality, providing asynchronous services. The reason for this restriction is the ESB's inherent publish-subscribe communication character and its less scalable, centralized, hub-and-spoke architecture. The investigation of this middleware's potential use in telecommunication scenarios is still in progress, and ESBs should certainly be considered when designing service architectures. Prominent, industry-driven ESB implementations are the Progress Sonic ESB Product Family (based on the MOM implementation Sonic MQ) and the IBM WebSphere Enterprise Service Bus (based on the MOM product MQ Series).

Another significant middleware engine is represented by application servers<sup>19</sup>, an extension of the concept of transaction machines. Like containers, they interact as programmable runtime environments for developed application modules. Functionally arranged above the ESB's, app servers are well suited for processing business functionality because of their application layer protocol stacks. For example, a common scenario would be a Java-based app server that consists of an HTTP-enabled web container and a programmable API to interface with the HTTP protocol. Java-based app servers were developed largely to provide non-functional properties, and like ESBs, they include state management, transaction handling, security and trust, and service composition. Their applicability to telecom service platforms led to their inclusion in the development of Service Delivery Platforms (SDP's). Current examples of this middleware category are the Java server JBoss, BEA's product line Weblogic, and the Microsoft .NET framework.

Descriptive and compilable programming languages with practical application in the field of telecommunication are usually based on C, C++, Java or C#. In most cases the declarative development is done using eXtensible Markup Languages (XML). Today's service composition solutions tend to use a separate composition description implemented in a descriptive language that specifies interconnecting member contracts, while the contract implementations are coded using compilable languages. The special focus of this thesis lies in the composition description languages, mainly approached through the web service framework, exposing composition capabilities using XML structures.<sup>20</sup>

Another useful adaptation of descriptive languages is the use of specification languages for the definition and configuration of non-functional properties in software ecosystems. A good example of such an approach is CQML+

---

<sup>19</sup>Application servers are usually referred to here and elsewhere as *app servers*.

<sup>20</sup>Related details are provided in the web service framework discussion in Section 5.6.1.

[RZ03], a language extension of CQML, which is a specification language to describe QoS offers and requirements in component-based systems. [AE02] CQML+ offers four points of improvement over the current handling of non-functional properties: an advanced computational ontology, resource clauses, structured characteristics, and explicit dependencies. In the literature, researchers describe the transformation of CQML into an XML representation, which facilitates the integration of CQML into existing software environments. [RZ03]

For the sake of clarity, we will mention two strategies used when implementing the accessibility of technical entities: interfaces and reference points. Interfaces semantically define the interaction, independently from the service requester. Reference points, on the other hand, are dependent on the requester, explicitly enumerate the end points of the interaction, and do not necessarily require an interface. [A1106]

In general, software service composition is used by several interacting software entities. Current solutions tend to use declarative programming languages to define service compositions, which are processed at runtime, invoking their member services. Interconnection and data transfer are arranged using IP-based protocols. Implementations to be bound on contracts are based on compilable programming languages, which can be bundled into functional API's and embedded in software engines. Conceivable composition runtime engines might be ESB's or app servers, which could potentially collaborate within the software infrastructure.<sup>21</sup>

## 5.4 Telecommunication Network Frameworks

---

Telecommunication Network Frameworks (TNF) are a class of solutions driven by the telecommunication community and propagating all functional planes of the NGN reference model.<sup>22</sup> Treating telecom services<sup>23</sup> as central build-

---

<sup>21</sup>When discussing this sort of potential for applicability, we will use the term *point of applicability* to mean a point in the software architecture where service composition can be applied efficiently.

<sup>22</sup>The model is presented in Section 2.4.4.

<sup>23</sup>The TNF understanding of telecom services differs significantly from the technical service implementations that are a focus of this thesis. This difference was explained in Section 3.2.2.

ing blocks, they structure service-oriented software and hardware elements from applications to the transport level, starting from the CPE. The value of these frameworks can be seen in their telecom domain-specific architecture and their deep, mature solutions for business and technical requirements interpretation.

### 5.4.1 IP Multimedia Subsystem (IMS)

The IP Multimedia Subsystem (IMS) was introduced by the Third Generation Partnership Project (3GPP), which was originally formed to specify a third generation of cellular network platforms. The primary purpose of IMS was to overcome circuit-switched limitations through its detailed solutions to interoperator roaming, its differentiated QoS and its service-independent charging (billing).

The current implementation efforts<sup>24</sup> are a collaboration between 3GPP, which developed the architectural framework<sup>25</sup>, and the Internet Engineering Task Force (IETF), which contributed the base technology and protocol specifications. Their vision was the unification of the cellular and fixed line worlds, both burdened with circuit-switched and packet-based transport as well as a broad landscape of legacy software and applications. With aims largely similar to those of TINA (though, unlike TINA, originating from a mobile background) IMS faces a threefold target: predictable QoS support, service-independent charging (billing) and the integration of heterogeneous service environments while still allowing the production of new ones. [GC06]

Access to the IMS system can be done directly using existing provider capabilities (such as packet-switched IP-enabled technology). Communication is implemented by an IP-extended protocol stack, producing one IP extension per function collection. For example, AAA functionality can be addressed using the Diameter protocol.

Driven by the very technically oriented community of 3GPP, IMS is specified in terms of functions arranged according to an information system view. The development of an explicit business view is consigned to the framework implementer. This is also true of the semantic definitions of services. The aggregated functions of IMS are linked by standardized reference points and refined into technical representations. IMS can be illustrated using visual

---

<sup>24</sup>As of this writing, the latest release of IMS is version 7.

<sup>25</sup>The 3GPP specifications are organized into Technical Reports (TR) and Technical Specifications (TS).

nodes to represent aggregated functions.<sup>26</sup>

To keep our terminology consistent and to set IMS in the context of this thesis, its aggregated functions can be seen as technical telecom services on the application, service and control planes of the NGN reference model. IMS's visual nodes describe its functionality and offer connectivity through standardized links. The nodes, technically service functions, and the links, which are technically reference points, are connected on the application and service levels with an interface to the central control functionality.

Figure 5.3 simplifies the structure of the IMS core system. Keeping in mind the research scope of this thesis, we limit our observation to the nodes and reference points involved in the development and provision of telecom services. The surrounding nodes, handling interconnectivity (transport emulation) and data persistence, are omitted.

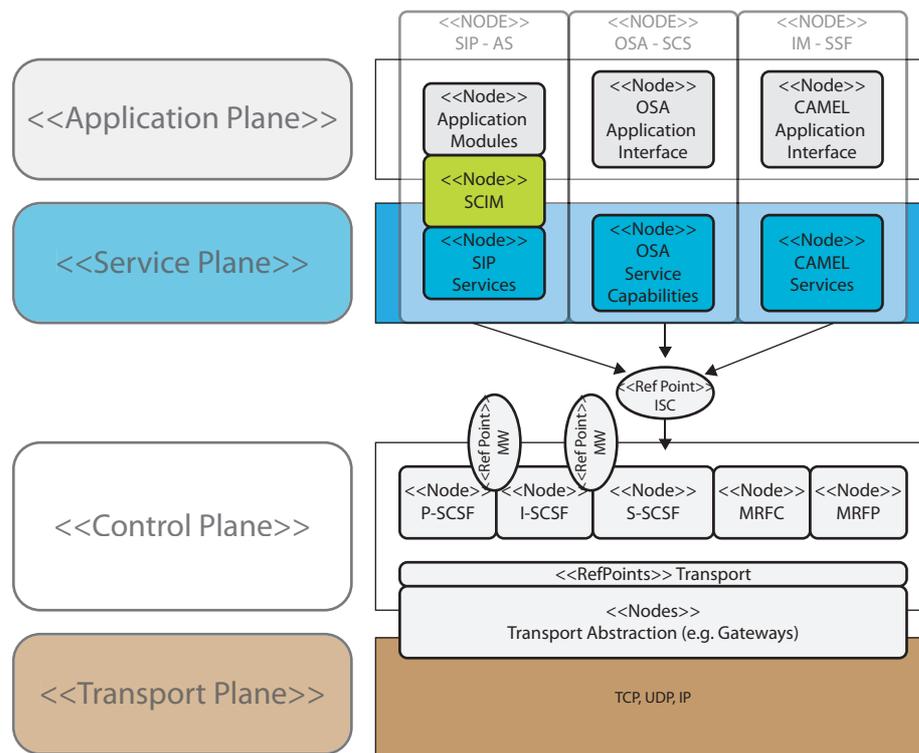


Figure 5.3: IP Multimedia Subsystem (IMS)

<sup>26</sup>The IMS collaborators specify functions, not nodes, which affects implementations where functions are embedded into one or more physical nodes.

## IMS Visual Nodes

IMS service nodes can be divided into three categories: signaling functions, media delivery functions and application server functions. Depending on its responsibilities, a service is signaled (signaling), processed (app server) or delivered (media delivery).

**Signaling Nodes** Signaling nodes are essential and mandatory in the IMS system. They are collectively known as Call Session Control Functions (CSCF), though they are divided into the categories of Proxy-CSCF (P-CSCF), Interrogating-CSCF (I-CSCF) or Serving-CSCF (S-CSCF).<sup>27</sup> [GC06] The session concept, arranged among the S-CSCFs as the central node in the signaling path, is important in the context of IMS-based implementations.

The P-CSCF serves as the access point of the IMS system by routing incoming service requests into the core network, acting in a proxy-like manner when accepting requests and serving or forwarding them to the core system. [MWK05] The P-CSCF can be located in the visited network. In the order of a service request, the node that usually receives the request after the proxy is the I-CSCF. The I-CSCF is the contact point, located in the providers network, that routes the request to the S-CSCF. At the S-CSCF the request is identified, authenticated and authorized. Depending on the service requester's contract, the appropriate response is initialized and forwarded to app server and media functions.

The following functional analysis provides a more in-depth picture of request signaling:

- P-CSCF provides:
  - User authentication
  - Verification of request correctness
  - Authorization of bearer resources for the appropriate QoS level
  - Redirection to I-CSCF
- I-CSCF provides:
  - Generation of Charging Data Record (CDR)

---

<sup>27</sup>S-CSCFs are sometimes called *SIP servers* because of their heavy use of the Session Initiation Protocol (SIP).

- Retrieval of user location information
- Encryption of request containing sensitive domain information
- Optional integration of Topology Hiding Internetwork Gateway (THIG)
- Redirection to S-CSCF
- S-SCSF provides:
  - Inspection of request and determining of further processing nodes
  - Central handling of session state information
  - Central user registration and publication to other nodes
  - End user billing and resource utilization

**Media Delivery Nodes** Another functionality, ancillary to signalling, is the delivery of media streams, which is carried out by the Media Resource Function (MRF) node. The MRF is functionally split into a signaling-related node, the Media Resource Function Controller (MRFC), and a media processing node, the Media Resource Function Processor (MRFP). The MRFC controls the deliverable media resources via a reference point from the S-SCSF. [GC06]

The following functionalities are specified in the MRF:

- Mix media streams (e.g. transcode different codecs)
- Obtain media-related statistics (e.g. for charging purposes)
- Determine media delivery contract constraints
- Process media delivery

**Application Server Nodes** App server (AS) nodes in IMS primarily host and execute domain services.<sup>28</sup> [GC06] This enables developers to develop and customize programmed logic. The AS is triggered by the signaling processes of the S-SCSF, which may include filter criteria. The AS also handles logic to determine which of the embedded application modules is appropriate for service processing. During processing, the AS can obtain control information (such as user, session, and media) at any time from the surrounding IMS nodes (such as signaling and media nodes).

IMS distinguishes between three types of AS, categorized according to their history of origin:

- The native Session Initialization Protocol (SIP-AS)
- The Open Service Access-Service Capability Server (OSA-SCS)
- The IP Multimedia Service Switching Function (IM-SSF)

These software-based engines represent runtime environments serving as Common Gateway Interface (CGI) or servlet containers. They support the integration of a Graphical User Interface (GUI) and accessibility via protocols such as HTTP or Wireless Application Protocol (WAP). In addition to GUI integration, AS could potentially serve as a forwarding engine, routing services into the IMS core. [GC06]

The default IMS-native app server is the SIP-AS, which acts as the central entity of service creation and processing. Commonly multi-layered, the SIP-AS follows the rules of modern application design, in which functionally separated components were designed to compute interactively. SIP-AS can be set up in three different ways to influence telecom service signaling:

- *User Agent (UA)*: UAs are regular endpoints that originate or terminate the SIP message flow (e.g hard or softphones).
- *Proxy*: AS acts as a redirection server. Based on the information included in the message, the AS reads and potentially changes SIP packet fields as they pass through.

---

<sup>28</sup>AS, in the context of IMS, refers to the app server software component described in Section 5.3.2 (Composition Essentials).

- *Back-to-Back User Agent (B2BUA)*: AS acts as an extended, more powerful proxy, processing service-specific logic depending on the incoming request. A SIP-AS proxy is not allowed, for example, to change certain header fields (e.g. `From`, `To`), change SDP or open nested SIP sub-requests.

In the SIP-AS environment, 3GPP further specifies a feature of particular interest, the Service Capability Interaction Manager (SCIM). This manager operates as a deployed application module and could potentially perform service composition.

## 5.4.2 Research Relevance

It is widely accepted that the IMS framework is a very advanced architectural approach to the implementation of NGN services. Accompanied by the enormous standardization accomplishments of 3GPP, the telecom community is forming an open and powerful framework to consolidate the role of the infrastructure provider. The key question is whether these specification efforts, challenged by the task of mobile and fixed-line integration, can be delivered in a well-engineered and agile manner. Moreover, if so the provider still must implement the proposed standard while at the same time operating legacy structures. Unfortunately, in our assessment the IMS does not represent a stable, practical solution. Rather, it can be seen as a resource from which to select potentially useful parts for implementation.

One definite advantage of IMS is its complete openness at the application level, where software solutions can adapt the reference point specifications. IMS's fundamental use of the IP protocol and its implementation of 3GPP-approved mechanisms (such as HTTP to SIP) underline this observation. According to our benchmark criteria, we can observe NGN-requirement compatibility in all of the IMS items we have examined. It is particularly important to emphasize the excellent design of IMS's telecom service-specific signaling and delivery processing stages, which are integrated into its separate, three-layered (application, service, control) service-processing environment.

IMS conforms to software service principles and enables the adoption of a service architecture of interacting telecom software services. However, we can identify the following critical points:

- NGN:
  - In general, although IMS covers all layers of the NGN model, it is lacking in maturity and practical application.
  - The specification still shows gaps.
  - The architecture is roughly put together and the nodes don't cooperate very well.
- Service Paradigm:
  - Software service paradigm-compliant development in IMS is enabled only, not integrated.
  - Reference point specifications define rigid connectivity on the underlying protocol level, implying the lack of a programming model for services as architectural building blocks.
- Service Composition:
  - Software service composition suffers from the missing programming model referred to above.
  - Non-functional properties and an appropriate control flow cannot be determined because of missing syntactically and semantically defined building blocks.
  - SCIM functionality appears to address composition essentials, but is only a superficial sketch and lacks completeness and depth.

## 5.5 Telecommunication Software Frameworks

---

Another kind of frameworking in telecom networks is represented by domain-dependent, software-dominated frameworks. Based primarily on mature software frameworks, Telecommunication Software Frameworks (TSF) attempt to combine business functionality and technology. Unlike the more overarching TNF solutions, the focus of TSF is restricted to the upper three, software-oriented layers of the NGN (application, service and control). However, TSF integrates software technology in a more advanced way. The advantage of

TSFs can be found in their very semantic definition of telecommunication functionality. TSFs are able to manage the transfer from semantic relationships to software implementations, from technology to coarse functionality. This facilitates the identification of telecommunication functionality and represents a step forward into mature software frameworks.

### 5.5.1 Open Mobile Alliance

The Open Mobile Alliance (OMA) is an organization formed in June 2002 to work toward the interoperability of mobile data services. With this goal, OMA assists in the creation of mobile services across countries, operators and mobile terminals. OMA cooperates in this work with organizations such as 3GPP and IETF. Central to the OMA concept is an open OMA Service Environment (OSE), arranging enablers that contain language-neutral interface specifications.<sup>29</sup> [All06] Instead of putting restrictions on enabler implementation, the OSE defines entities, visualized as architectural elements, that contain single functions in the manner of an assembly kit. Their implementation may be exposed as interfaces and packaged into dependent applications using service providers' resources.

Unlike IMS, OMA handles transitions using interfaces, as illustrated in Figure 5.4.

Approached from a technical angle, the OSE is based on an Execution Environment (EE). Like a container, the EE covers non-functional properties in an approach derived from the enhanced Telecom Operations Map (eTom), so as to offer a runtime environment for the enabler. This environment encompasses service lifecycle management, including a separate atomic step of service creation, but, unfortunately, does not include composition. Centrally covered non-functional properties are security and trust, quality (especially performance) management and error handling. To ensure non-functional and functional requirement fulfillment, OMA introduces a policy enforcer, separate from the enabler, which encapsulates utility functionality.

---

<sup>29</sup>An enabler is a technology intended for use in the development, deployment or operation of a service, defined in a specification or group of specifications and published as a package by the OMA.

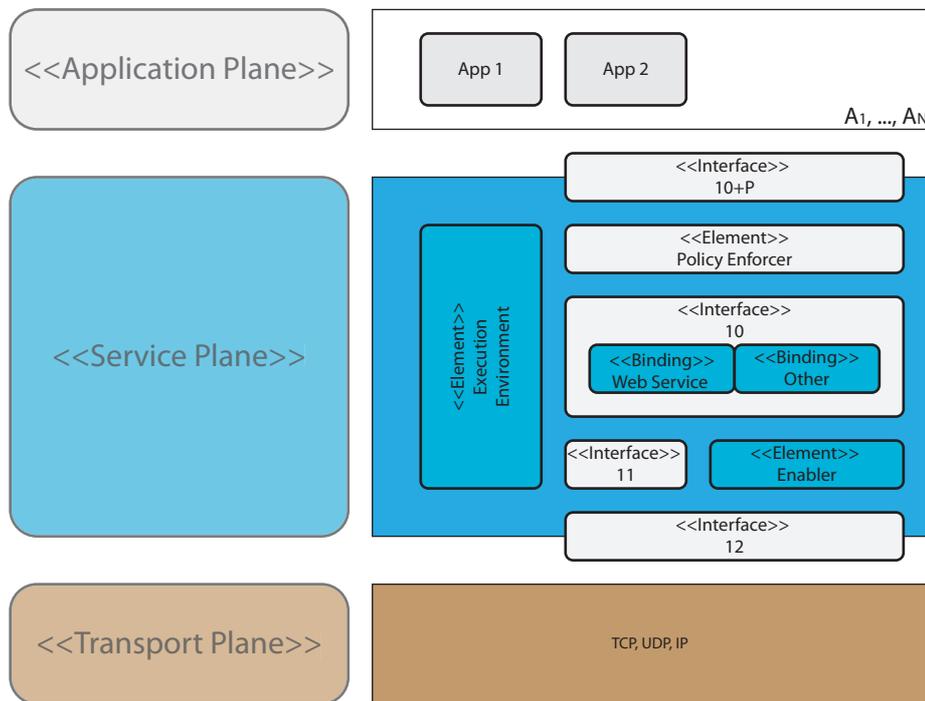


Figure 5.4: Open Mobile Alliance Service Environment

Beyond the EE, which is fundamental to OSE, OMA formulates four groups of standardized enabler interfaces:

- 10: Interfaces to enable OMA-called intrinsic functions. They are essential basic functionality.
- 10+P: Interfaces to satisfy policy guidelines. When exposing 10-interfaces they are extended through parameter (P).
- 11: Interfaces between the enabler and the EE.
- 12: Resource interfaces to transport plane or backend resources.<sup>30</sup>

<sup>30</sup>Transport plan interfaces and interfaces to legacy or backend systems are not in the focus of the OMA standardization efforts.

## 5.5.2 Open System Architecture (OSA)

The OSA activities are a joint effort of 3GPP, the European Telecommunications Standards Institute (ETSI) and the Parlay Group.<sup>31</sup> The Parlay Group was formed in March 1998 with the purpose of specifying a programming model to bridge the gap between IT and the telecommunication world.

From a technical point of view, OSA defines a framework containing a set of Service Capability Servers (SCS) which logically assemble Service Capabilities (SC).<sup>32</sup> SCs serve as bearers for Service Capability Features (SCF), each of which represents a single service. [Ins06a] The definition of these SCFs is the core work of the OSA consortium and consists largely of the definition of the logical structures (SCS and SC) and the specification of SCFs, using the interface strategy for transition facilitation.

Internally, the OSA architecture provides an abstract model using UML diagrams (Parlay UML) and high-level, model-derived, implementable interface APIs. For technical representation the OSA architects picked two types of API terminology and assigned interface description standards. [Ins06c]

- OSA/Parlay API can be realized by any of the following:
  - Corba IDL
  - Web services
  - Java Platform, Standard Edition (J2SE)
- Parlay X can be realized by:
  - Web services

The interface specifications further contain a strict model and description definition nomenclature to define the interfaces, and a predefined syntactic and semantic structure. We differentiate primarily between framework interfaces (discovery, security, manageability) and service interfaces (call control, mobility, messaging).

The basic specification and the transition from model to implementation are maturely specified, and, through the high-level API Parlay X description, directly compilable into runtime environments. OSA also specifies the

---

<sup>31</sup>ETSI was mentioned in Section 2.4 in the context of NGN.

<sup>32</sup>Service capabilities are defined by the parameters or mechanisms needed to implement services.

use of a Service Broker [Ins06b] which is expressed through an interface, `IpServiceBroker`, containing two services, `register()` and `unregister()`, as in the well-known observer design pattern. [GHJV00] Composition behavior functionality is planned for the future, but not as an explicit part of the specification.

### 5.5.3 Research Relevance

OMA's approach to providing high level interfaces seems to be successful. Based on their open perspective, they coarsely fit into the NGN facility criteria introduced in Section 2.7. In particular, they refine the concept of service clustering by introducing an architectural classification (10, 10+P, 11, 12) and a technical representation (web services). We can observe a general compliance with the software service paradigm, including its SOA implications: OMA defines architectural interfaces and requirements for decoupling interface and implementation, discoverability<sup>33</sup>, strict service clustering<sup>34</sup> and lifecycle support.<sup>35</sup> [All06] Also observable is a partial coverage of the identified non-functional properties of service composition.

However, a comparison with our benchmark criteria reveals the following gaps in the OMA approach:

- NGN:
  - Rudimentary support of service design
  - Lack of integration of signaling and delivery processing
- Service Paradigm:
  - Strong data and service focus without technical integration or implementable details
  - Semantic interface definitions restricted to the mobile, Wireless Application Protocol (WAP) background of OMA
- Service Composition:
  - No explicit composition handling, relying on enabler implementation

---

<sup>33</sup>Architectural Requirement 6.3.2#1,3,5

<sup>34</sup>Architectural Requirement 6.1.5#4

<sup>35</sup>Architectural Requirement 6.3.3#1

The OSA-specified framework succeeds in its syntactic and semantic definition of telecom services and their smooth integration into technical implementations. Supporting open interfaces for utility and domain services and constructed with a software service paradigm skeleton, OSA appears to be the most advanced and mature approach at the software service level. With explicit embedding in the IMS specification, OSA could be adopted as a complete NGN approach. Isolating OSA from IMS, however, in comparing OSA with our criteria we can observe some critical points:

- NGN:
  - Lacking consideration of signaling and delivery separation
- Service Paradigm and Composition:
  - Service paradigm compatibility and service composition capabilities restricted to web service and Corba mechanisms

## 5.6 Software Frameworks

---

A third category of framework approaches can be found in Software Frameworks (SF), which are technological concepts, transferable into the telecom domain and compliant with the service paradigm. Generally driven by the software engineering community, they implement SOA mechanisms and address service composition. The SF approach, in general, is characterized by resolving issues in order to satisfy functional and non-functional requirements.

### 5.6.1 Web Services

Web services are a recently emerging framework consisting of loosely coupled technical software services that can be addressed directly from the network and described using interfaces. The design of the web service framework follows the principle of one specification per main requirement, and as a consequence defines a multiplicity of XML-compliant descriptive languages. Unfortunately, the specification processes are sometimes uncoordinated; some

of them are redundant or even contradictory. Here we will emphasize those with the most composition relevance and maturity.

Web services can be seen as a triple-layer functional stack:

1. Native capabilities: basic service paradigm and SOA-compliant fundamentals (e.g transport, formatting, description, discovery)
2. Composition capabilities: composition-related functionality (e.g composition, coordination, orchestration)
3. Managing capabilities: non-functional characteristics (e.g. security, transaction handling)

Starting with the native capabilities, there are three main essential standards that can be identified: the Simple Object Access Protocol (SOAP) [W3C00] for transport and message styled formatting , the Web Service Definition Language (WSDL) [W3C01] for description and the Universal Description Discovery and Integration (UDDI) [OAS04] for creating directories and searching for services. SOAP can be set on top of HTTP, or alternatively, SMTP, FTP, Java Message Service (JMS) or IIOP could be used. SOAP was proposed to the World Wide Web Consortium (W3C) in May 2000 as the simplest common denominator for system interconnectivity.

Encoded using XML, SOAP specifies certain nested elements representing the main transport format. In more detail, web services use WSDL as a template to describe the way a SOAP message is syntactically and semantically constructed. WSDL is a central element that describes the service contract through complete decoupling of interface and implementation. Restricted for this reason to the basic service attributes, WSDL describes the service endpoint and the operations included in the services. These interfaces could be published in a UDDI directory, which represents broker functionality in the form of a technical registry.

The composition-related specifications are modeled above the native capabilities. This thesis focuses on composition approaches that feature a strict service paradigm compliance and develop services by reusing existing ones, mainly those with little composition depth. The software constructs involved are generally WSDL-described service interfaces. Using the narrow definition of the term *composition* proposed in Chapter 4.1, web services composition development began with the specification of Web Services Flow Language (WSFL) by IBM and XLANG by Microsoft. These eventually merged into

the WS-BPEL efforts driven by the Organization for the Advancement of Structured Information Standards (OASIS) consortium. There have been other interesting industry efforts, one by the BPMI, published as BPML, and another, the Web Services Choreography Description Language Version (WS-CDL)[W3C04], later submitted to the W3C.<sup>36</sup>

WSFL, XLANG, and BPML all implement composition using XML encoding. They take the data exchange into account when composing elementary service interfaces. Their language structures are fully elaborated and compared in the literature, from which we can deduce that BPEL seems to be the industry vendor-preferred approach. BPEL composes web services on the technical level (e.g. protocols, operations) by ordering member services based on WSDL-exposed interfaces.

Not directly assignable to a particular function, managing web services address the non-functional properties described in Chapter 4.7 but are not restricted to them. Prominent examples of managing capability specifications are WS-Transactions [BS02], WS-Security [OAS06] and WS-Policy [W3C06].<sup>37</sup>

## Research Relevance

Web services are a very generic concept, having, of the approaches we have examined so far, the most advanced service paradigm consideration, well adapted for SOA ideas and offering an embedded composition component. However, web services were developed as isolated solutions and not as a framework. Each specified interface must be customized, implemented and deployed as an individual domain architecture. We addressed some other issues in the preceding paragraphs. We can see web services as a simplified, practical approach to implementing service composition-related functionality in telecom software frameworks, but burdened with some critical limitations:

- NGN:
  - Non coverage on domain specific design and processing, e.g. the telecommunication specific functionalities of signaling and delivery
- Service Paradigm:

---

<sup>36</sup>The Business Process Management Initiative (BPMI) consortium published the Business Process Modeling Language (BPML).

<sup>37</sup>The frequently used prefix WS abbreviates web services and indicates a web service framework related language specification.

- Weak transition capabilities (e.g. a unified methodology) from semantics into technology
- Service discovery lacking in sufficient matchmaking capabilities
- Lacking domain-specific interface definitions
- Service Composition:
  - Support of non-functional properties involves the deployment of another specification (e.g. policy or transaction handling)
  - Optimized to message-styled asynchronous communication in a publish-subscribe message format, lacking integration of synchronous performance communication
  - Limited integration of quality parameters in the service description
  - Composition automation restricted to manual and semi-automatic invocation via declarative programming languages

Web services inspired many researchers to focus their efforts on domain-independent composition frameworks. Under the terms *coordination*, *orchestration* and *composition*, various approaches investigate the improvement of agility on the service level. Three areas are primarily targeted by the research:

- Goal definition and matchmaking
- Maintaining control and data flow and non-functional property optimization such as improvement of quality (including performance)
- Embedding of synchronous communication and automation of invocation during composition

Much of the current research addresses the improvement of goal definition and matchmaking capabilities as one of the steps during service creation and composition. An unambiguous expression of the service is essential to discoverability, machine readability and automation. To meet these challenges, the web services goal definition uses AI planning, agent theory-based techniques and contract-based design techniques to define initial states, necessary transitions and an explicit goal representation. [HKHA06]

Another research approach addresses the transformation of goals into control and data flow, implementing web service structures and the formalization of

these goals in terms of visualization and maintainability of compositions. This process is accompanied by efforts toward optimization of the composition, which would make it available for investigation by formalisms such as  $\pi$ -calculus. Furthermore, certain techniques in web services were dedicated to the textual or graphic description of compositions. Such techniques organize compositions by pattern [vdAADtH04] and visualize it using, for example, UML activity diagrams. [Aal03]

In the transition from contracts to web services, the main challenge lies in the identification of appropriate member service candidates, a process known as *matchmaking*. Semantics are largely used to resolve the descriptive issues.<sup>38</sup> To this end, several researchers extended UDDI to provide an appropriate infrastructure for matchmaking and discovery.

Current research also focuses on the improvement of non-functional composition-related properties. This includes, for example, the recognition of transaction composition patterns [BGP06], the significant increase in and prediction of performance and the extension of monitoring and quality control in general. The need to integrate quality awareness into a service composition is approached from various directions. Beginning with algorithms, researchers formulated several integration patterns, including the integration of a quality broker as a separate instance for quality assurance. [YL05] Of interest are approaches that embed SIP in a web service architecture to integrate synchronous communication. Liu and Chou [CLL06] encapsulate SIP messages in a WSDL interface document, thus enhancing the control of media delivery.

Another, more framework-oriented composition strategy is represented by frameworks that focus on runtime composition by defining an engine that can design and process services synchronously. This strategy is prototyped by the Star Web Services Composition Platform. [DS05] This system introduces entities such as an intelligent system for abstracting requirements into service descriptions, a broker registry that includes a discovery engine, a composition engine embedding wrapper instances which integrate technical heterogeneous platforms and the aforementioned execution engine. We see two innovative sub-concepts: the first is a rule set that provides a mechanism to translate user requirements into semantic service queries using the mediation format of an abstract service. The second innovation is the extension of WSDL to support the processing of real-time QoS metrics. [DS05]

Some researchers also elaborate on eFlow, a platform introduced by HP labs. [CJ01] eFlow enables the plugin-style publishing of services based on a service

---

<sup>38</sup>Semantics and their background will be investigated in Chapter 6.

discovery system. The composition engine selects services for binding based on predefined selection criteria. These criteria can be described in the context of a process schema containing multiple and generic nodes. The system can optimize service binding within the composition by monitoring the available plugged-in web services.

The Grid community has further adapted web services by transferring web service capabilities to Grid services, components that are based mainly on the Open Grid Services Architecture (OGSA) and use the BPEL language concept to handle flow control. [Cyb06] The reason for this conceptual transfer is the maturity of the web service composition approaches and BPEL's aptitude for reflecting concrete theoretical concepts (e.g behavioral patterns) and non-functional properties as implementable specifications.

## 5.6.2 Ontologies

We have seen in previous sections that semantic expression capabilities are notably lacking in most existing approaches. It was to fill that deficiency that the idea of the Semantic Web was born. The rapid development of the World Wide Web generated an urgent need to describe the data structure of the Web on a meta level and thus exploit the potential for automatic processing (such as that based on Agent technology) and improve recognition capabilities between highly distributed systems. As formulated by net visionary Tim Berners-Lee, the idea of the Semantic Web was to define and link Web data in such a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of the data across various applications. [HM02] One central methodology for modeling such behavior is introduced by ontologies.

### Definition of Ontology

*An ontology is a formal, qualified specification of shared conceptualization within a domain. The conceptualization typically contains classes (or sets), attributes (or properties), and relationships (or relations among class members).* [Gru08]

The term *shared conceptualization* implies a distributed, iterative and commonly acknowledged process of creation. This process results in a qualifiable specification, well formed and machine- as well as human-readable. The term

*formal* suggests the descriptive approach used by ontology languages.<sup>39</sup>

Domain ontologies, potentially distinguished by vertical and horizontal classifications, represent a very adequate alternative for modeling vertical dependencies. Their main contribution is to identify specific classes of objects and relations that exist in a focused domain. [FdGGN+02]

## Ontology Languages

Ontology languages generally organize sets of classes that mirror the core entities of a system and are described using systematized attributes. In this context a class is analogous to the class in object-oriented programming and is an abstract descriptive construct. Attributes are integrated to describe properties of entities or relationships between them. Basic, useful relationship expressions include classifications such as `subClassOf` or `superClassOf`, which can be used between attributes.

The ontology approach to software services began with the development of the Darpa Agent Markup Language (DAML). The goal of the DAML effort is to develop a language and tools to facilitate the concept of the Semantic Web. Encoded using XML and extending the Resource Description Framework (RDF), DAML provides a rich set of constructs with which to create ontologies and to label information so that it is machine-readable and -understandable. The Semantic Web arm of the DAML project focuses primarily on describing the capabilities of services in an unambiguous form. The project's proposed standard specification is published under the term Ontology Web Language - Services (OWL-S).<sup>40</sup> OWL-S claims to facilitate service development through service discovery, execution, interoperation, composition and execution monitoring. The specification includes the automatic processing of the lifecycle steps discovery, invocation and eventually composition, restricted to web services. [DPC11] A similar approach is presented by the Web Service Modeling Ontology (WSMO), which defines sets of services and mediation objects.<sup>41</sup>

Internally, OWL-S defines three entities: a service profile, a service model and a service grounding. While the `ServiceProfile` provides the information needed for a consumer to discover a service, the `ServiceModel` and

---

<sup>39</sup>A more general approach is represented by taxonomies, in which objects are classified into categories, hierarchically structured through explicitly defined parent-child relationships.

<sup>40</sup>DAML-S was previously developed under the umbrella of the W3C.

<sup>41</sup>An extensive evaluation of both concepts can be found in [LRPF04].

`ServiceGrounding` combined provide enough information to make use of a service once found. [DPC11] In a `subClassOf` relation, the `ServiceModel` derives a process model capable of handling flow and data control between interacting subprocesses. In considering the service composition capabilities of OWL-S, we observe that subprocesses (i.e. composition members) are divided into atomic, simple and composite processes and behave according to the behavioral pattern introduced in Section 4.6, using flow control elements.

### Research Relevance

Ontologies represent a central concept, applicable to services, of expressing the semantics of and between entities. The recognition of meaning, and the subsequent unambiguous transformation from coarse business functionality into fine-grained technical implementations, are essential in the design of services. This point is addressed by the ontology-related concepts of *partial understanding* and *transformability*. The term *partial understanding* refers to the idea that objects are represented in various ways and not all of them are transparent to potential consumers, whether human or machine. *Transformability* is based on the assumption that any information can be transformed into a homogeneous, machine-processable environment. Despite the potential inherent in these two concepts, some downsides could be observed looking at the current state of existing ontology implementations are deficient in the following ways:

- Missing of a unified methodology to transform defined ontologies transparently into fine granular technical service implementations and vice versa
- Missing an explicit integration of ontologies into software modeling (as in UML), tools and frameworks
- Lacking a definition for an ontology execution environment, including mature APIs for various programming languages to compute ontologies

Taxonomies and ontologies have been used frequently in research to capture service semantics. Their expression capabilities can be used to develop a semantically richer set of service attributes. The improved description of services directly enhances service composition, as it allows a more efficient control of composition behavior. This in turn implies an improvement in the definition of service contracts, the extension of matchmaking capabilities

based on these contracts and the facilitation of quality monitoring. Quality measurability is an essential factor in matchmaking and member detection. Through the advancements in expression capabilities, researchers were able to see the possibilities of runtime member detection and substitution, as well as the automation of composition. It is clear that improvements in semantic expression will be fundamental to the development of automatic composition. However, the research is still in a very early stage.

Some frameworks combine the potential of ontologies and web services to achieve composition. In this approach, while ontology languages deliver the expression of services, web services facilitate their execution. Two compelling questions arise from this context: how to define services sufficiently in order to compose them, and how to provide automated matchmaking.

The adequate description of services requires a comprehensive analysis and a careful transition from design to implementation. Some researchers have proposed dividing ontologies into meta, domain and mediation ontologies. Meta ontologies are intended to express data structures, domain ontologies are concerned with domain-specific relationships and mediation ontologies track data compatibility. In the context of this thesis, domain ontologies and particularly telecommunication ontologies are most relevant.

OSA and OMA<sup>42</sup> introduce fine-grained technical functionality, while the Information Technology Infrastructure Library (ITIL) and enhanced Telecom Operations Map (eTom) [Hua05] propose a business-related, more generic approach. Unfortunately, both lack the ability to make translations from business to technical and vice versa. One improvement could be the integration of ontology descriptions such as OWL-S service profiles. Other approaches attempt a solution via the assembly of services into processes using predefined internal processes and interactions. [PC03]

The accuracy of matchmaking is directly related to the quality of the service description. In other words, an inadequate description will always lead to a suboptimal set of composition member candidates. From a technical point of view, Sirin, Parsia and Hendler present a prototype of a composer for services. [SPH04] This software enables the composition of web services that are semantically specified using OWL-S. The composer takes a set of inputs, mainly WSDL descriptions, selects composition candidates and generates a BPEL-compliant composition description. Similar techniques are used by Agarwal et al [ADK<sup>+</sup>05], who introduce the concepts of logical and physical composition. Logical composition is the aggregation of functions into

---

<sup>42</sup>OSA and OMA were described in Sections 5.5.1 and 5.5.2.

new functionality, while physical composition allows the selection of service implementations based on non-functional properties.

### 5.6.3 Agents

Agent computing reflects the need of large heterogeneous organizations to do distributed planning and control of coordination. Agents, which act as building blocks like objects and services, are software implementations acting on behalf of a user-formalized goal. The users might be either humans or machines such as, for example, other agents.

Because of their distributed nature, agents provide the advantage of bypassing single points of failure. They facilitate the automation of machine processing in general, and could be used for service composition in a distributed system. Such a system might consist of multiple agents communicating from peer to peer. This communication could be specified using an Agent Communication Language (ACL).

An example of agent implementation is the Java-based Intelligent Agent Componentware (JIAC) framework, which is compliant with Foundation for Intelligent Agents (FIPA) specifications. JIAC uses OWL-S for semantic service descriptions and BPEL-based capabilities to provide semi-automatic service composition. The platform can be distributed across several machines and provides a development environment for customized behaviors. Composition efforts are in progress, based on Java's Service Description Language (SDL) [HKHA06], a BPEL extension intended to break down the hurdles of semantic expression and the lack of automation by functionally combining ontologies, declarative expressions, procedural expressions and composition.

#### Research Relevance

The origin of the agent concept can be found in artificial intelligence and is certainly transferable to service composition in the telecommunication domain. However, the agent approach is not mature in a practical sense, and the telecom domain aspect is lacking in semantic structuring, so there is still considerable research necessary before mapping can begin. This thesis, therefore, may serve to provide substantial input for future agent-based conceptualizations.

## 5.7 Comparative Analysis

A comparative analysis may be useful in assessing the relative strengths and weaknesses of existing solutions.

For this analysis, we have made a selection from the criteria associated with the three pillars on which this thesis is based: NGN, software services and service composition. Of the criteria we identified for NGN in Section 2.7, we have selected five for this table. Quality is omitted because it is already present as one of the criteria selected for service composition behavior. Access and regulation compliance are not included because they are not within our research scope. From the criteria for the service paradigm we have selected loose coupling, extensive lifecycle support and semantic predefinition of services. For service composition behavior, we omitted the criteria of security and error handling, because they were explicitly excluded in the definition of our research scope.

The analysis employs a threefold scale, using (-) for a missing functionality, (+) for elementary support of the functionality and (++) to represent a mature implementation.

Approach	Network Facility (NGN)					Service Paradigm			Service Composition Behavior					
	IP Foundation	Transport Abstraction	Services as SIB	Design / Processing Separation	Signaling / Delivery Processing	Loose Coupling	Lifecycle Support	Semantic Predefinition	Composition Pattern Support	State Management	Transaction Handling	Quality Measurement	Invocation Automation	Communication
<b>Telecom Network Frameworks</b>														
TINA	++	++	++	++	+	+	++	++	-	++	++	+	-	++
IMS	++	++	++	++	++	++	+	+	-	+	+	-	-	+
<b>Telecom Software Frameworks</b>														
OMA	++	+	++	+	+	++	++	++	-	++	++	+	-	++
OSA	++	+	++	++	+	++	++	++	+	++	++	+	-	++
<b>Software Frameworks</b>														
Web Services	+	-	+	+	-	++	++	-	++	+	++	+	+	+
Ontologies	+	-	+	-	-	+	+	+	++	-	-	+	+	+

Table 5.2: Comparative Analysis of Related Work

---

## 5.8 Summary

---

In looking back over the approaches presented in this chapter we can identify commonalities, complements and contradictions. Based on the research scope introduced in Section 5.2, we identified potential points of composition applicability for a software-based framework approach. The investigation introduced four categories of approaches that offer capabilities that may benefit service composition:

- *Essentials*: Mathematical notations reduce the complexity of composition behavior and enable simulation and thus proof of correctness. Additional software constructs such as EBSs or app servers represent appropriate composition runtime environments, while descriptive programming languages enable the persistent machine- and human-readable storage of a composition. APIs encapsulate functionality in programming models.
- *Telecommunication Network Frameworks (TNF)* offer a wide range of functionality and affirm the importance of the IN concept. The principles of NGN are comprehensively addressed. Concepts such as the abstraction of transport through an interface layer, and the structuring of the service plane using technical nodes and interaction strategies, are very useful to framework design. Of paramount importance is the defacto standard of IMS, which offers a feasible solution for the signaling (SIP) and media delivery (RTP) requirements on a protocol level. Based on the protocols specified in IMS, SIP-AS, conceptualized in compliance with the software service paradigm, represents a point of applicability for service composition.
- *Telecommunication Software Frameworks (TSF)* complement the TNF functionalities with software technology. The advantage of TSF lies in ready facilitation of entity and interaction definition on a very technical level. TNS defines objects, operations and interfaces according to service paradigm principles. Of particular interest is the semantic depth of definition in the OSA framework, where services become accessible through a high level API (Parlay X), introducing a practical way of exposing interfaces.
- *Software Frameworks (SF)* arrange domain-independent components while demonstrating mature standards and patterns of implementa-

tion. Functionally relevant concepts are the WSDL standard for service interface description, UDDI for service discovery, the BPEL standard for service composition and ontology descriptions such as OWL-S for semantic expressions.

Upon consideration of this survey of related research, it becomes apparent that a workable solution for the composition of telecommunication services can be built on existing work. However, we have identified four points where improvement is necessary before such a solution can be achieved:

- The conceptual semantic predefinition of telecommunication services and the appropriate placement of composition in this conceptualization (for example, the integration of composition as a common utility service)
- The general transformation of those semantics into and between various perspectives of computation, such as the business, data, and technology perspectives
- The conceptual computation of the service lifecycle, in particular match-making, lookup and composition
  - Detection, choice and verification of appropriate candidates for composition members
  - Creation of an optimized composition description according to given goal parameters
  - Verification of composition correctness (i.e., that members fit together), including the verification of semantic compatibility between members
  - Potential automation of member detection and composition description creation
- The implementation of service composition in the context of the telecommunication domain, with its heterogeneous semantic clusters and their highly diverse non-functional properties

Many of these points are related to the definition of semantics in the domain. In the next chapter we work toward a semantic understanding of NGN Services. We will propose an ontology as a semantic specification of the core functionality of the domain, addressing challenges such as the design of non-functional properties, the design and refinement of granularity, reuse and the automation of invocation.

## Part III

# Service Frameworking in the NGN



# Chapter 6

## Services in Next Generation Networks

*“Let us together rebuild this world that we may share. ”*

*Aragorn, Lord of the Rings: The Return of the King, [2003]*

### 6.1 Synopsis

---

Previous chapters presented an overview of the cornerstones of this thesis in the form of a theoretical outline of the research triangle of NGN, software services and service composition, from their inception to current research activities. The boundaries of the research scope have been stated and persistent issues have been identified.

A meaningful understanding of NGN Services will be essential to our framework development. The existing solutions introduced in Chapter 5 range from a completely undefined approach (as demonstrated by, for example, IMS) to concrete technical definitions (as shown in OSA). The current challenge, when defining semantics in a framework, lies in achieving semantic definitions of coarse business services that remain technologically agile and flexible in the framework construction. Many researchers bypass semantic

definitions entirely by focusing on technology. The complexity of this issue is the result of an extreme volatility of services on the business level. While utility services remain relatively stable over the time, domain services change frequently according to customer demand.

The problem of the volatility of business services can be approached through the identification of a minimally necessary set of service definitions that can be used to establish a sustainable model of telecommunication semantics. The central question faced in this chapter is: "Which domain and utility services must be provided, and which are optional, to form a minimal semantic commitment in the telecommunication domain, while remaining technologically flexible so as to allow the addition and removal of services at any time?" This question also implies a search for an adequate granularity of services, keeping in mind potential reuse, quality assurance and automatic invocation. The investigation and further clarification of these issues are the motivation for this chapter.

The chapter introduces NGN Services as a semantically interacting stack of functionality, and aggregates the services in clusters that bundle homogeneous functionality. Semantic expressions offer a high potential for describing services and thus enrich the composition activity with a concrete service definition. Such definition facilitates service matchmaking, including member detection, verification and selection. The purpose of this chapter is to produce a telecommunication ontology: a semantic conceptualization of the entire telecom domain, which will define services and their dependencies, from a consumer point of view, by defining descriptive service clusters using semantic clauses. The clusters will be formalized using the NGN specifics introduced in Section 2.4, the syntax developed in Chapter 4, and the seven non-functional properties of services introduced in Section 4.7. An implementation of the resulting ontology will be demonstrated in the prototype in Chapter 8.

## 6.2 Terminology

---

Before proceeding with our discussion of NGN Services, it may be helpful to clarify some of the terms involved.

**Working Definition, NGN Service** *An NGN Service is a set of telecommunication capabilities that are syntactically and semantically ascertainable, self-contained, observable from the business, informational, and technical perspectives, and consisting of implementations of the application, service, and control functional layers.*<sup>1</sup>

Two other terms of interest originate from various standardization organizations, extending Kessler's observations: [Kes05]

**Working Definition, Service Capability** *A service capability is the technological or business driver behind the implementation of services. It can be represented by a service cluster, a group of services or a standalone service.*<sup>2</sup>

**Working Definition, Service Cluster** *A service cluster is a group of services sharing the same describing criteria.*

Clustering can be used to reduce domain complexities and analyze dependencies between services. In this chapter we will specify meanings and dependencies inside and between clusters, with the aim of capturing useful semantics that will facilitate the computation and composition of software services.

---

<sup>1</sup>These layers are described as part of the NGN fundamental characteristics in Chapter 2.4.

<sup>2</sup>The term *service enabler* may be used synonymously.

## 6.3 A Telecommunication Service Ontology

---

The construction of a service ontology can be approached by way of the definition of descriptive clauses and the relationships between them. The purpose of such an ontology will be to establish a foundation of semantic understanding.

When discussing semantics, two helpful concepts can be employed. First, clustering facilitates analysis by delivering a satisfactory level of service functionality abstraction through the collection of similar services into groups. The second useful concept is that of semantic clauses, which allow the definition of semantic entities within a cluster.

Our proposal, based on the layers of the NGN reference model developed in Section 2.4.4, is to construct a tuple of four interacting service clusters in an NGN environment: *community*, *content*, *communication* and *control* services. These clusters aggregate similar service functionality and interact among themselves.

### 6.3.1 Semantic Perspectives

Three semantic perspectives aid in refining the semantics of the clusters:

- *Meta semantics* help to identify and describe a service cluster.
- *Static semantics* describe a service cluster in its non-composed atomic state.
- *Dynamic semantics* interpret service cluster capabilities, which are important in the behavior of a cluster (for example, during composition).

The ontology developed here presents a template for descriptive semantic clauses, useful in expressing the meanings of the service clusters and deducing their composability.<sup>3</sup>

---

<sup>3</sup>This ontology shows one way of arranging semantic clauses. Other arrangements are possible.

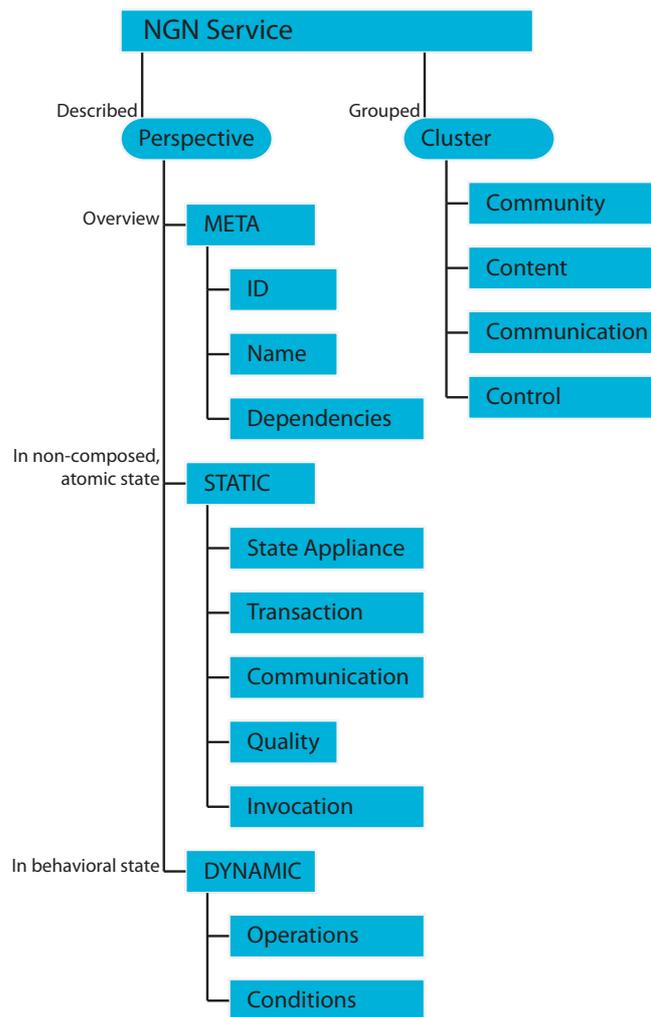


Figure 6.1: A Telecommunication Service Ontology

### Meta Semantics

The structure of the ontology minimizes descriptive entities so as to keep the analysis clear and understandable. The meta level is reduced to three clauses, entities we have labeled **ID**, **Name**<sup>4</sup> and **Dependencies**. The entities **ID** and **Name** represent unique, arbitrary identifiers. The **Dependencies** clause examines the cohesion between the clusters, coarsely assessed by the qualities of **weak** and **strong**.

<sup>4</sup>In defining semantic dependencies, the Name clause uses a naming convention in which clusters are given the prefix CL, while services use the prefix OP (for operations).

## Static Semantics

Non-functional properties<sup>5</sup> can be applied as meaningful clauses to describe service in a static state, providing useful information about the service before it is processed. This static information may include **State**, **Transaction**, **Quality**, **Communication** and **Invocation**.

**State and Transaction Clauses** The state clause uses the attributes of **Stateless** and **Stateful**. Transactions can be expressed by employing the clauses **Non-Transact** and **Transact**.

**Quality** Service quality can be statistically expressed with the help of metrics. Metrics, in this context, are predictable quantifications describing pre-defined conditions, meaning those that are known before the service is processed. An example of quality metrics is the key-value[unit] definition of thresholds, such as the description of service processing as **ResponseTime** < 1, unit [second]. Composition can be seen as a special case of service processing, and therefore it must satisfy static quality thresholds in the same way an atomic service does.

The process of adequately describing service quality can be supported by a general quality model that defines meaningful metrics. [ZBD<sup>+</sup>03] Among the most useful quality clauses in such a model are **Reactivity**, **Reliability**, **Availability** and **Scalability**. These clauses express the quality of a service  $S$  acting as part of an available set of services  $N_S$  and invoked in certain instances  $I_S$  with the potential for the occurrence of a set of failures  $F_S$ .

**Reactivity**  $q_{ret}$  measures the period of time elapsed between initialization and termination of a service. This clause can be arithmetically calculated as the sum of the network throughput  $T_{thr}$  and the complete service execution time  $T_{exe}$ , which is a sum of the initialization  $T_{int}$ , processing  $T_{pro}$  and termination  $T_{ter}$  time. **Availability**  $q_{avl}$  represents the probability of successful accessing of the service. **Reliability**  $q_{rel}$  measures the probability of successful service processing. Finally, **Scalability** measures the ability to process an increasing number of service instances. In other words, **Scalability**  $q_{scl}$  accumulates single service reliability data over a large number of processed instances and identifies the variations in successful process-

<sup>5</sup>Non-functional properties were introduced in Section 4.7.

ing depending on a changing number of service instances. Table 6.1 provides a mathematical illustration of quality metrics.

	<b>Reactivity</b>	<b>Reliability</b>	<b>Availability</b>	<b>Scalability</b>
Description	Elapsing time between service initialization and termination	Probability of successful processing	Probability of accessibility	Probability of successful processing averaged over a defined number of services
Arithmetics	$q_{ret} = T_{thr} + T_{exe}$ where $T_{exe} = T_{int} + T_{pro} + T_{ter}$	$q_{avl} = 1 - (F_{avl}/N_S)$	$q_{rel} = 1 - (F_{rel}/N_S)$	$q_{scl} = 1 - (F_{rel}/I_S)$

Table 6.1: Quality Model of Software Services

**Communication and Invocation** The communication model of a service can be organized by means of a differentiation between **Synchronous** and **Asynchronous** communication. The last static clause, invocation, assesses the service's potential for automated invocation. This assessment uses a scale consisting of **Manual**, **Semi-Automatic** and **Automatic** identifiers.

### Dynamic Semantics

The dynamic semantic level is helpful in describing the behavior of services in, for example, a composition context. This description can be carried out using the clauses **Operations** and **Conditions**. Operations indicate the functionality included in the service on a very coarse granular level, according to the service principles of autonomy and loose coupling.

Conditions define the relationships between services during processing. They describe the circumstances of service processing and thus the restrictions on processing or composing. Conditions can be analyzed using two different types of qualifiers:

- Arithmetic ( $\leq, <, \geq, >, =$ )

- Temporal (before, after, concurrent)

In the next section, semantic clauses are mapped to clusters to express their meanings.<sup>6</sup>

## 6.4 Service Cluster Semantics

---

A service cluster is an abstract pool of NGN functionality. The specification in Figure 6.2 is a template for semantically describing service clusters, based on the structure of the introduced ontology and its three levels of semantics. Chapter 4.9 introduced a formal description for the syntax of services. Figure 6.2 extends this service-independent syntax to a notation that specifically expresses the semantics of a cluster.

The notation in Figure 6.2 is based on the eBNF<sup>7</sup> syntax language and uses a declarative key-value format. The advantage of the eBNF format lies in the meta perspective, which allows a simple definition of relationships without predefining any implementational details. Using eBNF for the service cluster description, we can organize the clauses of the model and assign appropriate terminal symbols.

This specification establishes a meaningful semantic understanding for the domain and will be integrated into our framework and prototype.

We have proposed a tuple of four major domain service functionality clusters: *community*, *content*, *communication*, and *control*. In the following sections, we will map this structure to each of these clusters by assigning appropriate terminal symbols and specifying default values to develop a computable, semantic structure for the domain.

---

<sup>6</sup>The semantics introduced here are somewhat arbitrarily defined to serve as a template for framework development.

<sup>7</sup>The used eBNF symbols are outlined in Appendix A.1.

```

1 SERVICE-CLUSTER
2 (
3   (*identification*)
4   (META
5     {ID}; (*domain wide unique ID*)
6     {Name}; (*domain wide unique name, beginning with the prefix CL_*)
7     [Dependencies('Weak'|'Strong')]; (*cohesion between clusters*)
8   )
9   (*semantics in atomic state*)
10  (STATIC
11    [State('Stateless'|'Stateful')];
12    [Transaction('Non-Transact'|'Transact')];
13    [Quality]; (*reactivity, availability, reliability and
14    scalability*)
15    [Communication('Synchronous'|'Asynchronous')];
16    [Invocation('Manual'|'Semi-Automatic'|'Automatic')];
17  )
18  (*semantics in behavioral state, e.g. during composition processing*)
19  (DYNAMIC
20    [Operations]; (*operations use the prefix OP_*)
21    [Conditions];
22  )
23 )

```

Figure 6.2: Semantic Specification of a Service Cluster

### 6.4.1 Community Services

**Meta Semantics** Community services package functionality that is human consumer-centric and provides shared, largely non-technological, common information. The organization of communities is an iterative process, driven by explicit human interest and accompanied by the highly flexible development of services. Much ongoing research focuses on the mechanisms for establishing and maintaining online communities. Ranging from demographic and analytical approaches that target similarities between humans based on attributes such as age, sex, wealth, and education, to more technical research involving sensor networks or even ambient intelligence, the process of establishing and administering communities enjoys significant research attention.<sup>8</sup>

To support identification, our convention is to give communities the name

<sup>8</sup>It is interesting to note a related emerging research field: Human-Centered Computing (HCC), which aims to bridge the existing gaps between the various implementations of computing systems that support human activities. [JS05] The field was established to design computations and computational artifacts in support of human endeavors. [CDA+06]

CL\_CTY and the ID 100.<sup>9</sup>

Communities frequently change their functionality and thus their domain service portfolio. They attract users with personalization opportunities and connect them using content and communication capabilities. To keep users' interest, it is important to create frequently varying and inter-communicative content. Communities act as containers of functionality, accumulating routines belonging to other service clusters such as content, communication and control. [DK07]

Without elaborating user's demands in more detail, we can observe a strong dependency between communities and the content, communication and control functionalities. For example, online communities involve rich interactions between users based on content (e.g. graphic images and videos), communication (e.g. SMS messaging), and control (e.g. central session handling). These relationships between the clusters must be computed in a technically reliable manner.

**Static Semantics** Because they behave like containers in requesting services from other clusters, communities must include strong integration and, particularly, mediation capabilities. User-centricity requires state management and long-range transactional support.

To predict the user-perceived reactivity of a community service, the primary criterion is response time. General thresholds have been determined for response times acceptable to users: [CRM91]

- ResponseTime  $\leq$  1 second is the limit within which the user does not feel the delay.
- ResponseTime  $\leq$  10 seconds is the limit within which the user feels the delay, but still feels comfortable.
- ResponseTime  $>$  10 seconds is the interval beyond which users are not attracted to the service.

Response time can be calculated as the product of service execution and network throughput. Reliability, availability and scalability also must be guaranteed. We can assume that the generic end user expects reliability and

---

<sup>9</sup>The ID serves as a unique identifier to trace the relationships between coarse business services and their technical, more finely granulated representations.

availability with a probability of 99.9 percent, which implies a maximum failure rate of 0.01. Scalability, interpreted as the average reliability over a certain number of service instances, should also not exceed the 0.01 percent failure rate.

Communities require a highly secure and trusted access, especially when processing sensitive user data such as financial information. They rely on a scalable and high-performance service integration layer to process functionality from third-party providers. This layer includes the management of quality parameters for providing and retrieving services in intra-, inter- and extra-scoped integrations.<sup>10</sup> When integrating services with different distribution scopes, the information exchange can be performed via synchronous and asynchronous communication scenarios, depending on the technical requirements of the integrated services.

**Dynamic Semantics** In order to succeed, communities need to meet the needs of their individual members and maintain themselves over time. Communities consist of humans and the content they exchange, [ABJ<sup>+</sup>06] and these human users require capabilities for communicating with each other. Keeping these facts in mind, we can identify three main community operations: user handling, the integration and assignment of content and communication between users. User handling includes all functionality connected with personal data such as authentication, authorization and access to information. Content integration addresses the requesting and assembly of appropriate content according to the users' demands. Communication integration collects all functionality related to collaboration between users.

Predictable quality can be predefined using the quality model introduced in Section . The four quality clauses, **Reactivity**, **Reliability**, **Availability**, and **Scalability**, use metrics which must be implemented in a composition through the goal service  $S_{gol}$ . The quality of a goal service is an aggregation function of member service quality:<sup>11</sup> [ZBD<sup>+</sup>03]

$$\sum_{S_{mem}=0}^N q_C; q_C = q_{ret} + q_{avl} + q_{rel} + q_{scl}$$

Composition can be initialized in communities through goals defined by a (human) service consumer. The system translates a user goal into a viable

<sup>10</sup>Distribution scopes were discussed in Section 3.5.2.

<sup>11</sup>Within a composition  $C$ , a goal service  $S_{gol}$  is processed by the invocation of its member services  $S_{mem} = S_1, S_2, \dots$

software service processing. This translation can be automated, especially the sub-activities of lookup, matchmaking and binding of high quality member candidates.

Based on the assumption of a quality aggregation between members and goal service, the various clauses can be interpreted in different ways. The **Reactivity** of a goal service can be seen as the sum of the reactivity of all member services. The **Availability** and **Reliability** of composed services must be determined using a factor  $Y$ , which measures the importance of a member service in a composition. For example, if  $Y = 1$  the member is a critical service in the composition, while if  $Y$  becomes 0 the service has no effect on the composition. From this we can formulate the following coherences in dynamic behavior:

$$q(C)_{ret} = \sum_{S_{mem}=0}^N q_{ret}; q(C)_{avl} = \sum_{S_{mem}=0}^N q_{avl} * Y; q(C)_{rel} \equiv q(C)_{avl}$$

Scalability depends on the total number of invoked instances of a service in a composition. Assuming that a service does not depend on the type of consumer that invokes it, scalability investigates the behavior of a service only when its instances are heavily increased. Consequently, scalability is an average of the reliability of all composed services  $q(C)_{rel}$  in the system.

Ultimately, from a human end user's perspective, it does not matter whether a service is composed or atomic. For this reason, the predictive thresholds for atomic and composed services are identical in static and dynamic semantics.

**Formalization** Communities represent a significant point of applicability to service composition, with their requirements for quality assurance and potential support for automation. This makes them a particularly suitable starting point for the composition prototyping in Chapter 8. Communities serve as a human-centric service hub, semantically expressed in Figure 6.3:

```

1  Community
2  (
3  (META
4    ID = 100;
5    Name = CL_CTY;
6    (*dependencies to other clusters CL_CTT(content),
7    CL_CCN(communication) and CL_CTR(control)*)
8    Dependencies = CL_CTT:Strong,CL_CCN:Strong,CL_CTR:Strong;
9  )
10 (STATIC
11   State = Stateful;
12   Transaction = Transact;
13   Quality =
14     Reactivity(ResponseTime) < 10[seconds],
15     Availability > 99.99[percent],
16     Reliability > 99.99[percent],
17     Scalability > 99.99[percent];
18   Communication = Synchronous, Asynchronous;
19   Invocation = Manual, Semi-Automatic, Automatic;
20 )
21 (DYNAMIC
22   (*operations UH (user handling), CI (content integration),
23   UC (user communication)*)
24   Operations = OP_UH, OP_CI, OP_UC;
25   Conditions =
26     OP_UH before, concurrent OP_CI, OP_UC;
27 )

```

Figure 6.3: Semantic Specification of the Community Service Cluster

## 6.4.2 Content Services

**Meta Semantics** While communities are human centric, the content cluster represents data-centric services. Content is a digitalization of information as media, and content services deal with the maintenance of those digitalizations. The content creator has full knowledge of the content and complete control over all aspects of its delivery, such as how the content will be accessed, delivered, presented, cached, and replicated. [CLC06] The service cluster forms the informational backbone of the domain, administering the information submitted by users.

For identification purposes we give content services the name `CL_CTT` and assign an ID of 200.

Content is a heavily requested resource in contexts such as communities.

This cohesion is not commutative: content services do not necessarily rely on community services. The implication of this is that the management of content does not require a community or human centricity. Further, content services do not necessarily cooperate with communication services in a situation where, for example, communication services describe user-perceived communication scenarios such as electronic mail and telephony. Content does cooperate mandatorily with the control cluster, which directs initiation and delivery of content objects. Like communities, content services cooperate with control services to centralize user and data management.

Media, the main entity of content, is the network-distributable representation of a telecommunication information object. Media objects are stored and transmitted using media representations and formats appropriate to each object. Generally, we can distinguish three major media format categories:

1. *Text*: visual content, transmitted as a variety of digitalized media components (e.g. words, pictures)
2. *Audio*: aural content, transmitted as a media stream
3. *Video*: mixed visual and aural content, transmitted as media streams

The initiation and delivery of content are closely connected with the definition of the underlying communication model. Content is mainly delivered synchronously, which means that the invocation and response of a service are dependent and closely connected in time.

**Static Semantics** Since content services act as the main repository of data in the telecom domain, the services must be highly accessible to consumers. This access requires a scalable but not necessarily stateful information exchange. However, in the case of sensitive information the content must be transaction-capable. The delivery and therefore the transmission of media depends on the communication properties requested by the consumer. The primary distinction is that between synchronous and asynchronous delivery of content.

In terms of quality prediction, content uses the same criteria described in Section 6.4.1 for communities. Content is delivered mainly as data packages, redundant and inherently stateless. Some content, such as audio and video, is delivered in a stream as a stateful, continuous sequence of encoded signals.

Unlike other media objects, streaming media does require delivery as quasi-synchronous communication.<sup>12</sup> [SKY03]

**Dynamic Semantics** The content service cluster is essentially a collection of four major operations: **Access**, **Authoring**, **Storage** and **Publishing**, closely linked to existing models of the content lifecycle. [Ler03] The **Access** operation describes the ability to provide content in an easily reachable and reliable manner. **Authoring** combines the creation, update and deletion of content media. **Storage** enables content persistence, and **Publishing** addresses the behavior of making content available to a potentially restricted number of consumers. **Receiving** describes the technically oriented process of transmitting the content to the consumer.

Content services are most heavily requested during the composition of community services. **Access** enables content sharing, while **Authoring** and **Publishing** serve as maintenance services to maintain currency. **Storage** is a utility required for content services because of their heavily persistent memory allocation.

**Formalization** Figure 6.4 formalizes the specification of the clauses for the content cluster.

### 6.4.3 Communication Services

**Meta Semantics** Communication services represent interaction-centric functionality and consist of single or multiple interactions. They collect the basic information-exchange scenarios among human and machine service consumers. The cluster is assigned the unique name `CL_CCN` and the ID 300.

Communication services control the connectivity between human users to guarantee a reliable transmission of information. For example, if two human users request an asynchronous one-to-one communication, communication services find a suitable candidate in email. More technically, they wrap underlying transport capabilities that can be integrated into other clusters. Communication services can be seen as a high level, user-perceived functional representation of the non-functional property *communication*.<sup>13</sup>

---

<sup>12</sup>Because of this commonality with communication services, quality predictions for streaming media will be elaborated in the communication services section.

<sup>13</sup>Communication as a non-functional property was introduced in Section 4.7.7.

```

1 Content
2 (
3   (META
4     ID = 200;
5     Name = CL_CTT;
6     (*dependencies to other clusters CL_CTY(community),
7     CL_CCN(communication) and CL_CTR(control)*)
8     Dependencies
9       CL_CTY = Weak,CL_CCN = Weak,CL_CTR = Strong;
10  )
11  (STATIC
12    State = Stateless;
13    Transaction = Transact;
14    Quality = Quality(Community);
15    Communication = Synchronous;
16    Invocation = Semi-Automatic, Automatic;
17  )
18  (DYNAMIC
19    (*operations AC(access), AT(authoring), ST(storage),
20    PB(publishing)*)
21    Operations = OP_AC, OP_AT, OP_ST, OP_PB;
22  )
23 )

```

Figure 6.4: Semantic Specification of the Content Service Cluster

Communication services are strongly cohesive with control services, so that they can request the administration and delivery of conversational data. This cohesion implies the explicit handling of signaling and delivery, especially when users or machines interact in real time.

**Static Semantics** Like content services, communication services integrate media and special media formats to transmit information. Transmissions can consist of interactions that take place with a certain required degree of quality. The most prominent distinction is that between real-time or non-real-time interactions. Both modes are associated with a set of unique functional requirements drafted in RFC-1821: [Li94], [DF90]

- *Strict Predictability*: the ability of a system to guarantee a service consumer (machine or human) predefined behavior
- *User Control*: transparency to the consumer of the system's behavior

- *Timeliness*: simultaneous media delivery, the main differentiation being that between real-time and non-real-time services
- *Quality*: stringent quality metrics measurement during runtime

User Control requires a stateful connection, but not necessarily a transactive one. Depending on the type of transported media (e.g. text, audio or video) and the timeliness (real time or non-real time), we can formulate predictable preconditions.

In particular, the real-time transmission done by audio or video streams contains some known general restrictions. [Wal06] Streams can be characterized by a complex interaction structure, different service modes (e.g. pull, push or both) and an increased bandwidth. [AZ03] The restrictions on continuous streams can be defined using the telecommunication network criteria *delay*, *jitter* and *loss*, elaborated in Section 2.5:

- Delay (one-way)  $\leq 0.15$ [seconds]
- Jitter  $\leq 0.03$ [seconds]
- Loss  $\leq 0.1$ [percent]

**Dynamic Semantics** Dynamically, communication services behave entirely differently from other services so that they can support direct user interaction. A classification introduced by Nahrstedt and Balke suggests distinguishing email, instant messaging, telephony, audio and video conferencing as separate services. [NB04] Keeping in mind our intent to compose communication service with other clustered services, this classification seems overly specific. We propose, instead, a focus on the generic functionality of **Signaling** and **Delivery**, where **Signaling** represents an optional real-time communication-related operation. Additionally, communication provides operations for collaboration between user, enabling multi-user interaction.

**Formalization** Figure 6.5 shows the characteristics of the communication service cluster.

```

1 Communication
2 (
3   (META
4     ID = 300;
5     Name = CL_CCN;
6     (*dependencies to other clusters CL_CTY(community),
7     CL_CTT(content) and CL_CTR(control)*)
8     Dependencies
9       CL_CTY = Weak,CL_CTT = Weak,CL_CTR = Strong;
10  )
11  (STATIC
12    State = Stateful;
13    Transaction = Transact;
14    Quality =
15      Reactivity(Delay) < 0.15[seconds],
16      Reactivity(Jitter) < 0.03[seconds],
17      Availability(Loss) < 0.1[percent];
18    Communication = Synchronous, Asynchronous;
19    Invocation = Semi-Automatic, Automatic;
20  )
21  (DYNAMIC
22    (*operations SG (signaling), DV (delivery),
23    CL (collaboration)*)
24    Operations = OP_SG, OP_DV, OP_CL;
25    Conditions =
26      OP_SG before OP_DV;
27  )
28 )

```

Figure 6.5: Semantic Specification of the Communication Service Cluster

#### 6.4.4 Control Services

**Meta Semantics** Control services act as a central authority, taking responsibility for controlling user and media data across all other service clusters. This responsibility includes the providing of, for example, user and related session data, as well as a central control of media interaction based on the concept of intelligent networks (IN).<sup>14</sup> For unique identification purposes, the

<sup>14</sup>The intent behind IN is to decouple the user state from service processing, so as to add and remove services seamlessly without losing state information or user-related

cluster is assigned the name `CL_CTR` and the ID 400.

We have already noted strong dependencies between control services and each of the other clusters, especially communities (as regards user control) and communication (real time media control).

**Static Semantics** Control services are accessed only by other service clusters, and not directly by the domain service consumer. Nevertheless, they form a cluster that collects from all other clusters a repository of user-sensitive information that must be maintained as stateful and transactive. Control services are usually requested synchronously, but it is also possible to access them in an asynchronous mode. Real-time communication presents a special case in that it is transmitted synchronously and must be centrally signaled and delivered within the control cluster.

In terms of quality, control services must satisfy the general thresholds of reactivity, availability, reliability, and scalability, as introduced in Section 6.4.1.

**Dynamic Semantics** The tasks of the control cluster consist of the functionality of `UserControl` and `MediaControl`, and to monitor the access state of a service consumer, `SessionControl`. A session in this context contains all access-specific consumer-related information, such as login and password combinations and permissions. The session can also be used to exchange information between services. Control services administer the delivery of media, heavily required by the content and communication service clusters. The information required by these clusters include, at a minimum, media format and delivery conditions such as predictable quality criteria.

**Formalization** Control services can be formalized as outlined in Figure 6.6:

---

media information. This decoupling facilitates mechanisms such as session-based charging, without regard to which services are invoked.

```

1 Control
2 (
3   (META
4     ID = 400;
5     Name = CL_CTR;
6     (*dependencies to other clusters CL_CTY(community), CL_CTT(content) and
7     CL_CCN(communication)*)
8     Dependencies
9       CL_CTY = Strong,CL_CTT = Strong,CL_CCN = Strong;
10  )
11  (STATIC
12    State = Stateful;
13    Transaction = Transact;
14    Quality = Quality(Community)
15    Communication = Synchronous;
16    Invocation = Automatic;
17  )
18  (DYNAMIC
19    (*operations UC (user control), SC (session control),
20    MC (media control)*)
21    Operations = OP_UC, OP_SC, OP_MC;
22    Conditions =
23      OP_UC concurrent OP_SC,
24      OP_UC before OP_MC;
25  )
26 )

```

Figure 6.6: Semantic Specification of the Control Service Cluster

## 6.5 Summary

In this chapter we have specified a foundation for telecommunication domain functionality with a minimal semantic commitment. The functionality is arranged in a tuple of four clusters, *community*, *content*, *communication* and *control*. The chapter presented an ontology to define and detail semantic meanings and assign functional properties (real timeliness, predictive quality) and non-functional properties (state, transaction, communication and invocation). This assignment consists of semantic clauses, investigated from the meta, static, and semantic perspectives.

The following image summarizes the ontology, illustrating the clusters, their semantic clauses and the cohesion between them:

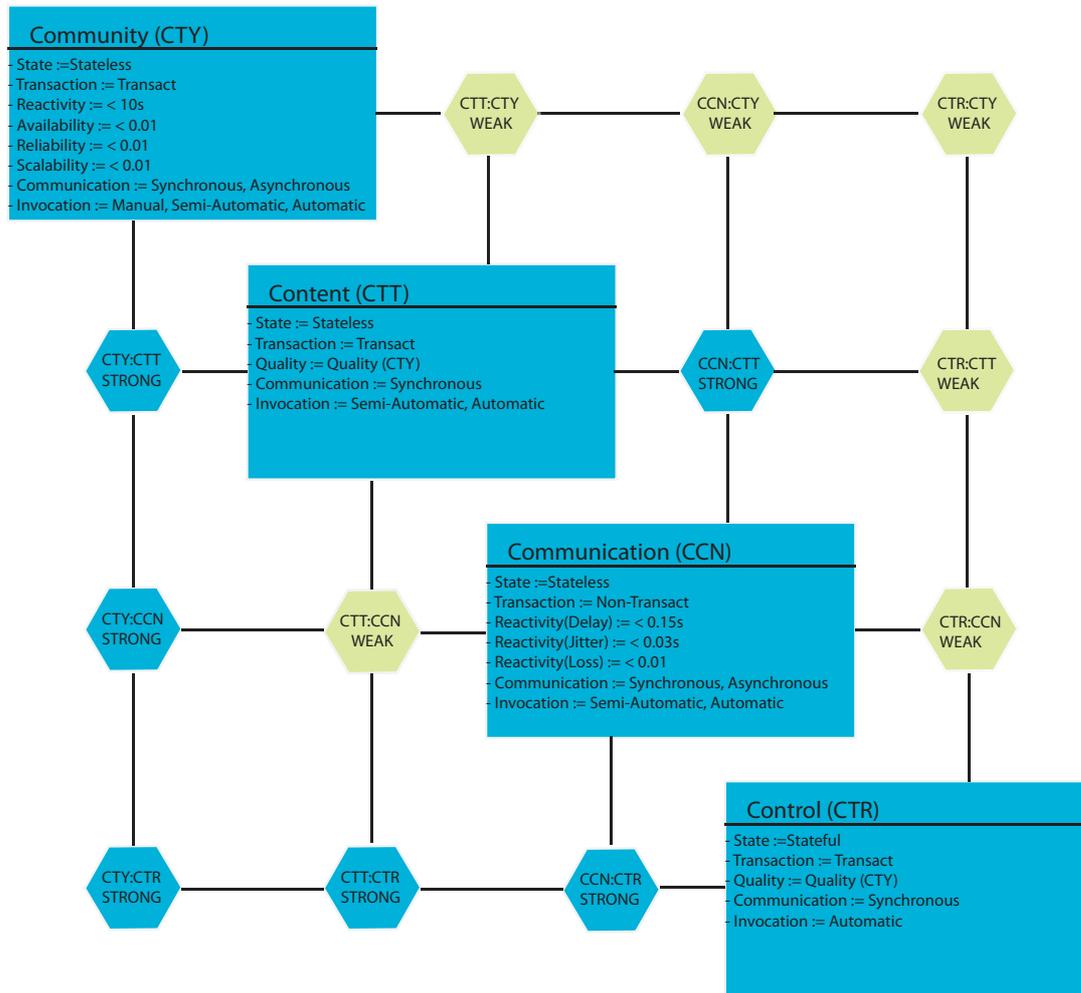


Figure 6.7: NGN Service Cluster: Cohesion and Clauses

Communities were identified in this chapter as a mediating hub to represent services in a user-centric manner. Content and communication can be integrated substantially into communities. This integration is accompanied by certain dynamic and static characteristics.

We introduced a general quality model, consisting of reactivity, availability, reliability and scalability, which defines cluster- and integration-specific quantified metrics. Particularly significant is the integration of real-time communication, which requires predictable quality and the two-stage processing of signaling and delivery.

The central service-independent integration of the control cluster was identified as a primary internal challenge because of its strong cohesion with all

of the other clusters. The various integration scenarios between the clusters were investigated in the light of the non-functional and functional specifics of software services introduced in Chapter 4.

In the next chapter, the services identified and expressed in semantic notations here will be computed within a software framework. This framework will be developed by translating the identified semantic relationships into business, data and application architectural building blocks, whereby the technological architecture will demonstrate the composition of services. In this way, the semantic analysis presented here will serve as a foundation for assessing the degree of composeability of and between services.

# Chapter 7

## SOTF: A Service-Oriented Telecommunication Framework

*“So, is mine within the framework of my rules.”*

*David, Knight Moves [1992]*

### 7.1 Synopsis

---

The translation of domain specifics, or semantics, into computable structures was identified in Chapter 5 as one of the main challenges in telecommunication engineering. Having established in Chapter 6 a semantic understanding of the telecommunication domain, we are now prepared to enter into the computation of the semantic artifacts known as clusters. Their computation (in other words, their composition) should take place in an IP-based environment, ideally based on a software service lifecycle (introduced in Section 3.5.1) and implementing the non-functional properties introduced in Section 4.7.

Frameworking is widely accepted as a design methodology for logically structuring complex information. [Ins06d] The usefulness of its concepts is not restricted to software engineering, but significantly facilitates the analysis of software systems on several abstract levels. The components that result

from the process of frameworking can be instantiated and custom tailored to serve as a template for developing concrete software solutions. The objective of frameworking is the structuring of design and development processes [GBBB05] in such a way as to document and reuse their eventual results in the context of other solutions. The methodology integrates several textual and graphical presentation formats.

The goal of this chapter is the conceptualization of an architecture that will bring together diverse perspectives to provide a representation of the software system that is the ultimate aim of this thesis. The architecture will be constructed using frameworking methodology in general and, in particular, its conceptual and constructional derivative, the TOGAF architecture framework. TOGAF serves as a practical guideline for bridging concept and implementation. The use of familiar methodologies (such as ADM) and standards (such as UML) will help to achieve system consistency and general guidance.

The framework developed in this chapter, a Service-Oriented Telecommunication Framework (SOTF), shows various aspects of software services interacting in an IP-based telecommunication system. The SOTF design process integrates IP network facility characteristics, the software service paradigm principles and implications and the syntactic and semantic functional and non-functional properties of telecommunication services. The comprehensive preparatory work of previous chapters, especially the theoretical background and analysis, will be integrated to create a sustainable software system of interacting Architectural Building Blocks (ABB) that is compliant with the conditions and limitations of the research scope.<sup>1</sup>

This chapter explores service composition as functionality embedded in a framework and cooperating with surrounding subsystems and components. The technical implementation of the related subsystems will be developed in Chapter 8, which will demonstrate a framework-embedded composition machine and its entities interacting in a real-world scenario.

---

<sup>1</sup>These boundaries were laid out in Section 5.2.

## 7.2 The TOGAF Approach

---

The Open Group Architectural Framework (TOGAF) is a framework that includes a detailed method and a set of supporting tools for developing software architecture components.<sup>2</sup> In the literature we can find a variety of attempts to unify frameworking efforts for the definition of software architectures. The TOGAF approach was chosen largely because of its explicit definition of a development methodology, its integration of other approved methodologies and tools and its advanced practical capabilities for transforming theoretical constructs into implementational artifacts.

Another reason for using TOGAF was its advanced, integrated treatment of requirements management. As we have observed in previous chapters, a large number of requirements must be carefully classified and considered frequently during development. TOGAF supports these processes explicitly by defining a direct relationship between the requirements and each view of the central methodology. In addition, TOGAF facilitates the employment of software service capabilities and therefore integrates some of the principles and implications of the software service paradigm.<sup>3</sup>

The core around which the TOGAF specification is organized is the Architecture Development Method (ADM), which defines a practical, mature, step-by-step way of approaching a target architecture, from business analysis to technology implementation. ADM proposes textual and graphic formats for presenting the results, which are classified into three main architectural viewpoints. [Gro06]

- *Business Architecture* includes business strategy, governance, organization, and key business processes.
- Information Systems Architecture defines the logical and physical data assets of the framework based on the business architecture, and is developed in two scopes:
  - *Data Architecture* determines the relevant data entities.

---

<sup>2</sup>The TOGAF specification is published by The Open Group on its public web site (<http://www.opengroup.org/architecture>), and is currently available in version 8.1.1. [Gro06]

<sup>3</sup>However, consideration of the service paradigm in TOGAF is in an early, immature stage.

- *Software Application Architecture* sets the data assets into the context of software-implemented processing components.
- *Technology Architecture*, based on the business and information architectures, maps the components to concrete software building blocks.

In TOGAF these viewpoints are arranged from the top downward as a breakdown from business to technology. The viewpoints are structured internally into objectives, approaches, inputs, steps and outputs. Each viewpoint also includes a set of principles and tools to facilitate a stringently structured development. Beginning with an introduction and an architectural vision and continuing through the assignment of the architectural views, TOGAF systematizes development as an iterative process. The process consists of nine phases arranged in an architecture development cycle, illustrated in Figure 7.1.

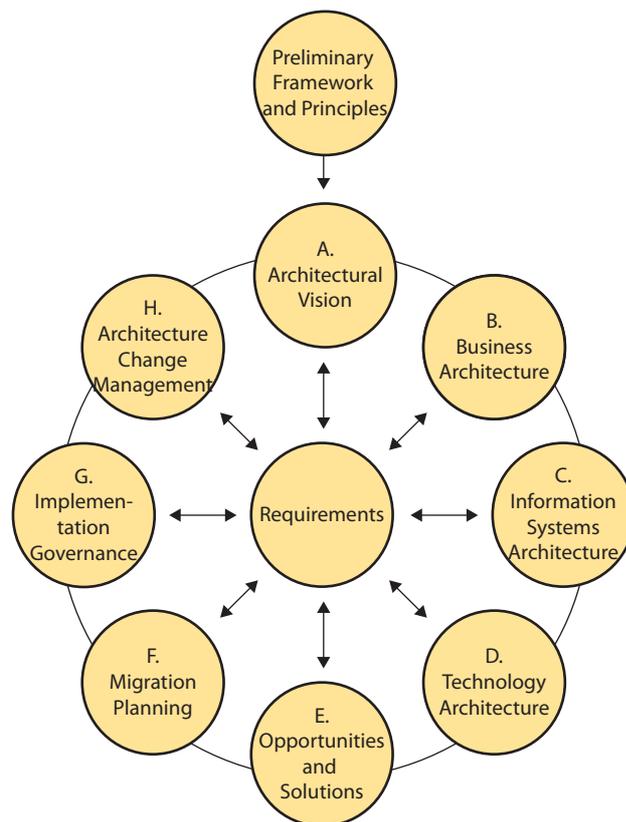


Figure 7.1: TOGAF 8.1.1 Architecture Development Cycle (ADM)

As a generic methodology, ADM is intended to be tailored to specific needs. As this thesis is focused on the pure development of the framework, here we

will make use of ADM's phases A-D and will not consider the governing or migrating phases (F-H).

## 7.3 A Service-Oriented Telecommunication Framework (SOTF)

---

The purpose of this chapter, again, is to develop a framework based on the theoretical discussions in previous chapters. Using TOGAF, we will focus on the phases<sup>4</sup> that correspond to the business, information and technology architectural views. For the result of this development we propose the name *Service-Oriented Telecommunication Framework* (SOTF).

The following criteria will be used to describe each view in detail:

- **Input and Requirements:** We will use the input information derived from previous chapters and the view-related requirements.
- **Approach:** Methodology and objectives. The Approach declares the architectural building blocks (ABB) used in formulating the view and specifies the methods for identifying and illustrating them, as well as their behavior.
- **Result:** The intended outcome is presented in textual and graphic format.

During development, each criterion will be applied within each phase. In this way, in conformance with TOGAF, each phase will provide its own architectural building blocks.

---

<sup>4</sup>*Phase* is used synonymously with *view*. Our previously analyzed views correspond with Phases A-D in TOGAF.

### 7.3.1 Framework and Principles

The first phase of our development, framework and principles, outlines the main coherences of the framework development. These basics include general facts about the perspective (such as scope, assumptions and stakeholders), the starting point of the analysis and the general architectural principles.

The scope of the framework is generally defined in Section 5.2. One of our primary assumptions restricts the functional implementation of the project to the software-dominated layers of the NGN reference model: the application, service and control layers. The framework focuses strictly on the service layer, achieving agility by constructing software components that can process semantically defined services, facilitated by manual and automatic service composition that takes into consideration properties such as quality and error handling.<sup>5</sup>

#### Use of NGN, SOA, Other Resources

The framework analysis begins with the alignment of the characteristics of NGN with the principles of SOA. The NGN facility requirements<sup>6</sup>, service principles and implications<sup>7</sup>, composition properties<sup>8</sup> and service semantics<sup>9</sup> will be considered when discussing overall requirements.

Throughout the design process we will integrate a variety of design fundamentals. The main sources of these are the service paradigm, the SOA implications and the TOGAF continuum. Certain existing design pattern catalogues have been useful for consulting common design practices.<sup>10</sup>

---

<sup>5</sup>The main stakeholders in the general architecture, specifically in a composition setting, were introduced in Section 5.2.2.

<sup>6</sup>NGN facility requirements were listed in Section 2.7.

<sup>7</sup>Service paradigm principles, with SOA implications, were elaborated in Chapter 3.

<sup>8</sup>Composition characteristics, mainly non-functional, were described in Section 4.7.

<sup>9</sup>The semantics of telecom services were explored in Chapter 6.

<sup>10</sup>A pattern describes a "problem to be solved, a solution, and the context in which that solution works." [Joh97] The main pattern catalogues used in this research are listed in Section 7.7.

## Use of UML

The outcome illustrations have benefited greatly from the achievements of the Unified Modeling Language (UML), a standardized specification language developed to facilitate object modeling.<sup>11</sup> Uniquely in the software engineering community, UML offers a common design dialect in software development. It defines a variety of diagram types (e.g. structure, behavior and interaction) and relationships (e.g. aggregation, association, composition, depends on, generalization) to enhance software frameworking.

Because of its differentiation between static and behavioral analysis and its coherence between objects and services, UML is a good fit to serve as a facility for design and documentation. Unfortunately, UML does not support the service paradigm explicitly, so for our purposes it is necessary to adapt UML's general object modeling techniques to support service-oriented development.

## 7.4 Architectural Vision

---

Our architectural vision is built upon the research questions posed in Chapter 1, the issues, derived from these questions, that require resolution, the business and technical requirements exposed by our research, and telecommunication's actors (human and machine), their roles and their responsibilities.

### 7.4.1 General Vision

The mission of this architectural endeavor has been defined as the establishment of an agile telecommunication service layer. The need for agility on an infrastructure level is based on the requirement for flexible provisioning of architectural building blocks. This flexibility includes the capabilities for adding and removing services, for processing of events in the environment and for changing properties (even at runtime), as well as the efficient interaction between building blocks that is achieved by assuring full coverage of non-functional properties. The business environment is represented by the telecommunication domain, organized by characteristic roles (e.g. manufacturers, operators) and attracting customers with diverse, frequently changing telecommunication service portfolios. [DK07]

---

<sup>11</sup>UML is currently available in version 2.1.1 [Gro03]. An overview can be found in [KS05].

Telecom applications, representing the top layer of technical abstractions, invoke software services to perform a specific, coarse-granular task. However, applications may also act as service consumers, binding services from the service and control layer. To reduce complexity and avoid ambiguity in this cycle of consumption, here we will focus on applications as the main service consumer. This focus leads to a complete decoupling of applications and services on the technical level, in conformance with the service paradigm. We will establish another loosely coupled relationship between transport and the service and control functionality. This will enable the transparent binding of transport capabilities, independent from the underlying access and aggregation technology.

In summary, the technical environment consists of three decoupled functional layers exposing application, service and transport capabilities. In this ecosystem, applications as service consumers request services, and services wrap transport capabilities. This vision is depicted in Figure 7.2.

The service layer aggregates domain and utility services. It includes software services conceptualized as reusable building blocks which operate centrally in the core of the network by connecting applications with transport capabilities. Below the domain functionality, the control layer acts as the central point of data (e.g. user and session data) and media interaction (e.g. signaling and delivery). Because of the tight relationship between service and control functionality, the two layers should be designed homogeneously.

## 7.4.2 Composition Vision

Service composition can be employed on various levels of telecommunication architectures, but must be applied selectively and used only where appropriate. As we have observed in previous chapters, it is possible to ascertain optimal points of service composition applicability. The software-implemented service layer has been previously identified as such a point of applicability. Services, acting in lifecycles, are fine-grained software components invoked by telecom applications, which can be seen as intermediaries between human-requested requirements (goals) and technically provided resources (services).

The dynamic, automated, and accurate translation from human-centric coarse-grained business functionality to available technical resources can be a superior solution for achieving agility through composition. Having reviewed and compared the most current implementations, however, we must consider the realization of this solution a long-term target. For this reason, here

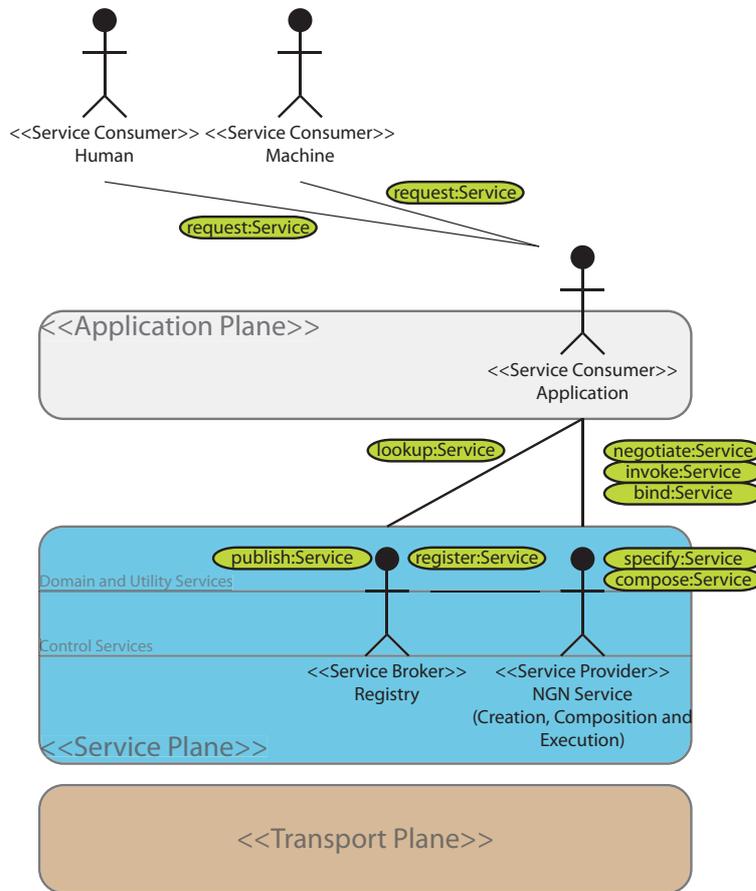


Figure 7.2: SOTF: Architectural Vision

we will concentrate on working toward this goal by building a sustainable framework foundation for future enhancements. This long-term approach addresses some of the most prominent composition issues: fundamental semantic definitions and their translation into technology, the quality-assured matchmaking of member candidates and the generation of a goal-optimized service composition description. The basis and also the main challenge of composing services is the definition of descriptive service contract entities to be used in service composition and to be processed in a system environment. The contract must describe the services sufficiently to allow well formed, expressive queries that conform to the requested goal parameters. Once these structures have been defined, the composition must be embedded in a framework, which allows the reproduction of the service lifecycle.

### 7.4.3 General Principles and Requirements

In previous chapters we collected the principles and requirements needed to facilitate the use of a service-oriented software approach in NGN. These criteria offer a range of implications that must be considered in a framework design. TOGAF arranges such criteria centrally, so that they can influence the architecture at any stage of the framework development, with effects on one or multiple viewpoints. Every item on the following list influences the design in a specific way:

- Network facility characteristics, which are the analytical basics of NGN networks
- Software service paradigm principles and SOA design implications
- Service composition properties and behavioral patterns
- Service semantics

While the basic software service principles include design consequences in all views, network facility characteristics and service-specific requirements deploy their effects mainly in the information and technology architecture viewpoints. Figure 7.3 lists all of the criteria and assigns their implications to the TOGAF viewpoints.

## 7.5 Business Architecture

---

Defining the business architecture is a prerequisite for architectural work in the information systems and technology viewpoints. It enlarges upon the business environment by analyzing organizational, functional, informational and geographical aspects of the predefined domain. We propose using coarse business services as the main analytical building blocks of this architectural view.<sup>12</sup>

---

<sup>12</sup>These business services were previously defined in Chapter 3.2.2 as a technology-independent, packaged set of capabilities perceived by the human end user.

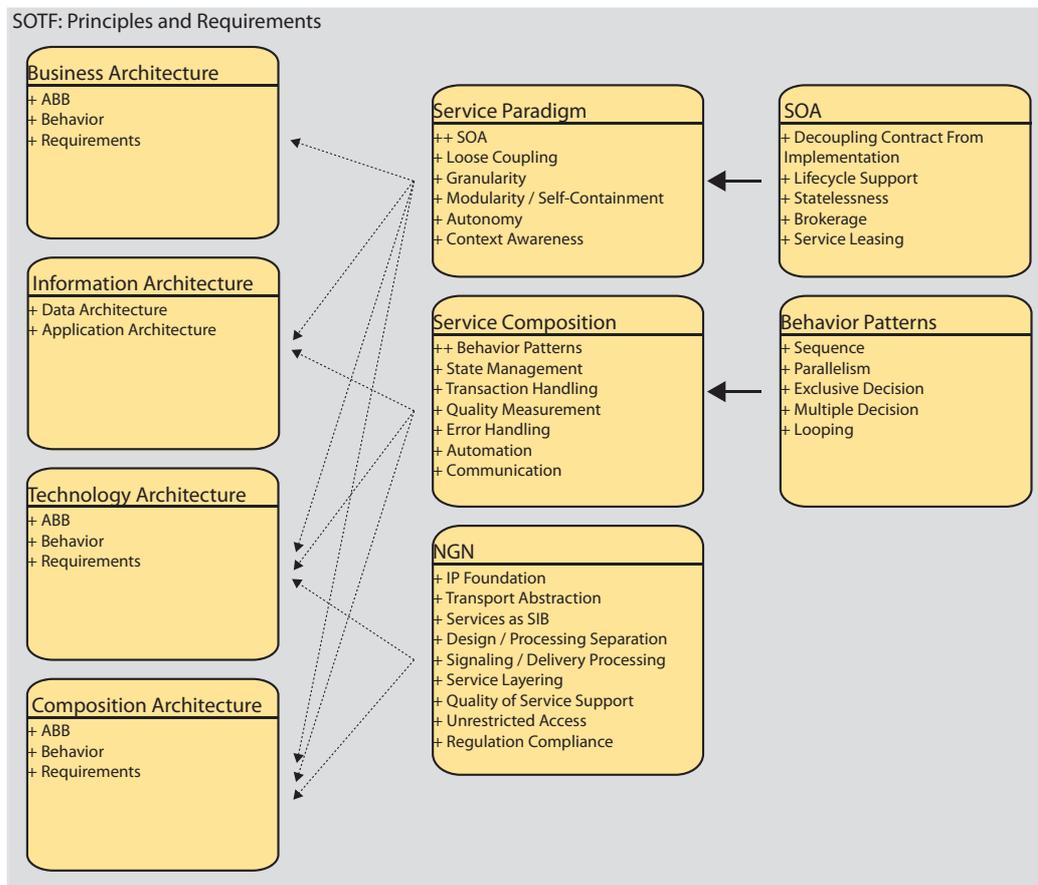


Figure 7.3: SOTF: Principles and Requirements

### 7.5.1 Input and Requirements

Based on the service paradigm and its principles of loose coupling, coarse granularity and modularity, several business services can be identified and organized. Operating as packaged functionality, the basic semantics of these services were developed and named in Chapter 6. The semantics of these services were organized into four clusters in the telecom domain and mapped into the NGN ecosystem. Their coarse granular structure, consisting of the clusters and their corresponding operations, can be refined by assigning several business methods<sup>13</sup> which further refine the granularity by attaching implementational methods to the services.

<sup>13</sup>Business methods are fine-grained conceptual descriptions of software system behavior. Their identification can be used to refine the behavior of service operations.

To maintain non-ambiguous child-parent relationships between clusters, operations and business methods, the identifier scheme we have selected for the clusters will be extended to label the relationships between the concepts. In this way the business architecture is represented as a related network of business clusters, operations, and methods of decreasing granularity. For example, if 100 identifies the community cluster, the relationship between that cluster and the specific user handling and session mediation can be denoted as `100(Community):110(UserHandling):112(mediateSession)`. The resulting labels can be used to assess the effects of potential changes of the clusters on the operations. In other words, they can help to answer the question of which operations are affected if a given telecom cluster is changed, or even vice versa.

In this view, it is important to define an appropriate association between operations and business methods. The methods must be modular, but also autonomous and inherently stateless. The architecture must also identify specific domain functionality and general utilities. According to the service paradigm principles, the identification of valid operations of the service cluster must follow some general guidelines to reduce undesirable side effects and improve the flexibility of applications: [FM04]

- *Orthogonality*: Each business method should define a distinct, non-overlapping functionality in the context of the application domain under consideration.
- *Minimality*: Business methods should be designed to cover reusable, generic functionality, customizable by any implementing entity.
- *Extendibility*: A service provider should be able to add or remove functionality.
- *Evolution support*: Business methods should support versioning and upward and downward compatibility between versions.
- *Clarity and elegance*: Naming schemes should be well defined and easy to understand.
- *Uniformity*: Naming schemes should be consistent.

## 7.5.2 Approach

The proposed analytical breakdown from service clusters to operations to business methods requires a high degree of modularity and independence. The identification of the clusters is built upon the semantics of the four cluster elements: community, content, communication and control.<sup>14</sup> This set of domain-specific clusters will be completed with the addition of a *system cluster* and a *composition cluster*. The system cluster groups the surrounding, non-service-specific properties that should be provided by a software system. The composition cluster packages the functionality needed for composing and decomposing services, which is non-service specific and can be used across clusters. Since composition should not be provided in a service-specific manner, the composition cluster can be regarded as independent functionality and swapped from one cluster to another.

Extending the system used in the previous chapter to identify clusters, operations within the clusters are described using a unique name and an identifier. Using the UML stereotyping feature, to distinguish between domain and utility operations we label them with the markups <<Domain>> and <<Utility>>.

The clusters and operations are represented as functional units in a UML object diagram. Our identification efforts are confined to a representative set of functionalities, complying with the principle of minimal semantic commitment<sup>15</sup> by identifying generic core functionality rather than delivering exhaustive lists of potential business processes.

The recognition of fine-grained business methods is an iterative process and can be accomplished in several cycles. The model introduced here represents only one possible way of representing the telecommunication domain. It can be enriched, changed and refined when requirements or environmental parameters change. The main contribution of this analysis is to capture a complete picture of the telecom domain's business landscape, in which services of various granularities can be computed and composed when interacting in a framework.

---

<sup>14</sup>These clusters were introduced in Chapter 6.

<sup>15</sup>The principle of minimal semantic commitment was introduced in Chapter 6.

### 7.5.3 Result

The following specification illustrates the business service landscape as a semantically dependent network. Our six new service clusters have been systematized using the approach, introduced earlier in this chapter, of refinement from cluster to operation to business method.

#### Community Cluster

Based on the results of the semantic analysis, the community cluster (ID and name: 100:CL\_CTY) serves as a mediating hub incorporating operations from other clusters under different parameterizations. Considering communities as a pure mediator, their functionality can be restricted to the invocation of other clusters. This restriction arises directly from the nature of communities, which represent the connection from the framework to the human end user. For this reason, the community cluster is the one where the greatest necessity for agility can be observed. Agility can be achieved here by assigning loosely coupled, rather than rigidly assigned, mediating methods.

Based on the semantics introduced in Section 6.4.1, the community cluster can be represented by the following business operations and business methods:

Business Operations	Business Methods	Functionality
User Handling (OP_UH)	111 mediateUser	Integration of personalized user data into the community
User Handling (OP_UH)	112 mediateSession	Maintainance of user access-specific data
Content Integration (OP_CI)	121 mediateContent	Integration and assembling of content in the community
User Communication (OP_UC)	131 mediateCommunication	Provision of communication capabilities between single and multiple peers

Table 7.1: Community Cluster Business Operations

## Content Cluster

Characterized by a strong cohesion with communities, the content cluster (200:CL\_CTT) stores the current data of the telecommunication service landscape. Based on its semantics (developed in Section 6.4.2), business operations and business methods can be assigned as shown in Table 7.2.

Business Operations	Business Methods	Functionality
Access (OP_AC)	211 contentInfo	Addition, removal and search of all information pertaining to content objects
Access (OP_AC)	212 search	Lookup of content data according to predefined search criteria
Access (OP_AC)	213 filter	Filtering a content object or a group of objects by certain criteria
Access (OP_AC)	214 deliver	Transport of data from provider to consumer
Authoring (OP_AT)	221 edit	Changing of content objects
Authoring (OP_AT)	222 classify	Assignment of types and categories to certain content objects
Storage (OP_ST)	231 store	Persistent storage of content
Storage (OP_ST)	232 backup	Emergency storage
Storage (OP_ST)	233 recover	Restoration of content from emergency storage
Publishing (OP_PB)	241 publish	Opening content for consumption by service consumers
Publishing (OP_PB)	242 subscribe	Time-based consumer request for content

Table 7.2: Content Cluster Business Operations

## Communication Cluster

In addition to content, communities heavily request communication (300:CL\_CCN) capabilities, which were coarsely developed (in Section 6.4.3) to enable interactions between communicating users. Distinguishing between real-time and non-real-time communication, this cluster provides connection logic in synchronous or asynchronous mode between single or multiple users.<sup>16</sup> Real-time communication is processed in two steps, signaling and delivery. Both stages include various legs, each of which represents an association between a communication and an address. [PSM03] The concept of a leg as a directed communication step (inbound or outbound) can be generalized to describe, and eventually compute, each of several communication steps. The legs must be signaled and delivered depending on the required communication mode.<sup>17</sup> Beyond legs, the communication can be functionally enriched by assigning certain media and media formats,<sup>18</sup> including, for example, the enrichment of an ongoing audio real-time communication with a video signal.

Table 7.3 outlines the main communication business operations/business methods.

---

<sup>16</sup>A communication between multiple users is often referred to as a *conference*.

<sup>17</sup>Communication modes were elaborated in Chapter 4.7.7.

<sup>18</sup>These were introduced in Section 6.4.2.

<b>Business Operations</b>	<b>Business Methods</b>	<b>Functionality</b>
Signaling (OP_SG)	311 <code>signalLeg</code>	Initiation and control of real-time communication
Delivery (OP_DV)	321 <code>deliverLeg</code>	Transport of all available communication modes
Collaboration (OP_CL)	331 <code>setup</code>	Initialization of a conference
Collaboration (OP_CL)	332 <code>addParticipant</code>	Adding user to conference
Collaboration (OP_CL)	333 <code>removeParticipant</code>	Removal of user from conference
Collaboration (OP_CL)	334 <code>addMedia</code>	Addition of another media format (text, audio, video)
Collaboration (OP_CL)	335 <code>removeMedia</code>	Removal of a media format from the conference
Collaboration (OP_CL)	336 <code>terminate</code>	End of conference

Table 7.3: Communication Cluster Business Operations

### Control Cluster

Control services (ID-Name:400:CL\_CTN) provide central administration of community, content and communication services. As outlined in Section 6.4.4, they also store user-related state information and serve as a repository for central access to this information. Acting as a utility type of service, business operations and business methods in the control cluster can be represented by the list in Table 7.4.

Business Operations	Business Methods	Functionality
UserControl (OP_UC)	411 <code>userInfo</code>	Add, remove and search all transiently and persistently saved user information
UserControl (OP_UC)	412 <code>authorize</code>	Determine permissions based on a user's identity
UserControl (OP_UC)	413 <code>authenticate</code>	Check user's identity
UserControl (OP_UC)	414 <code>locate</code>	Request geographical location of user
UserControl (OP_UC)	415 <code>presenceInfo</code>	Check whether user is currently network-available
UserControl (OP_UC)	416 <code>charge</code>	Calculate charging factors based on user's service use
SessionControl (OP_SC)	421 <code>init</code>	Initialize user access session
SessionControl (OP_SC)	422 <code>sessionInfo</code>	Add and remove transient user session information
SessionControl (OP_SC)	423 <code>terminate</code>	End user session
MediaControl (OP_MC)	432 <code>encode</code>	Encode attached media using media codec
MediaControl (OP_MC)	433 <code>decode</code>	Decode encoded media codec

Table 7.4: Control Cluster Business Operations

### Composition and System Clusters

To complete our systematic itemization of operations, two more utility clusters should be adopted. Composition services will be bundled into a separate cluster (500:CL\_CPS) as independent functionality that operates across clusters. Based on our previous non-functional property analysis, the system cluster (600:CL\_SYS) aggregates system environment functionalities, which are listed in Table 7.6.<sup>19</sup>

In the systems operations, local invocation refers to a service invocation inside a non-distributed (hardware or software-wise) component. Further-

<sup>19</sup>512 decompose: Services can be disassembled into other composed services, atomic services or both.

<b>Business Operations</b>	<b>Business Methods</b>	<b>Functionality</b>
Composition	511 <code>compose</code>	Assemble services into composed services
Composition	512 <code>decompose</code>	Disassemble composed services
Composition	513 <code>composeabilityInfo</code>	Assess the degree of composability of a service in relation to a given goal context

Table 7.5: Composition Cluster Business Operations

more, operations of manual invocation can be used when binding a service to a dedicated service consumer, e.g. using a developer created invocation stub. However, automatic invocation operations can be used, for example, when invoking a service inside a service composition.

In summary, our analysis proposes a breakdown of the telecommunication domain into six functional service clusters. Business operations and business methods have been assigned to these clusters to directly specify the cluster functionality.

The semantic network can be depicted using the UML-based object model as a graphic overview of the business architecture. Domain and utility services are tagged using UML stereotypes, as shown in Figure 7.4.

<b>Business Operations</b>	<b>Business Methods</b>	<b>Functionality</b>
State	611 <code>init</code>	Enrich services with the ability of maintaining state information
State	612 <code>stateInfo</code>	Add and remove transient state information
State	613 <code>terminate</code>	Delete all state information
Transaction	621 <code>init</code>	Enrich services with transaction capabilities
Transaction	622 <code>transactionInfo</code>	Add and remove transaction information (e.g. isolation level)
Transaction	623 <code>rollback</code>	Recover consistent service state after transaction failure
Transaction	624 <code>commit</code>	Save service state persistently
Quality	631 <code>metricInfo</code>	Add, remove and retrieve service quality metrics
Quality	632 <code>qualityInfo</code>	Retrieve service-related quality information at runtime or historically
Communication	641 <code>synchronous</code>	Definition of synchronous communication mode
Communication	642 <code>asynchronous</code>	Asynchronous communication mode
Invocation	651 <code>local</code>	Local invocation of a service
Invocation	652 <code>remote</code>	Invoke a service across components
Invocation	653 <code>manual</code>	Definition of parameter to invoke a service manually
Invocation	654 <code>automatic</code>	Automatic invocation definition

Table 7.6: System Cluster Business Operations

The relationships illustrated in Figure 7.4 are based on four critical concepts:

- Communities and their human-centric alignment can be seen as a central, mediating point of the telecommunication domain, with a high demand for agility and flexibility.
- Content and communication can be integrated into communities seam-

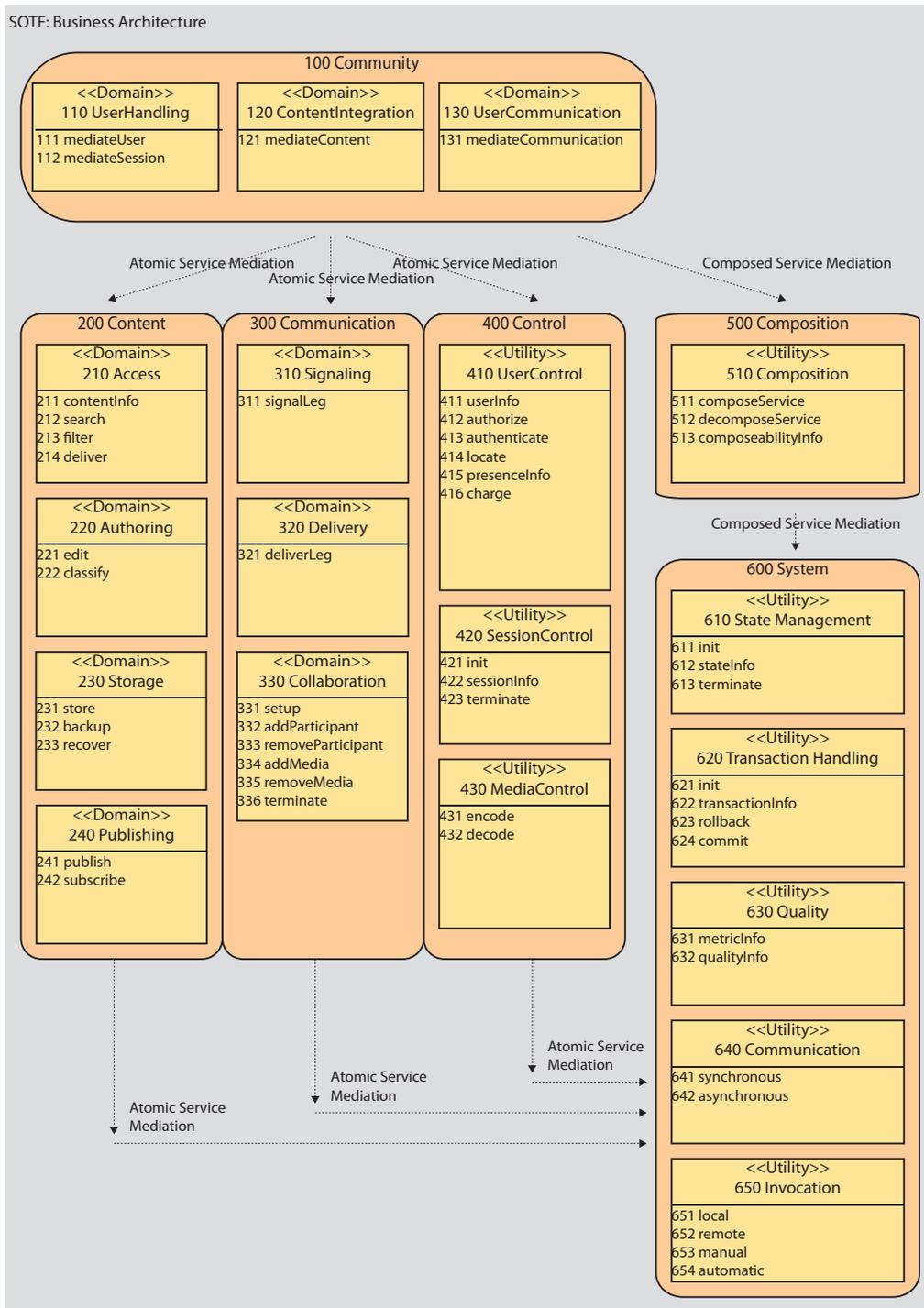


Figure 7.4: SOTF Business Architecture, Services, Clusters and Operations

lessly. Heavily requested by communities, these two clusters maintain their own cluster-specific functionalities.

- Control acts as a strongly cohesive utility, heavily requested by the other clusters.
- System and composition complement the other clusters as cross-cluster operating system functionality.

These service clusters contain the main functionality of the domain. They serve as largely independent architectural building blocks, yet they have a certain degree of dependence. In conformance with the service paradigm principles, they are decoupled and modular.

The next step will be to transform the business services into data and technology architectural views. To this end, computable data entities and components must be identified, representing the informational systems perspective.

## 7.6 Data Architecture

---

The data architecture viewpoint defines the major data entities, and relationships between them, necessary to support the business architecture. Our intention, in developing the data architecture, is to produce a common and extensible view of service data in the domain, and not to provide a logical or physical storage system. [Gro06]

The telecommunication domain integrates huge numbers of real-world entities and several dynamic perspectives. [OAF88] Because of this, it is an enormous effort to capture all of the domain's data in a general model. From a service perspective, the data architecture must provide a common denominator to manage compatibility between the primary interacting entities, the provider, consumer and broker. The relationships among these entities are technically expressed in the service contract.<sup>20</sup> This interconnection implies an understanding of data in which the service contract entities, the Service Level Agreements, determine the main ABBs of the view, hierarchically arranged in a dependent (tree) structure.<sup>21</sup>

---

<sup>20</sup>The service contract was introduced in Chapter 3 as the foundation for composition in the software service lifecycle.

<sup>21</sup>In general, data entities can be structured using basic types such as primitive and

Our investigation of the data architecture therefore concentrates on the development of a generic contract data model, identifying entities to act as a basis for negotiation between telecom provider and broker. The contract data model establishes the foundation to implement the service interface on the basis of a descriptive programming language, well-formed and machine readable.

### 7.6.1 Input and Requirements

Chapter 4 described *metadata*, *policies*, *parameters* and *attributes* as the main elements of a service description. These elements are also valid when interacting dynamically, as in the potential embedding of services in a composition. The semantic discussion in Chapter 6 developed three levels of relationships, the *meta*, *static* and *dynamic* levels, including descriptive clauses for each level. These two perspectives together provide a good starting point for structuring the data entities of a service contract.

The contract entities can also be supported by consulting the Design by Contract (DBC) methodology. DBC fundamentally eases the structuring of a service contract with respect to static, dynamic and quality attributes, accommodating the contract constraints of preconditions, postconditions, rely conditions and invariants (IOPE) to enable the development of more reliable and robust software applications. [CGB05] DBC further supports the computation of constraints using programming elements such as assertions and annotations.

Since the service contract acts as the central service description, composition and processing entity, it must conform to the basic principles of the service paradigm (e.g. loose coupling and modularity), but it must also implement the seven non-functional properties of service composition.

---

complex types. Complex types are formed by assembling primitive types. Primitive types represent entities that cannot be disassembled any further, and thus are the smallest data unit to be explicitly computed.

## 7.6.2 Approach

While primitive data types serve as the basic computable layer of data, complex types facilitate the structuring and further aggregation of entities to support reuse. In general, contracts can be described in such a way as to distinguish between functionality and structure. These two aspects should be labeled in the data description. Among other advantages, the differentiation between functional and structural elements facilitates their translation into common descriptive programming languages such as XML. A common way of supporting this differentiation is the definition of basic types dedicated to the assignment of functionality or structuring.

Our analysis shows data entities as a set of name:type pairs. Functionally, entity types can be derived directly from the basic contractual elements identified above (metadata, policies, parameters and attributes), which leads to a `ParameterType` and an `AttributeType`. To support behavioral constraints, an additional `ConditionType` can be added. The `AttributeType` is used to store information as descriptive (key-value) entities over all service instances, while the `ParameterType` provides service-specific information. The `ParameterType` includes a `Description` and a `Type`. The type tags the direction of processing as `In`, `Out` or `Inout`. The `Force` parameter type describes processing as `Mandatory` or `Optional`.

During processing, parameters can be changed, and one to multiple conditions can be attached to them to define behavioral constraints. For this reason, the `ConditionType` allows the definition of a `Restriction` in relation to a `RestrictedElement`. For example, the input parameter for a location search could be a minimum of three characters which could be defined in the data model as follows: `Type = arithmetic`, `Restriction = less`, `Threshold = 3`, and `Unit = Seconds`. The restriction further contains a `Scope`, enabling the distinction among system, composition or service scopes, and a `Type`, defining the type of restriction, such as arithmetic or comparative (e.g. equals).

Aside from functionality, it is useful to assign structural elements to facilitate processing. A common pattern for structuring a service contract is a tree-based composite arrangement, as defined in the structural patterns of the Gang of Four (GoF). [GHJV00] Using this concept, entities can be marked as `<<Root>>`, `<<Group>>` or `<<Leaf>>` UML stereotypes, defining a position in an assembled structural tree.

Figure 7.5 outlines data entity types for assembling service contracts by disassociating functionality from structure.

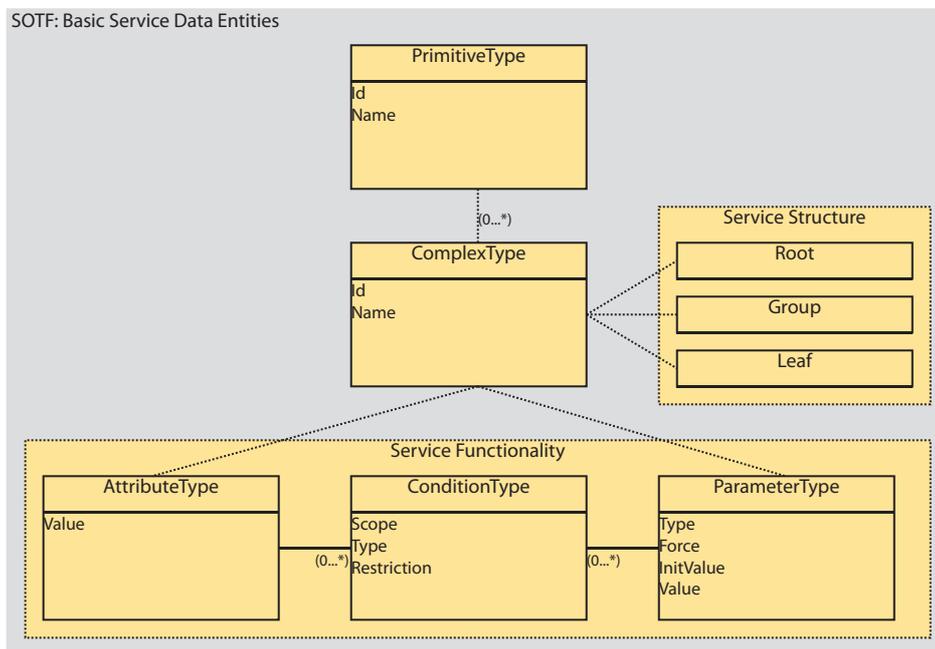


Figure 7.5: SOTF Data Architecture, Basic Service Data Entities

### 7.6.3 Result

Following the basic structure suggested in Figure 7.5, five meaningful group entities may be considered for structuring a telecommunication service contract:

- The **NGNService** is the root or central element of the contract data entities.
- The **Meta** entity aggregates descriptive, global information about the service.<sup>22</sup>
- The **Types** entity structures all information referring to primitive and complex functional data entities.
- The **Methods** entity captures the dynamics of a service and operates above the specified data structure.
- In addition, **Properties** group the outlined (Section 4.7, Section ) non-functional service data.

<sup>22</sup>This information is derived from the semantic levels described in Chapter 6.

In relationship to one another, **Meta**, **Types** and **Methods** describe the explicit functionality of an **NGNService**, while **Properties** can be used to add more meaning to contracts in order to enrich their semantics implicitly.<sup>23</sup> **Properties** data can be embedded in the **Meta** group to define non-functional properties globally for the overall service. Alternatively, **Properties** can be defined per method in the **Methods** section to describe specific method behavior. **Properties** are usually defined during the contract specification to add necessary semantics before registration or lookup. This facilities heavily machine to machine interaction such as the dynamic composition of service contracts.

At this point, when having developed operations and business methods we reached a fine granular level of domain semantics. A business method can be further mapped to one or multiple implementation methods. Implementation methods finally represent the finest granularity of functionality, while combining data and business architecture in a single definition. Implementation methods are part of the finally specified and deployed service contract. They are directly associated with an executable implementation by a service endpoint definition inside of the contract. Since data and business architecture form the conceptual foundation, the specification and deployment of implementation methods will be demonstrated in the prototyping Chapter 8.

---

<sup>23</sup>In subsequent chapters we will continue using the words *explicit* and *implicit* as identifiers for functional (explicit) and non-functional data definitions.

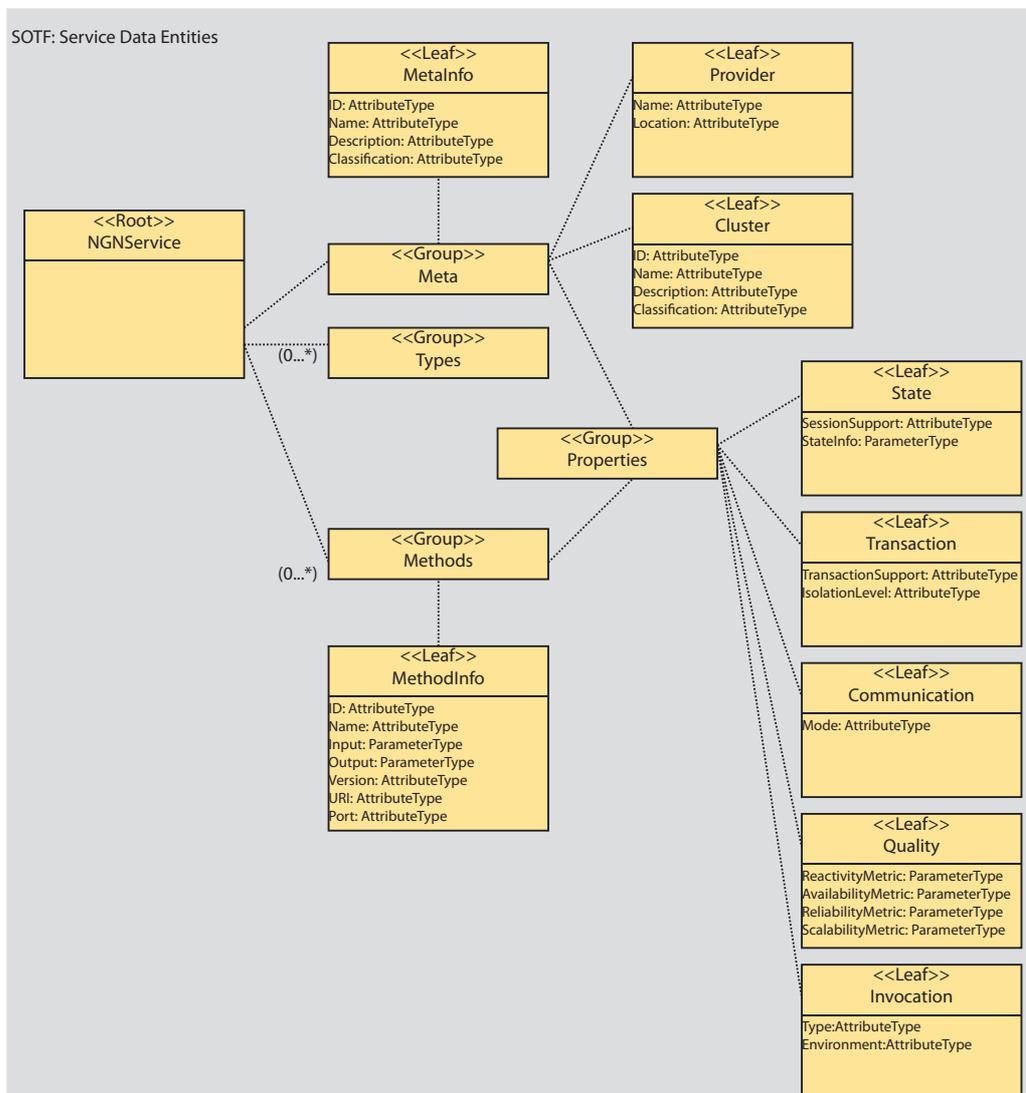


Figure 7.6: SOTF Data Architecture, Service Data Entities

## 7.7 Application Architecture

The purpose of the application architecture is the definition of the major kinds of components necessary to process data and provide business services. The applications and their capabilities are defined without reference to particular technologies. [Gro06] Service clusters and operations are computed by software components, which serve as the main ABBs of this view. The

main software component types can be derived from the discussion in Chapter 5.3.2 as protocols, middleware, and APIs implemented using descriptive and compilable programming languages. Components can contain subcomponents and provide a defined set of interfaces for one-to-one or multiple component accessibility. The outcome of this architectural viewpoint is a component diagram illustrating the layered components and subcomponents and their assigned services and interfaces.

### 7.7.1 Input and Requirements

The identification of appropriate component candidates can be accomplished by considering business services mapped to the service lifecycle. The service model developed in Chapter 6, consisting of six service clusters including services and operations, and the service lifecycle with its eight activities, facilitate a definition of components, subcomponents and interfaces. Additionally, certain design patterns can be used to arrange the components in layers and attach appropriate subcomponents and types. Since service patterns, which describe the arrangement of services, are in a very early stage and not applicable to components [Arn04], our investigation focuses on three common catalogues of design patterns: the GoF patterns [GHJV00]), the Java EE patterns [(SD06] and IBM's proposed catalogue of service patterns [MK04], [Ars05].

The component architecture to be defined must conform to the service lifecycle<sup>24</sup>, where services undergo eight phases from specification to binding to the consumer. Composition, as part of the lifecycle, must also be implemented in conformance with its specific behavioral pattern and non-functional properties. Finally, the architecture must satisfy the characteristics of the NGN infrastructure.

### 7.7.2 Approach

Besides the six clusters of functionality (community, content, communication, control, composition and system) the service lifecycle contains a set of service activities, arranged in chronological order: specify, compose and decompose, register, publish, lookup, negotiate, invoke and bind. Composition, as an embedded activity in the lifecycle, maintains a strong dependency

---

<sup>24</sup>The service lifecycle was introduced in Section 3.5.1.

with service matchmaking, an integrated part of the service lookup, which comprises the critical functions of member detection and verification.

From a service perspective, the clustered business operations and methods must be provided using the service lifecycle in a set of interacting software components. Because agility is an important factor in the design, our component specification is based on three assumptions:

- **Generic encapsulation:** The core software components will be designed to be domain service-independent, by outlining the components according to the service lifecycle. This is accomplished through the encapsulation of the domain specifics (i.e. the community, content, communication clusters) in a generic top layer.
- **Reduced distribution, but with cloneability:** Since the service paradigm supports a highly distributed, almost gridlike arrangement of the entities consumer, broker and provider, the component architecture will narrow this structure to a single component per entity. However, the architecture will allow the addition of new instances with a similar structure. Components can be cloned, for example, to support distribution, openness and scalability.
- **Java alignment:** To accommodate the research scope, the components will be outlined using the Java development community, where a very large set of modular components and service lifecycle implementations exists to facilitate the prototyping process.<sup>25</sup>

The main structure of the components can be layered using a multi-tier architecture approach. Multi-tier architectures and especially their primary representative, the three-tier architecture, promote the layering of software systems according to their domain-independent, underlying system functionality. A three-tier architecture implies a top-down separation of presentation, business logic and persistence functionality.

Bringing together the multi-tier approach with the concept of separating services from control and domain from utilities, a detailed four-layered breakdown of components can be envisioned:

- **Tier 1:** The *domain layer* specifies community, content and communication cluster functionality.

---

<sup>25</sup>Aside from the Java alignment, SOTF includes no implementational dependencies.

- Tier 2: The *utility layer* specifies utility functionality, including system and composition functionality.
- Tier 3: The *control layer* specifies general control of the domain layer in accordance with the specified control cluster.
- Tier 4: The *persistence layer* defines the storage and retrieval of persistent data.

ESBs and app servers were previously introduced as appropriate middleware environments to support service lifecycle processing. Like a container, they serve as a runtime environment for programming artifacts largely developed as reusable APIs. The APIs perform specific tasks and communicate with other APIs as well as with their runtime environment.

Both the container and the APIs it contains provide specific interfaces. There are two types of interfaces observable, outbound and inbound. While outbound interfaces are remotely accessible from outside the container, inbound interfaces are restricted to local access, within the container only. To keep the framework open and extensible, our component specification focuses on outbound interfaces only.

### 7.7.3 Result

Because of the enormous amount of inter-working functionality in the telecom domain, it is useful to compress the specification of software components into a scheme that evaluates each component by a central set of criteria. Three descriptive criteria seem to be sufficient for our software component specification:

- Layer, Type: The architectural layer, where the component is assigned and the type of component.
- Design principles: The underlying design rule, inspired mainly by design implications previously introduced, or by existing design patterns
- Tasks: Tasks to be handled by the components, derived from the service lifecycle
- Interfaces: Open points of interaction to access the components

Based on frameworking's essential motivation toward reuse and component tailoring<sup>26</sup>, the following component specification interprets service lifecycle phases in a telecom-specific way.

As in Kruchten's modeling [Kru98], the prefix I is used to tag interfaces, while component types are labeled with UML stereotypes such as <<Middleware(ESB)>>, <<Middleware(App Server)>>, and <<API>>. Arranged in a multi-tier layering, the components fulfilling the functionality of service design, execution, brokerage, control and persistence so as to provide a complete infrastructure for telecommunication service. Useful design concepts can be found in the collections of patterns mentioned in Section 7.7. Our specification begins with the design components, and execution and brokerage are arranged above the control and persistence elements.

An agile telecommunication service layer can be synthesized by providing the seven software components shown in Table 7.7 and Table 7.8.

Each of the seven components includes their subcomponents, which are all specified in Appendix B.1 using the same criteria shown in the preceding Tables 7.7 and 7.8. To complete the specification, we have arranged the components in a unified UML component diagram. The diagram illustrates the layering, the arrangement of components including subcomponents and the attachment of the relevant interfaces. Communication relationships, dependencies between the components and inbound interfaces are omitted.<sup>27</sup>

---

<sup>26</sup>Tailoring refers to the process of taking APIs from one component for reuse in other contexts.

<sup>27</sup>Each of the seven primary components can be implemented heterogeneously, but each must provide the specified open and accessible interfaces.

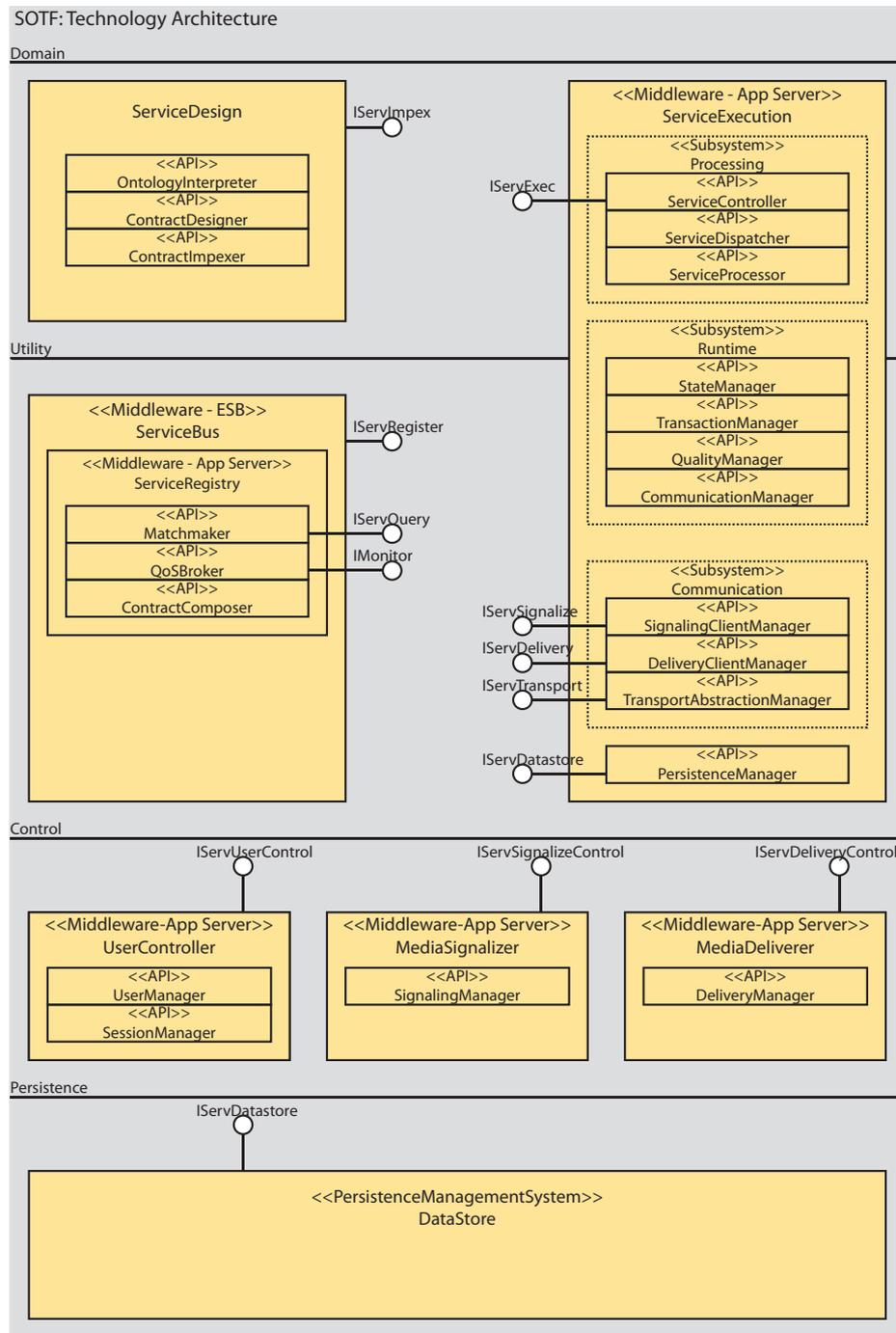


Figure 7.7: SOTF Technology Architecture, Software Components

## 7.8 Composition, Composeability and Matchability

---

We have proposed a structure for a telecom service framework consisting of business services (operations and conceptual methods), data entities and software components. In order to compose services in a framework it is necessary to assess the feasibility of composition of and between services, which we have termed *composeability*. Composeability predicts the ability of a service to provide a given set of functionality in a given context to achieve a given semantically defined goal. Two services that have a low degree of composeability in one context may have a higher degree of composeability in another. For this reason, composeability must be ensured during each composition activity, regardless of whether it occurs during design or runtime and regardless of the type of invocation (manual, semi-automatic, or automatic).

A prerequisite for the composability of a service is its matchability, the quality of being described in a machine-accessible, machine-readable, and machine-understandable manner and hence searchable and deliverable to a potential service consumer. The functionality of checking for composability of a service (usually during service composition, whether the invocation is automatic or manual) is usually carried out by the service composer, a software component. Matchability of a service can also be assured by the exposure (interface) of a well-formed service contract, and it can be verified by the service registry, usually during the registration of a service.

Both of these service composition prerequisites, composability and matchability, should be verified in the eight steps of the service lifecycle, during computation. At this point it seems desirable to introduce a set of checking criteria, an assessment scale and the integration of composeability verification into SOTF.

### 7.8.1 Degree of Composeability

The degree of composeability is highly dependent on the context of the composition, which is represented by the goal service. Given the postulate that a service provides a matchable service description, composeability can be assessed on a graduated scale of 0 to 1, where 0 [zero] represents completely uncomposable structures and 1 [one] an exact match. It is not always neces-

sary to meet complete composeability. Some situations require only partial composeability on the operation or method level.

There are two criteria that primarily serve to assess the degree of composeability between two member services in a composition context: *semantic compatibility* and *quality sufficiency*. On the basis of the previously introduced semantic perspectives<sup>28</sup> and quality model,<sup>29</sup> we can refine these criteria by associating with them the semantic clauses developed in Chapter 6:

- Semantic compatibility:
  - Meta semantic compatibility (ID, Name, Dependencies, Cluster)
  - Static semantic compatibility (State Application, Transaction, Communication, Invocation)
  - Dynamic semantic compatibility (Operation, Constraints)
- Quality sufficiency:
  - In Community, Content, Control clusters (Reactivity, Availability, Reliability, Scalability)
  - In Communication cluster (Delay, Jitter, Loss)

## 7.8.2 SOTF Service Composition

Within the SOTF framework, composability verification is handled on the service layer. The `ServiceRegistry` is responsible for ensuring the required degree of composability, including basic matchability. In greater detail, matchability is assured by the `ContractComposer`, which inspects the syntactic correctness of a service using the `Matchmaker`. In addition, the `ContractComposer` also synchronizes the service with existing ones and provides information about existing functional redundancies during registration of a composed service. Once a composed service is registered, it can be used in the service lifecycle analogously to atomic services.

The registry containing the `ContractComposer` uses the `Matchmaker` and the `QoSBroker` to ensure semantic compatibility and quality sufficiency. It checks for semantic compatibility by comparing each semantic clause to the

---

<sup>28</sup>Semantic perspectives were discussed in Section 6.4.1.

<sup>29</sup>A quality model was introduced in Section 6.4.1.

surrounding member service and the goal service. A set of clause-specific rules guarantee semantic compliance. For example, a rule of transaction handling might be that a goal service requires all member services to be transactive. Another example would be a check for method compatibility, which would include the compatibility of signature elements such as name and input and output data.

Quality sufficiency can be checked each time the composed service is looked up, and the composition could be improved automatically, by replacing non-sufficient services automatically. To this end the `ServiceExecution` can trace the service quality and forward the information to the registry. Transparent to the customer, the `QoSBroker` can substitute, with the help of the `Matchmaker`, a sufficient service for a non-sufficient one, or at least demand the improvement of the service.

## 7.9 Summary

---

Using TOGAF, tailored to act as a general guide for our efforts, this chapter developed a Service-Oriented Telecommunication Framework (SOTF), illustrating business, data and application views of the telecom domain. Based on the semantic domain analysis developed in Chapter 6, and with the aid of frameworking methodology, we have defined a semantic service specification which outlines NGN services as a network of interacting building blocks in a business architecture. The granularity of the building blocks is increasingly refined, from clusters into business operations and further into business methods.

In terms of data structures, SOTF specifies service-contractual entities as the lowest common denominator for describing a service using explicit and implicit functional descriptive data. These entities can be used to implement service contracts as structured, richly described interfaces. SOTF proposes a software component model for computing services in a manner that is compliant with the software service paradigm. The model conceptualizes the handling of non-functional properties as well as service specifics such as real-time communication and quality of service. The structure of the component model is based on the task structure of the Java development platform.

During composition, SOTF provides the capability to verify the composability (including matchability) of a service to ensure semantic compatibility

and quality. Both composability and matchability will be integrated in the prototype.

The next chapter will demonstrate an implementation of the framework as a paradigmatic implementation of the technology architecture developed in TOGAF. Middleware, protocols and API's will be used to provide a service lifecycle based on a real-life scenario. Service composition will be demonstrated using Java technology, adapting the service lifecycle. The demonstration will include specification of contracts, integration of certain heterogeneous programming models such processes and events, registration of services, matchmaking, verification of matchability and composeability and manual and automatic composition, taking into consideration the semantic relationships between services.

Component (Layer, Component Type)	Design Principle	Tasks	Interfaces
Design	Service Design (SOA)	Creation, specification of atomic contracts and manual composed contracts, development of service implementation	IImpexService Import and export of service contracts and implementations
Execution (Domain and Utility, Middleware (App Server))	Service Implementation Provider Entity (SOA), Design / Processing Separation (NGN), Quality of Service Support (NGN), Service Composition Processing (State, Transactions, Quality, Errors, Communication)	Processing of composed and atomic service implementations including quality monitoring and non-functional property handling	IServExec Invocation of service implementations, IServSignalize and IServDelivery initiation of signaling and delivery, IServTransport transport abstraction, IServDatastore
Bus (Utility, Middleware (ESB))	Service Broker Entity (SOA), Decoupling Contract From Implementation (SOA), Brokerage (SOA), Service Leasing (SOA), Automation (Composition)	Publishing of service contracts, service lookup including matchmaking, contract negotiation, and automatic composition	IServRegister Registration of service contracts, IServQuery Matchmaking of service contracts, IMonitor Reporting and monitoring of service execution

Table 7.7: SOTF Component Specification: Design and Execution

<b>Component</b> (Layer, Component Type)	<b>Design Principle</b>	<b>Tasks</b>	<b>Interfaces</b>
<b>User Controller</b> (Control, Middle- ware (App Server))	Service Layer- ing(NGN)	Central main- tenance of user and access ses- sion data	<b>IUsrCtrl</b> User and session data mainte- nance
<b>Media Signalizer</b> (Control, Middle- ware (App Server))	Signaling / Delivery Processing (NGN), Service Clustering (NGN)	Central signal- ing of real-time media streams	<b>ISigCtrl</b>
<b>Media Deliverer</b> (Control, Middle- ware (App Server))	Signaling / Delivery Processing (NGN), Service Clustering (NGN)	Central deliv- ery of media streams	<b>IDevCtrl</b>
<b>DataStore</b> (Persis- tence, Applica- tion (e.g. RDBMS))	Multi-tier Architec- ture	Persistent data storage	<b>IData</b>

Table 7.8: SOTF Component Specification: Control and Persistence

## Part IV

# Applied Service Composition in the NGN



# Chapter 8

## Prototype Implementation

*“I helped building it and my son explains me how to use it. ”*

*Nicholas Bambos, in a meeting about Telecommunication Networks [January, 2007]*

### 8.1 Synopsis

---

The SOTF framework developed in the previous chapter presents a very generic approach applicable to a wide range of components. For this reason it may be useful to demonstrate, verify, and validate the framework’s viewpoints with the aid of a prototype. In this chapter we will introduce a real-world scenario that computes some of the core business services of SOTF. The scenario is implemented in a prototype constructed in the scope of an internal Deutsche Telekom AG research project, Multi-Modal Communication and Collaboration Services (MMCCS).

Above the MMCCS infrastructure, the prototype contains an application, *Activitylife*, whose functionality, for the purposes of our demonstration, is limited to service mediation. The mediator application, in performing this task, uses only the service layer to compute operational logic. The scenario

demonstrates how to perform the service lifecycle steps,<sup>1</sup> most prominently composition. It presents the application layer in the role of a service consumer and the service layer in the roles of a service broker and provider. Computation is restricted to the research scope delineated in Section 5.2.

The prototype implementation is based on the *Java EE* programming environment, extended by open source components such as the Spring framework, and taking advantage of the environment's practically proven concepts such as non-functional property handling (including state, transactions, and synchronous/asynchronous communication), endpoint bindings and messaging. The prototype includes contract descriptions, heterogeneous implementations and several communication protocols and formats. To further elaborate on our previously identified implementation obstacles, the model includes demonstrations of inter-service invocations and circularity (discussed in Section 4.6.5), automation of invocation (Section 4.7.6) and quality/performance (Section 4.7.4), while composing services using diverse patterns (Section 4.6).

The prototyping consists of two main divisions. The first part is the infrastructure, implemented within the scope of MMCCS. The second part is the demonstration scenario, a travel community that we have named *Activitylife*. Where they intersect, *Activitylife* is a sample application implementation in the NGN application layer that invokes the reusable services of the MMCCS infrastructure.

## 8.2 Prototype Specification

---

### 8.2.1 Project Background

MMCCS is a research project driven from the innovation area of multi-access service frameworks, whose purpose is the development of platforms that enable application independence from the network used. In the MMCCS system, a user is offered enhanced communication and control. The user can monitor IP calls running on any phone device, whether mobile or fixed. It is possible to control a call from a PC, for example to upgrade it to a phone conference or transfer the call. The user can enhance a running call with a collaboration session. Furthermore, private service session mobility is provided, which lets the user seamlessly switch over from one device-specific user

---

<sup>1</sup>The service lifecycle is described in Section 3.5.1.

interface to another, even during a running service session. MMCCS is entirely IPv4 based and enables interfacing with common PSTN- or GSM-based gateway technology.

## 8.2.2 MMCCS Software Components

MMCCS consists of software components connected through Java API interfaces and IP-based protocols. In our prototype, the MMCCS components encapsulate the main service operations of SOTF and make them accessible via clearly defined component interfaces. These interfaces each contain an IP-based protocol adapter, which includes the supported IP Protocol (e.g. HTTP) and an endpoint, reachable via the network. The MMCCS components are completely decoupled and thus technologically interchangeable.

Functionally layered in accordance with the SOTF guidelines, MMCCS provides application, service (domain and utility) and persistence components. The application and service layers are decoupled by means of a broker, where services are registered and published to become bound and invoked by applications. Each directly invocable service layer operation provides both service contract and implementation. Since the routing between contract and implementation is done by an endpoint definition, exposed in the contract, here we will refer to the service implementation as the *endpoint implementation*.

In the prototype, services provide *service bindings* to define routings between contract and implementation. Typically, each contract contains one binding, which contains the optimal technical infrastructure for the specific service. In some cases, however, where the service consumer is unknown to the provider, it makes sense to define multiple bindings so that the consumer can choose a preferred binding technology.

The implementation of the communication and control interfaces and their components is based on the general structure of IMS.<sup>2</sup> In particular, the configuration of the Call Service Control Functions (CSCF) and their underlying protocols, SIP for signaling and RTP for delivery, uses an IMS-influenced structure to provide real-time media streams.

The application layer has been reduced to a pure presentation and mediation entity. `Activitylife`, as an example of an application, passes through the client-side requests and invokes the service layer by fulfilling lightweight tasks such as data validation, data transformation or response formatting.

---

<sup>2</sup>IMS was introduced in Section 5.4.1.

Operational logic has been implemented completely in the service layer to demonstrate the service lifecycle.

The main MMCCS components of the service layer, `ServiceBus` and `ServiceExecution`, are capable of providing all of the service lifecycle activities introduced in Section 3.5.1, including composition. Service contracts are published in the `ServiceRegistry` and service (endpoint) implementations are deployed in the `ServiceExecution` environment. This strictly dissociated structure decouples service contracts from implementations and applications from service computation.

During execution, non-functional properties such as state, transaction and communication handling are delegated by the processing subsystem to the built-in Java app server environment using common Java EE APIs. The `ServiceExecution` contains components to initiate real-time communication and others to abstract the persistence layer.

Both atomic and composed services are published in the `ServiceRegistry`. They are equally queryable by means of a common interface. Their behavior in the service lifecycle will be demonstrated in various scenario use-cases in which the service consumer invokes the components via their interfaces.

The MMCCS control layer integrates two loosely coupled app servers: The `UserController` operates as the user data repository, providing access to secure data (e.g. credentials) while maintaining session information independently from the service execution. The `MediaDeliverer` transfers the media stream in accordance with the signaled communication arrangements. In the prototype, SOTF's `MediaSignalizer` has been implemented within the `ServiceExecution`. A local (tighter) coupling between a `ServiceController` (such as a SIP servlet) and the `MediaSignalizer` is much easier to implement and enhances system performance. However, the decoupling proposed in the SOTF architecture is the more elegant design, since it enables interchangeability of the `MediaSignalizer` and decouples services from control completely.

Figure 8.1 illustrates the MMCCS components, including their generic SOTF entities as stereotypes, the Java EE API implementation used and the component interfaces, including their protocols. Where no implementation is noted, the component was developed within this project. In the following sections we will go on to introduce all of the components shown in Figure 8.1, focusing on the functionality used in the project.

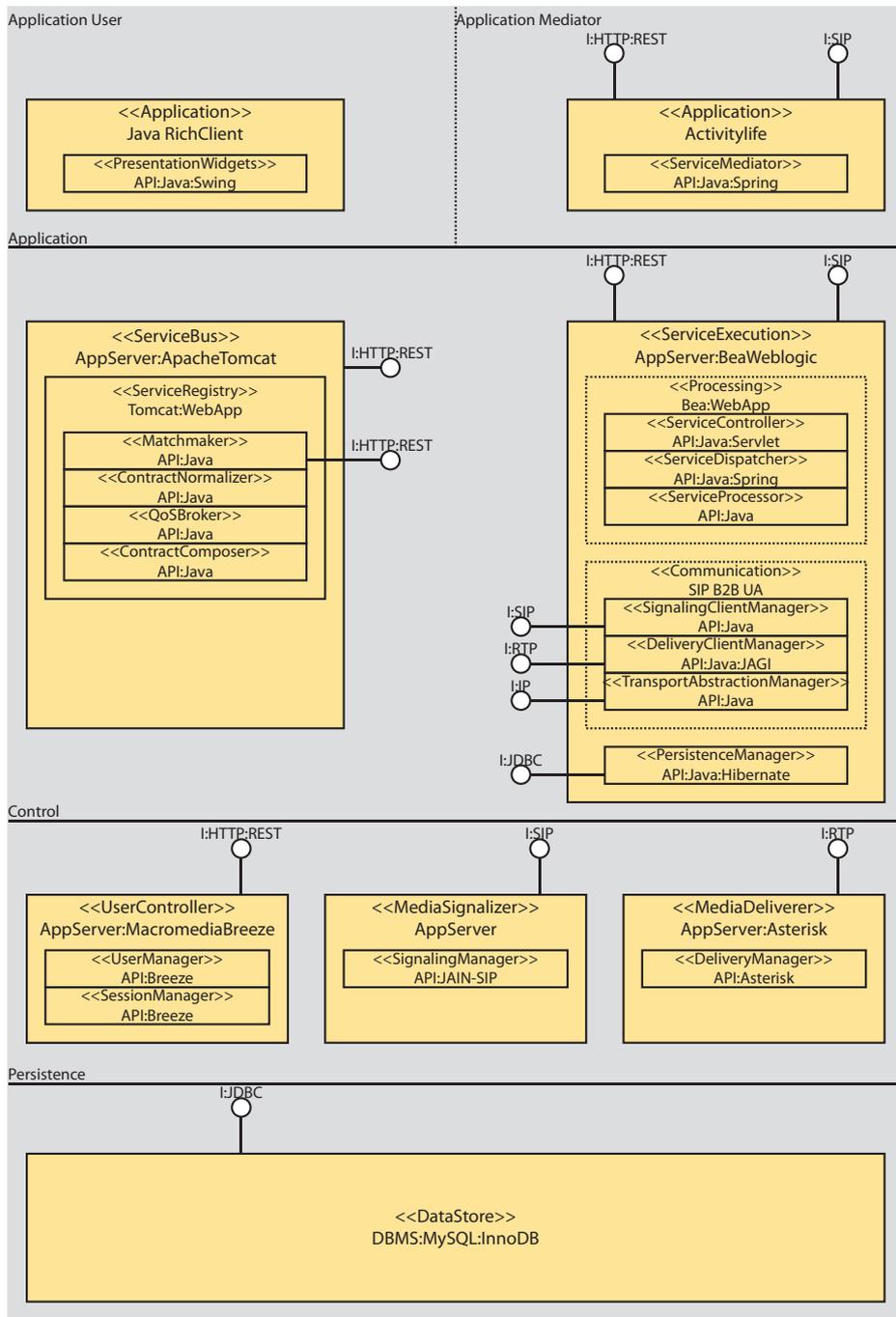


Figure 8.1: Components of the MMCCS architecture

## MMCCS ServiceDesign

For the MMCCS service design we used the BEA Workshop as an integrated development environment (IDE). [Bea07b] This IDE supports the editing of service contract files (such as XML documents, WSDL documents and XML schemas) and service implementations (such as Java classes). It also facilitates deployment by providing module packaging and server upload capabilities.

## MMCCS ServiceExecution

The processing of service implementations in MMCCS is based on the BEA Java app server Weblogic, [Bea07a] which is a runtime environment for packaged Java components known as Enterprise Application Archives (\*.ear). An ear archive consists of several web, logic and/or data access modules. The server provides protocol stacks (such as HTTP, SIP and JMS), making them centrally accessible through programmable `ServiceController` APIs. The `ServiceControllers` are implemented using servlet APIs (which are part of Java EE [Jav07]), in particular HTTP servlets and SIP servlets (JSR 116: [Sip03]). SIP servlets are integrated seamlessly into the Weblogic product family as part of the SIP server [Bea07c], using the basic Java app server as runtime environment.

Incoming endpoint implementation requests are received by a dedicated `ServiceController`, a servlet endpoint usually accessible via a network URI. The `ServiceController` validates the implementation request, extracts the request information (format and parameter) and invokes the processing components.

Once an execution request passes the `ServiceController`, the `ServiceDispatcher` uses the extracted information and associates the request with a logic component, the actual implementation. The `ServiceDispatcher` also detects an appropriate `ServiceProcessor` for the implementation.

In order to associate the request with the implementation component, the `ServiceDispatcher` uses the Spring Java framework [2107], a layered Java/J2EE application framework acting as a lightweight container for Java components. The Spring framework not only supports the `ServiceDispatcher` implementation binding, it also facilitates organizing the `ServiceExecution` environment into layers, using design patterns such as MVC, Bridges, or Adapters [GHJV00], [(SD06], and facilitates the integration of common Java EE APIs through various support objects such as `HibernateDaoSupport`. In the APIs,

on a finer grained level, components are represented by objects to facilitate service processing.<sup>3</sup>

In the `ServiceExecution`, the basic transient data storage objects used are Plain Old Java Objects (POJOs), based on the Java EE pattern of a transfer or value object. [(SD06)] A POJO operates as a data container, transporting data between components within the `ServiceExecution` component.

The lifecycle and customization of the POJOs during handover from component to component is managed by *service facades*. The primary use of facades in in service implementations, based on the Java EE business delegate pattern. [(SD06)] Spring uses dependency injection to manage the POJOs and facades. Dependency injection decouples incoming service requests from their processed implementations based on configuration files, usually written in XML. The following snippet depicts a dependency-injected service facade:

```
1 <beans xmlns="http://www.springframework.org/schema/beans">
3     <bean id="searchItineraryEndpoint"
4         class="do.sotf.endpoint.RestItinerary">
5         <constructor-arg ref="searchItineraryService"/>
6     </bean>
8     <bean id="searchItineraryService"
9         class="de.sotf.service.SearchItineraryFacade"/>
11 </beans>
```

Spring further helps to organize the `ServiceExecution` by providing adapters for transactions (whose implementation is JTA, part of Java EE [Jav07]), for state handling (i.e session handling of the web container), for asynchronous communication (implemented as Messaging, part of Java EE [Jav07]), and for database abstraction (i.e. Hibernate [Lab07]).

After the `ServiceDispatcher` binds a service implementation (a facade) using dependency injection, it hands the information over to the processor. As part of the MMCCS project, the following `ServiceProcessor` components are provided:

- `AtomicProcessor` is the default service processor for atomic services. It processes synchronous or asynchronous communication, administers

---

<sup>3</sup>Components and objects share a useful symbiosis previously described in Section 3.2.3.

parameter and condition handling and manages the state and the transaction of the implementation instance.

- **SIPProcessor** performs SIP session control and routing, which enables services like call conferencing and call transfer. In this way it acts as a back-to-back user agent (B2BUA)<sup>4</sup>

The **SignalingClientManager** is a facade that wraps the **MediaSignalizer** functionality to decouple this component. The **MediaSignalizer** is based on the JAIN-SIP implementation stack, a Java API which fully implements the SIP RFC 3261.<sup>5</sup> The **DeliveryClientManager** uses the JAGI server<sup>6</sup> to remotely initiate real-time media transfers, such as audio or video transmissions. The SIP stack acts as a B2BUA, with the ability to change SIP header fields or to open nested SIP requests (such as requests to the **MediaSignalizer**).

## MMCCS ServiceBus

The **MMCCS ServiceBus** works as a service registry deployed inside an Apache Tomcat web container. [Tom06] Here, Tomcat hosts four components: the **Matchmaker**, the **ContractNormalizer**, the **QoSBroker**, and the **ContractComposer**. All of them are packaged Java application modules, developed for this project.

The **Matchmaker** mainly enables remote and local querying<sup>7</sup> of service contract information. Remotely, the function is used from applications to look up appropriate service contracts and begin negotiation. Locally, the **ServiceComposer** uses the **Matchmaker** to find appropriate member services in order to manually or dynamically compose goal services. Service contracts are registered, either in WSDL format (for atomic services) or as XML documents (for composed services).

A central concept developed in this prototype is that of *contract normalization*, the process of partitioning service descriptions during registration into one unified, queryable data scheme. In the prototype, this data scheme is

---

<sup>4</sup>The B2BUA concept was introduced in Section 5.4.1, using the *SignalingClientManager* facade to control the *MediaSignalizer*.

<sup>5</sup>As previously mentioned, the *MediaSignalizer* is directly integrated into the *Service-Execution* environment to simplify the implementation.

<sup>6</sup>Since our use of the JAGI Server in this project, it has been renamed *Orderlycalls*.

<sup>7</sup>The terms *remote* and *locally* refer to the prototype implementation, where the components are deployed in a single app-server.

based on the SOTF data architecture. To provide multiple, diverse service contracts, the prototype contains an extra component, the `ContractNormalizer`.

In our currently implemented, first phase of MMCCS, the `ServiceRegistry` stores contract information during registration in the filesystem and, in addition, creates a search index of the contract data. The index creation is accomplished by the `ContractNormalizer` using the Apache Lucene API. [Luc07] Lucene provides a rich query language through the `QueryParser`, a lexer that interprets a string into a Lucene query. Within Lucene, contracts are stored as `LuceneDocuments` containing the SOTF data scheme as searchable fields.

We chose to implement the querying process based on the Lucene search because this technology has powerful search and filter criteria and the storage is not restricted to any one format. The disadvantage of this querying approach is that Lucene does not support semantic relationships or, consequently, semantic reasoning between services, nor does it support extended query logic (such as decision logics). [BG04]

It may be worth mentioning that another design possibility would have been to implement the query/reasoning functionality using a combination of the Resource Description Framework (RDF) and its related SPARQL format. [Con07b], [EP07] Exposing RDF entities (i.e. subjects, predicates, and objects), for example in an OWL-S document, would provide a finer-grained, more meaningful data hierarchy than the one used. The disadvantage of SPARQL is that its querying capabilities are restricted to RDF data schemes and SOAP bindings. These technologies imply a tremendous amount of data storage and permanent access to the file system while querying, which could drastically decelerate the service lookup and composition process.

### **MMCCS UserController**

To emulate a central user database, we employed the Macromedia Breeze System, technically another app server. [Bre05] The Breeze solution integrates conferencing functionality, including audio, video and data capabilities. The user database works similarly to an IMS Home Subscriber Server (HSS), centrally storing a minimal user profile.

## MMCCS MediaSignalizer

In the prototype, the `MediaSignalizer` has been directly integrated into the `ServiceExecution` component. It is based on the JAIN-SIP API and is addressable via the `SignalingClientManager` facade. The `MediaSignalizer` takes on the responsibilities of a SIP proxy and SIP registrar, acting as a powerful B2BUA. It uses `UserController` information to initiate the user data and then aggregates all information into a SIP session. A SIP session is a media session based on a SIP URI, such as `sip:user@activitylife.com`. This identifier uniquely identifies the user agent by sending a SIP REGISTER request to the `MediaSignalizer`. The `MediaSignalizer` registers the agent as active for the duration of the SIP session.

## MMCCS MediaDeliverer

The final delivery of media is done by the Asterisk engine, an open source telephony app server. [Ast07] Asterisk has a very modular architecture, allowing the on-demand integration of functionality by a module architecture. MMCCS uses the codec encoding/decoding module to deliver audio and video streams controlled by a SIP interface. SIP is supported in Asterisk with the `chan_sip.so` module.

## MMCCS PersistenceManagementSystem

To store persistent data, MMCCS uses MySQL, in particular its InnoDB engine, to demonstrate transaction handling and support referential integrity. [MyS07] In the `ServiceExecution` environment, the Hibernate API has been employed as a `PersistenceManager` to facilitate persistent data handling and to map relational data into objects. [Lab07]

## MMCCS User Equipment

To access the system, we developed three types of end user clients:

1. A fully featured PC client including telephony control, soft phone, collaboration client and presence list. The PC client is a standalone Java application.

2. A PDA client, including a presence list. This client was mainly developed to demonstrate the possibility of porting the JAIN SIP stack to mobile Java.
3. IP phone hardware.

Both the PC and PDA clients are capable of communicating with the core network through SIP and HTTP. The SIP communication relies on standard IETF SIP, with some MMCCS-specific extensions implemented using SIP IM Messages. The IP phones are standard Siemens Gigaset A140 DECT phones, connected to the IP network over a SIPURA SPA-2000 VoIP adapter.

## 8.3 Prototype Scenario

---

### 8.3.1 Scenario Introduction

The SOTF framework enables the reuse of software-realized domain logic through the conceptualization of a software service layer between applications and transport. The conceptualization includes the organization of architectural building blocks from the business, data and application perspectives.

To demonstrate the capabilities of SOTF, we assume the existence of two general roles: the provider, who designs, implements and maintains SOTF building blocks, and the consumer, who requests services and thus forces a certain behavior from the building blocks. In Chapter 1, we divided the provider's role into sales, service and network provider. Through the decoupling of service and application logic, a fourth provider role emerges, the *application provider*. In SOTF, an application provider is integrated into the service lifecycle as a service consumer, mediating between end user and service provider. To verify and demonstrate this mediation process, and especially the relationship between application and service provider, we constructed the scenario that is presented in this chapter.

The scenario is based on *Activitylife*, a self-organizing travel community that provides a basic portfolio of telecommunication services, such as searches of travel content and communication between community members. The user, by invoking the application, transparently invokes the domain and utility services of the MMCCS infrastructure, using the MMCCS-supported user

equipment and the Activitylife platform as mediator. To separate the services from the application, we further assume a telecommunication provider, *ConnectME*.

### 8.3.2 Scenario Assumptions

Before describing our scenario use cases in detail, we will present some scenario assumptions that will be helpful in facilitating the demonstration and focusing it on the essential aspects of the prototype.

**ConnectMe's Roles and Organization** ConnectME takes the role of a general service provider, offering transport, service, and application capabilities which include a transport abstraction layer, a control and service layer, and an application layer (Activitylife). Although in this scenario one provider provides all of the layers, the components are created in such a way that they can be completely decoupled. To demonstrate a simple decoupling, ConnectME is assumed to be organized into three divisions:

- The *network division* provides gateway functionality as a software API, focusing on the interoperability between pure transport capabilities.<sup>8</sup>
- The *service division* is responsible for the telecommunication logic based on the cornerstones of this thesis. It develops services based on the service lifecycle and makes them available to the application division. The service infrastructure is built on the SOTF perspectives. It implements control services to administer the infrastructure centrally.
- The *application division* manages the travel community, Activitylife, with minimal computational logic, working, instead, as a mediation gateway to the service layer, to maximize service reuse.

---

<sup>8</sup>Since legacy integration and transport/network interoperability are not within the focus of the prototype, this division is not included in our use case discussions.

**Implementation Limitations** The prototyping process does not elaborate on configuration or software installation and customization procedures. The implementation is limited to the use of the IP protocol stack and the Java EE programming environment.

**Focus on Operational Services** The implementation is concentrated on the development of operational services, such as those targeted by the SOTF specification and its six-cluster service definition.<sup>9</sup> Accordingly, Business Support Services (BSS) such as product ordering or billing will not be considered.

### 8.3.3 Scenario Use Cases

Use cases serve as sequential aggregations of operations with the purpose of analyzing a specific way of using a software system. A use case model consists of actors, use cases and the relationships among them. [Con07a] We have identified the human end user and ConnectME as the main actors. Within ConnectME, three divisions cooperate to maintain the infrastructure on the application, service and network levels.

The prototype demonstrates `searchItinerary` and `communicateToUser` as user-invoked functionality in `Activitylife`. However, the emphasis of the prototype discussion is not on the use cases, but rather on the interaction between software components and the implementation of the service lifecycle.

The use cases shown in Figure 8.2 mix end user activities and the service lifecycle.

Beginning with the service division, in a bottom-up analysis, in the next section we will demonstrate that a service architecture must be accompanied by an organization that reflects the service paradigm, a decoupled, service-oriented organization operating software building blocks with various service lifecycle responsibilities. Further assumptions and conditions of the use case execution will be made within the following actor-centric prototype specification.

---

<sup>9</sup>The six-cluster service definition is elaborated in Chapter 7.5.

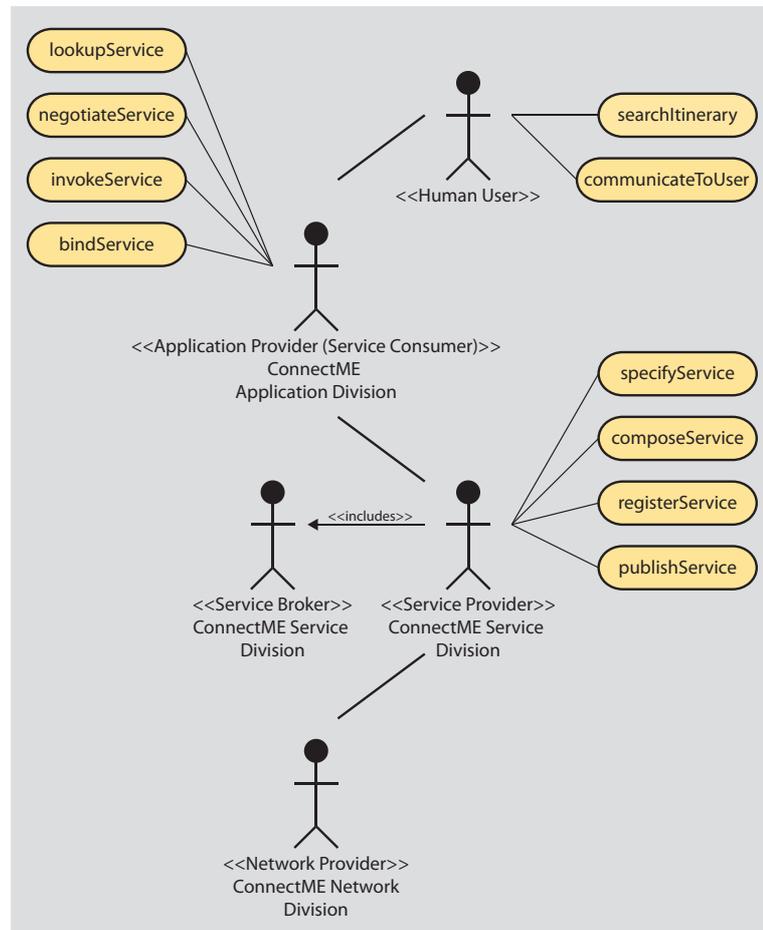


Figure 8.2: Scenario Use Cases

## 8.4 ConnectME, Service Division

The service division of ConnectME is responsible for the specification and provisioning of reusable, operational telecommunication services, in conformance with the software service paradigm. Furthermore, the services must be provided with open accessibility, meaning that they must be usable even in an extra-company scope and in heterogenous service consumer environments. The service division acts as the central hub of computation logic in ConnectME, providing open interfaces to enable communication with the application layer.

### 8.4.1 Overview

The prototype of the service layer is intended to disassociate contract from implementation, and application from service logic, on each implemented service. The service layer is accessible through various protocols and formats, and it supports multiple contract and implementation technologies. It must also enable composition design and processing. An additional goal was to implement composition in such a way that it is transparent to the service consumer, meaning that the service consumer does not know whether the service being consumed is atomic or composed.

The development of the prototype was based on the MMCCS infrastructure and the three viewpoints of the SOTF architecture. In this section we will demonstrate the specification, implementation, registration and, finally, composition of software services.

### 8.4.2 Use Cases

The primary task of the service division is the production of software services. For this reason, the division maintains a **ServiceBus** and a **ServiceExecution** environment, which assume the roles of service broker and service provider respectively. In accordance with the service lifecycle, services are *specified and implemented* as a contract (endpoint)-implementation pair. After specification, service contracts are *registered* at the **ServiceRegistry** and, completely decoupled from the contract, service implementations are deployed into the **ServiceExecution** environment. Finally, the service division supports *composition* by manual or dynamic aggregation of services in a composition description. Composition descriptions, too, are registered at the **ServiceRegistry**. All three activities, specification, registration and composition, are discussed in the sections that follow.

#### Specify, Implement Service

Services<sup>10</sup> are specified based on the general semantic structures developed in the SOTF business and data architectures. While the business architecture forms a basic functional understanding based on business operations and

---

<sup>10</sup>The specification and implementation use case in this section refers to atomic services. The specification of composed services will be demonstrated in the composition use case. In the context of the prototype implementation, a composed service is specified using a composition description, the composite contract.

methods, the data architecture complements that functionality with a generic descriptive data scheme on the method level.<sup>11</sup> In greater detail, the data architecture distinguishes between the **Meta**, **Types** and **Methods** entities to describe the data structure of a service. As we noted in the data architecture discussion in Section 7.6, services define their data structure in the service contract. The service contract thus provides the basis for the subsequent service lifecycle activities of lookup (including matchmaking), negotiation and composition.

**Specification of the Service Contract** The specification of a software service in the prototype starts with the design of the service contract.<sup>12</sup> The contract specification is based on the basic types proposed in Section 7.6: **AttributeType**, **ParameterType**, and **ConditionType**. These are defined using an XML Schema.<sup>13</sup> XML Schemas are qualified to serve as general data definitions, as they are human- and machine-readable and, further, can be computed in various service description languages such as WSDL or plain XML.

**The Data Entities Problem** Data entities presented a particularly interesting problem in the implementation of the service contract. The SOTF data entities represent a semantic model for organizing telecommunication service information. Internally, the data architecture contains two types of data: *explicit data*, which is explicitly related to a function, and *implicit data* which is only implicitly functionally related. For explicit data there is a minimal set of descriptions required to describe the functionality of the service. In SOTF, this data is represented by the **Types** group entity and the **MethodInfo** entity. These data entities are sufficient to develop and use a simple atomic service.

To enable a more accurate service lookup and matchmaking, it is necessary to provide implicit data semantics that describe the non-functional properties of the service beyond the actual functionality. To facilitate composition, most of the generic data in SOTF is dedicated to such description. The

---

<sup>11</sup>An example of the implementation of the NGN Service contract data model is attached in Appendix C.1.

<sup>12</sup>We can distinguish between two general design strategies: *contract-first* and *contract-last*. We have chosen the contract-first method to assure complete independence between contract and implementation. The contract-last strategy, in generating the contract from the implementation, tends also to generate dependencies between them.

<sup>13</sup>The complete meta type definition can be found in Appendix C.1.

Meta group and the Invocation, Quality, Communication, Transaction, and State entities are provided to describe a service in a meaningful way and provide better search and composition capabilities.

The atomic service contract implementation in the prototype is developed in the WSDL 1.1 format. [EC01] WSDL 1.1 currently supports the following elements: types, messages, operations, port types, bindings, ports and services. While the first four elements are relatively abstract and can be used to specify a contract, the latter three deal more with the actual service implementation.

With the aim of mapping the SOTF entities into the WSDL structure, and overcoming the WSDL limitations regarding expressivity in the contract's implicit data definition, we adopted the following implementation:

- Explicit functional data definitions are directly supported in WSDL 1.1. **Types** are mapped into the types element, while **MethodInfo** is defined in the message element.
- Implicit data, since it is not supported in the WSDL standard, is defined in external XML schemas. Two schemas are provided: For the **Meta** group, the **MetaProperties** schema defines information about the entire service, and another, the **MethodProperties** schema, provides the capability to define method-specific non-functional properties such as those mentioned above (in the brief discussion of implicit data in SOTF). Implicit, non-functional data can be defined during specification of the service or at runtime.

Chapter 6 clarified that the key to efficient service lookup and composition can be found in meaningful semantics, both explicit and implicit. The following code exemplifies the straightforward wrapping of the data definition of the **searchItinerary** service in a WSDL document:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wsdl:definitions
3   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
4   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
5   xmlns:schema="http://activitylife.com/schemas"
6   xmlns:tns="http://activitylife.com/services"
7   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
8   targetNamespace="http://activitylife.com/services">
9   <wsdl:types>
10    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```

11     <xsd:import namespace="http://activitylife.com/schemas"
12       schemaLocation="SearchItineraryTypes.xsd"/>
13   </xsd:schema>
14 </wsdl:types>
15 <wsdl:message name="SearchItineraryRequest">
16   <wsdl:part name="SearchItineraryRequest"
17     element="schema:SearchItineraryRequest"/>
18 </wsdl:message>
19 <wsdl:portType name="ConnectMeServices">
20   <wsdl:operation name="SearchItinerary">
21     <wsdl:input name="SearchItineraryRequest"
22       message="tns:SearchItineraryRequest"/>
23   </wsdl:operation>
24 </wsdl:portType>
25 <wsdl:binding name="SearchItineraryBinding"
26   type="tns:ConnectMeServices">
27   <binding verb="POST"/>
28   <wsdl:operation name="SearchItinerary">
29     <operation location="urn:/#SearchItinerary"/>
30     <wsdl:input name="SearchItineraryRequest">
31       <mime:content type="application/x-www-form-urlencoded"/>
32     </wsdl:input>
33   </wsdl:operation>
34 </wsdl:binding>
35 <wsdl:service name="SearchItineraryService">
36   <wsdl:port name="SearchItineraryPort"
37     binding="tns:SearchItineraryBinding">
38     <address
39       location="http://test.activitylife.com:8080/searchItinerary"/>
40   </wsdl:port>
41 </wsdl:service>
42 </wsdl:definitions>

```

**MetaProperties and MethodProperties** Since WSDL currently does not support the direct embedding of implicit service data,<sup>14</sup> represented in the SOTF data architecture by the **Properties** data group, it was necessary to implement a workaround for our prototype. To answer this need, we developed an additional schema to integrate such properties, usable to define properties in the context of **Meta** or **Methods** information of an NGN service, as described in the SOTF data architecture. The subsequent example shows simplified code of the definition of the **Properties**, according to the SOTF data architecture:

<sup>14</sup>However, the W3C recently proposed a working draft for using semantic annotations in WSDL [JF07].

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema
3   xmlns:xs="http://www.w3.org/2001/XMLSchema"
4   xmlns:al="http://activitylife.com/schemas"
5   targetNamespace="http://activitylife.com/schemas"
6   elementFormDefault="qualified">
7   <xs:include schemaLocation="NGNServicePropertyTypes.xsd"/>
8   <xs:element name="Properties">
9     <xs:complexType>
10      <xs:sequence>
11        <xs:element name="StateManagement" type="al:StateManagementType"/>
12        <xs:element name="TransactionHandling"
13          type="al:TransactionHandlingType"/>
14        <xs:element name="Communication" type="al:CommunicationType"/>
15        <xs:element name="Quality" type="al:QualityType"/>
16        <xs:element name="Invocation" type="al:InvocationType"/>
17      </xs:sequence>
18      <xs:attribute name="Type" use="required"/>
19      <xs:attribute name="Identifier" use="required"/>
20    </xs:complexType>
21  </xs:element>
22 </xs:schema>
```

The data schemas provide a basic data definition for all service contracts, whether they are atomic or manually or dynamically composed. In the prototype, this data definition acts as a mandatory foundation for every service to be implemented.

**Properties Definitions** Properties definitions must be integrated in atomic WSDL definitions. The property definition strategy used determines, in particular, the way in which integration is done. Two strategies of property interpretation and definition can be distinguished: design time definition, which is a static strategy in which interpretation takes place in the service lookup, or runtime definition, a dynamic strategy where both definition and interpretation take place during invocation.

The two strategies can be combined to achieve a more efficient description of services. In the prototype, the **Properties** of atomic services are described at design time, while the **Properties** of composed service are created at runtime, based on the properties of the embedded member services. The **Properties** are usually described globally, as **MetaProperties**. **MethodProperties** have been used rarely, most often for the definition of a specific quality metric of a service method.

In the specification, design time properties are attached, along with the WSDL document, as an additional XML document, using the `MetaProperties` and `MethodProperties` as data templates. As an example, the following code snippet illustrates the implicit data of the `searchItinerary` service:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <al:Properties
3   Type="MetaProperties"
4   Identifier="200:210:212"
5   xmlns:al="http://activitylife.com/schemas"
6   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7   xsi:schemaLocation="http://activitylife.com/schemas
8     NGNServicePropertyTypes.xsd">
9   <al:StateManagement>
10    <al:SessionSupport>
11      <al:Id>610001</al:Id>
12      <al:Name>IsEnabled</al:Name>
13      <al:Value>TRUE</al:Value>
14    </al:SessionSupport>
15    <al:SessionInfo>
16      <al:Id>610002</al:Id>
17      <al:Name>User</al:Name>
18      <al:Type>IN</al:Type>
19      <al:Force>MANDATORY</al:Force>
20      <al:Value></al:Value>
21    </al:SessionInfo>
22    ...
23  </al:StateManagement>
24  <al:TransactionHandling>
25    <al:TransactionSupport>
26      <al:Id>620001</al:Id>
27      <al:Name>IsEnabled</al:Name>
28      <al:Value>FALSE</al:Value>
29    </al:TransactionSupport>
30    <al:IsolationLevel>
31      <al:Id>620002</al:Id>
32      <al:Name>IsolationLevel</al:Name>
33      <al:Value>read-committed</al:Value>
34    </al:IsolationLevel>
35  </al:TransactionHandling>
36  ...
37  <al:Quality>
38    <al:ReactivityMetric>
39      <al:Id>630001</al:Id>
40      <al:Name>ReactivityMetric</al:Name>
41      <al:Type>METRIC</al:Type>
42      <al:Force>MANDATORY</al:Force>
43      <al:Value>TRUE</al:Value>
```

```

44     <al:Conditions>
45       <al:Id>6300011</al:Id>
46       <al:Name>ReactivityMetricCondition</al:Name>
47       <al:Scope>Service</al:Scope>
48       <al:Restriction>
49         <al:Type>Arithmetic</al:Type>
50         <al:Restriction>Less</al:Restriction>
51         <al:Threshold>1</al:Threshold>
52         <al:Unit>Seconds</al:Unit>
53       </al:Restriction>
54     </al:Conditions>
55   </al:ReactivityMetric>
56   ...
57 </al:Quality>
58 <al:Communication>
59   <al:Mode>
60     <al:Id>640001</al:Id>
61     <al:Name>Synchronous</al:Name>
62     <al:Value>TRUE</al:Value>
63   </al:Mode>
64 </al:Communication>
65   ...
66 </al:Properties>

```

Together, the WSDL document (explicit data) and the XML document (implicit data) describe the atomic service `searchItinerary`, consisting of one method, `searchItinerary`, semantically sufficient for integration in a service composition.

The disadvantage of specifying services in a WSDL/XML combination is that the registration process must process multiple documents and formats to make them available for the subsequent activity, the service lookup. The complexity of the registration process is resolved by the `ContractNormalizer`, which performs the task of slicing service contracts into the unified SOTF schema during registration. Contract normalization is used for normalizing both atomic service contracts (using a WSDL/XML combination) and composed contracts (XML only). The `Matchmaker`, as the main component of the service lookup activity, searches services against the Lucene registry index, which is the outcome of the service normalization process.

Such a method, one that, like `searchItinerary`, is described using both a WSDL document and an XML document, might be called an *implementational method*. Unlike the conceptual methods of the business architecture, implementational methods are directly routed to an implementation via an endpoint binding. These methods represent the finest granularity of service

functionality description, and the part which is actually executed. Usually, one or more implementational methods are aggregated in one service contract. Bringing together data architecture (e.g. `Inputs` and `Outputs` as `ParameterType`) and business architecture (e.g. `Name`), implementational methods present the core design challenge. The subsequent activities, service lookup and composition, focus on these contract entities while searching or composing services. For that reason, implementational service methods require a clear specification. In our prototype their specification is based on the SOTF specification and principles introduced in Section 7.5.

**Endpoint Routing and Service Bindings** The endpoint routing in a service contract is done in WSDL via a binding definition and a concrete address assignment in the service element. Services can contain one or multiple bindings, depending on how much flexibility is necessary during invocation. The SOTF services that have been prototyped use two types of service bindings: `HTTP.POST` and `Java` bindings. `HTTP.POST` bindings are used to invoke atomic, non-real-time service. To reduce the complexity of service bindings, requests are made via an `I:HTTP:REST` interface, exclusively using the HTTP POST method, where request information (e.g. parameters) can be encoded in the HTTP payload. `I:HTTP:REST` interfaces are a very simple, generic way to encode data interchange between components via a `URI:PORT` address combination.

However, `Java` bindings have been used in the prototype to invoke composed services and real-time services. The address of a `Java` binding is a Java class, a proxy object which handles the invocation. Composed services use a unified `CompositionStub` to control the invocation of a composition, while real-time service invocation is based on the SIP protocol and uses one stub per SIP method to initiate communication. Both bindings are further detailed in the coming sections.

**Endpoint Implementation** After the specification of the service contracts, the actual endpoint implementation must be designed and deployed. Generally, each contract uses its own binding definition to map to one endpoint, where all implementational service method invocations are routed. The prototype groups the methods as accurately as possible according to the business architecture operations developed in Section 7.5. As this implies, the prototype contains one service contract per business architecture operation. As the operations are semantically homogeneous, it is therefore not necessary to use more than one binding per service.

Once invoked, the endpoint address routes the consumer directly to the endpoint in the `ServiceExecution` environment, either via the `URI:PORT` or the stub object. This routing can be changed easily by changing the endpoint address in the service contract. The `ServiceExecution` environment then assigns a Java facade object to the incoming invocation. This facade assignment is done by the `ServiceDispatcher` in the `ServiceExecution` component using the dependency injection of the Spring framework.<sup>15</sup>

**Service Contract Specification Summary** During specification, each atomic service contract must be specified as a WSDL document, optionally attaching implicit data definitions (`Properties`) as additional XML documents. Once the contract is specified and implemented, a service binding connects the contract with the implementation via an endpoint address (one per binding). Additionally, endpoint implementations can be created (e.g. as Java facades using a POJO data container) and deployed into the `ServiceExecution` environment. Finally, the endpoint implementation must be associated with the endpoint address in order to fully connect the service for the process of registration. This association is done in the `ServiceExecution` component using the dependency injection pattern.

## Register Service

A crucial step in the service lifecycle is the registration of the service at the `ServiceBus`, also known as the `ServiceRegistry`, which plays the role of a broker. Located in the service division, the `ServiceRegistry` is the single point of contact to register or look up and negotiate contract data. For registrations, the prototype registry provides a GUI to upload contracts (as WSDL documents). Starting the upload, the contract is syntactically and semantically validated (by means of a syntax checker and schema correctness, respectively). Non-valid contracts are rejected for registration.

If validation is successful, the contract is stored in the filesystem of the service registry. Simultaneously, the service is parsed into its individual parts to create a matchmaking index. To this end, the `ContractNormalizer` component<sup>16</sup> disassembles the contract into the data entities of the SOTF data architecture: `Meta`, `Types`, `Properties`, and `Methods`. This indexing has two purposes: First, to provide a single search interface for atomic and composite

---

<sup>15</sup>The Spring framework dependency injection is discussed in Section 8.2.2.

<sup>16</sup>The `ContractNormalizer` was specifically developed in the context of the MMCCS project.

services,<sup>17</sup> and second, to speed up the matchmaking process by allowing queries against an index rather than traversals through the available contract instances.

For potential service consumers, the indexing enables searching and filtering across all contracts, and allows users to look up services transparently, whether the services are atomic or composed. This latter concept we have termed *composition transparency*. The indexing also hides the complexity of a composition from the service consumer, which should be adopted as a basic rule for all technical service composition approaches.

### Compose Service Contract

Following registration, atomic services are ready for lookup, negotiation, invocation, binding and composition. The composition vision of this thesis was described in Section 7.4.2 as the manual and dynamic creation of a goal-optimized service contract, reusing existing services (members) while considering semantic compliance (composability and matchability) and quality of service.<sup>18</sup>

As we have seen, a service composition can be created manually or dynamically. The difference lies in the way in which the member services, and their control and data flow, are assembled. Manual composition requires a human designer to define the members and their control flow. Dynamic composition requires only a minimum set of input data to detect appropriate member candidates, and assembles the composition contract without human intervention. The `ContractComposer`, a sub-component of the `ServiceRegistry`, supports both types of composition.

The analysis in Section 5.8 assessed current composition approaches (most of which are based on web service composition) as only partially suitable for telecommunication services. Based on that analysis, three main challenges come to light: the integration of event-based real-time communication, non-functional property handling and the assurance of composition quality.

To overcome these obstacles, in Section 7.7 SOTF presented three decoupled components, which are developed in the prototype as part of the `ServiceRegistry`: the `Matchmaker` to facilitate detection of members and

---

<sup>17</sup>For this reason, indexing is also called *service normalization*.

<sup>18</sup>In our prototype, the outcome of composition may be called a *composition description* or a *composition contract*.

their composability, the `ContractComposer` to validate and generate composition contracts and the `QoSBroker` to assess and monitor the quality of atomic and composed services. These components serve both manual and dynamic composition, based on a process and event computation prototyped in the simple example of the two services, `searchItinerary` and `communicateToUser`.

In the prototype, composition contracts, like atomic service contracts, are stored in the `ServiceRegistry` in a, for this project developed, XML format. Like their atomic counterparts, they contain `Meta`, `Types`, `Properties`, and `Methods` descriptive entities, and a `Flow` section.

In the composition contract, the newly added `Flow` element is closely related to the inherent `Methods` elements, defining the composition path of the methods via a reference element, the `FlowRef`, for each method. The `FlowRef` element defines the behavior along the composition path, using connector and transition nodes as developed in the basic composition syntax in Section 4.8. Connector nodes define the behavior along the composition path (i.e. `START`, `END`, `SEL`, `CH`, `FORK`, `MERGE`, `JOIN`, and `LOOP`), transition nodes define the data flow (i.e. `AND`, `OR`, `XOR`) and condition handling (`PRE`, `POST`, `INV`, and `REL`) between them.

The following code snippet illustrates the same method as in the atomic example, except that it is specified as a composed method implementation using a `MethodFlow` section:

```

1  <!-- Meta, Types, and Methods Similar to Atomic Service -->
2  ...
3  <al:Flow>
4    <al:Id>601-24</al:Id>
5    <al:Name>FlowRef</al:Name>
6    <al:SequenceBehavior>
7      <al:Connector>START</al:Connector>
8      <al:ServiceReference>
9        <al:ReferenceValue>searchActivitiesService\#searchActivities
10       </al:ReferenceValue>
11       <al:ReferenceType>Name</al:ReferenceType>
12     </al:ServiceReference>
13     <al:Transition>AND</al:Transition>
14     <al:ServiceReference>
15       <al:ReferenceValue>searchFlightsService#searchFlights
16       </al:ReferenceValue>
17       <al:ReferenceType>Name</al:ReferenceType>
18     </al:ServiceReference>
19     <al:Connector>END</al:Connector>

```

```
20     </al:SequenceBehavior>  
21 </al:Flow>
```

In the prototype implementation, composition behavior has been limited to sequences (exclusive and multiple) and decisions so as to focus on both manual and dynamic composition creation. Those two behavioral patterns are easier to automatically create than the complex behaviors of parallelism and looping. The prototype process has shown that the complexity of service composition implementation lies primarily in specification (especially dynamic creation) and invocation. During invocation, the invoking component must extract explicit and implicit data, add runtime specifics and initiate communication. Mastering the optimization of the invocation, under the constraint of the given parameters, is critical.

For invocation, the prototype suggests a straightforward design: All composed contracts are routed to the same endpoint address, the `CompositionStub` component. This component routes potential service consumers of composed services to a dedicated `CompositionController`, a central entry point for all composed services in the `ServiceExecution` component.

**Manual Composition** To describe the procedure of composing more in depth, we will start with *manual composition*, performed by a composition designer, i.e. a software developer using an IDE. The service designer uses the `Matchmaker` engine to query appropriate atomic member candidates using the matchmaking index, created by the `ContractNormalizer`. Query criteria can be any combination of descriptive entity types such as `Methods`, `Types`, or `Properties`. In addition, Lucene query features such as wildcards or fuzzy matches can be used. When results are found, the `Matchmaker` returns collections of the identified service contracts. The services can be referenced in the `Flow` section, using their unique ID or `Name` identifiers. Once the `Flow` section is built, the flow can be referenced in the method section.

After completing the design process for each method, the designer can register the service in the `ServiceRegistry`, as with atomic services. During this registration, the `Matchmaker` recognizes a composite service and hands the service over to the `ServiceComposer`. The composer component then checks the composability of the services and creates the `Meta` information, which is calculated from the composition members. This calculation is based on a ruleset implemented in the `ServiceComposer`. As an example, if the composer finds one composed service member that is transactive, it marks

the complete service as transactive. The composability check includes checks for input and output compatibility and for quality assurance.<sup>19</sup>

When the composability checks are completed successfully, the service is indexed by the `ContractNormalizer` and stored in the `ServiceRegistry` filesystem, queryable, negotiable and invocable, similar to atomic services.<sup>20</sup>

**Dynamic Composition** Unlike manual composition, *dynamic composition* assembles members using an algorithm rather than a human designer. To trigger a dynamic composition, the service consumer must hand over a minimal set of goal data, usually the desired `Methods` characteristics, such as the `Name` and the `Input` and `Output` parameter descriptions, including their constraints. The algorithm used to compose services is simplified below:

```

1 // member
2 composition;

4 compose(Id, Name, Input[], Output[]) {
5 // traversing through registry
6 while(registry.hasNext()) {
7     Service service = registry.next();
8     Methods methods = service.extractMethods();
9     while(methods.hasNext()) {
10        Method method = method.next();
11        // exact single match
12        if(method.equals(Id) ||
13           method.equals(Name)) {
14            composition.add(service, method);
15            break;
16        // sequential match
17        } else if(method.extractInput().equals(Input[])) {
18            composition.add(service, method);
19            compose(null, null, method.extractInput(), Output[]);
20        } else if(method.extractOutput().equals(Output[])) {
21            if(!composition.empty) {
22                if(composition.getLastMember().extractOutput().equals
23                   (method.extractInput())) {
24                    composition.add(service, method);
25                    break;
26                }
27            }

```

<sup>19</sup>Quality assurance is elaborated in Section 9.4.

<sup>20</sup>At this point, the *ServiceComposer* does not allow the inclusion of composed services as members in other compositions, which could be another potential enhancement of the prototype.

```
28     }  
29   }  
30 }  
31 }
```

## Non-functional Property Handling During Composition

Revisiting the seven non-functional properties described in Section 4.7, here we will elaborate on state management, transaction handling, and the handling of synchronous and asynchronous communication during composition.

**State Management During Composition** Services in the prototype can be parameterized, and they can also change these parameters during an invocation, in accordance with constraints defined in the contract. State information can be stored transiently or persistently. Within a composition, state information (usually `Input`, `Output` or `Inout` parameters) can be saved in two different places: transiently, in the `CompositionDictionary` during handover of data between member services, and persistently, in the datastore. Transient data is stored only in the dictionary and becomes deleted once a composition is globally committed.

Persistent state operations are usually performed using the CRUDF pattern (`create`, `read`, `update`, `delete`, and `find`) within encapsulated Data Access Objects (DAO). [(SD06)] These operations are usually performed against relational databases, where we used the Hibernate API to deal with high-level objects rather than low-level programming constructs.

**Transaction Handling** To guarantee data consistency during processing, a computing environment should be able to commit and roll back transactional state, including composition processing. In the prototype, service composition invocation and processing is done in the application environment, and for this reason transaction handling must be done remotely from the actual service processing.

Within a composition, transaction handling requires the invocation of each member service in its own transaction, assuming the member is described in its contract as transactional. Also, if it is described as transactional, the member state must be stored for the duration of the composition in order to commit or recover the entire composition in case of failure. The cause of failure must be communicated to the invoking component. The component

must then either compensate the individual member's failure (for example, by restarting), or roll back the complete composition execution.

The task of handling transactions in compositions is initiated by the invoking component, the `CompositionStub`.<sup>21</sup> As in the similar process for atomic services, the `ServiceInvoker` determines the endpoint address after the service lookup and hands over the address to the `ProxyFactory`. In the prototype, all composed contracts point to the same stub object address, the `de.soft.composition.client.CompositionStub`. Once the stub is instantiated, the `ServiceInvoker` passes the invocation parameters and contract information to the stub, and begins the invocation of the composition using `invoke`.

Starting from the invocation, the stub determines whether the composed service is transactive or not by referring to the `Meta` section of the composition contract. If so, the stub starts a global composition transaction in which the state of each member is transiently stored. The component used for this transient storage is the `CompositionDictionary`.

After creating the `CompositionDictionary`, the `CompositionStub` starts the invocation of the first member as a completely new service lifecycle: lookup, invoke, and bind. If the first member is processed, the outcome is saved in the dictionary and handed over to the next member. Taking the output of the caller member as an input, the callee member is invoked and delivers an output which another member uses as an input. Finally, the output of the last member is formatted into the output of the entire composition.

In summary, the `CompositionStub` processes the composition, member by member, as defined in the `Flow` section, and additionally provides non-functional property handling in the composition context. Each member transaction can be committed in its own scope, but will be finally committed only if the composed service execution is successful.

To determine the transactivity of a composition, the `CompositionStub` extracts data from the service contract. The global `MetaProperties` section of the composition contract defines whether a composition is transactive at all. If so, the individual member contracts define whether a given member is transactive. If a composition is transactive, the `CompositionDictionary` will be initiated as transactive. Each subsequent transactive member delivers its output to the `CompositionStub`, which writes its output state into the dictionary.

---

<sup>21</sup>The *CompositionStub* was developed in the context of this project.

More in detail, member contracts contain two entities: `TransactionSupport`, which can be set to either `transactive` or `non-transactive`, and `IsolationLevel`, classified by `read`, `repeatable-read`, `serializable`, `read-uncommitted`, or `read-committed`. Once a composition execution is successfully ended, the stub directs the `ServiceExecution` environment to commit all persistent state changes. In this process, the `CompositionStub` delegates the success information to the JTA API in the `ServiceExecution` environment. Acting as a central coordinator, the JTA API locks the transaction for the final commit. This lock is installed during the composition initialization phase, where the `CompositionDictionary` was created.

**Communication during Composition** The prototype supports both the synchronous and asynchronous communication modes. The mode is extracted by the `ServiceInvoker` and handed over to the instantiated stub. The prototype supports the synchronous communication mode in the basic request/reply pattern. Asynchronous operations are delivered as a simple pass, not expecting any response.

During composition, the `CompositionStub` component first determines the global communication mode from the composition contract. If it is asynchronous, all members will be invoked without considering their communication attributes. If the composed service is marked synchronous, all member communication attributes are analyzed, and the ones marked synchronous will be invoked. The processing of the next member then idles until the previous member is successfully finished. If the processing duration exceeds a configurable time, the composition raises an error, and the state changes are rolled back using transaction handling. Asynchronous members are simply invoked and the composition execution continues immediately with the next member, without waiting for a response.

**Integration of Real Time Communication** A special case in composition is the integration of real time services, an interaction that uses the service operations 310 Signaling and 320 Delivery.

The signaling methods are implemented on the basis of SIP commands. Using Java stubs as endpoint addresses in the service contracts, one contract method implementation maps to one dedicated SIP method. For example, the contract method `doInvite` uses a component named `SIPInviteStub`, which invokes the `ServiceExecution` and is then associated with a facade processed in the `SIPProcessor`. The facade delegates the invocation to the

`SignalingClientManager`, which wraps the underlying `SignalingManager`. An example of this delegation is a `doRegister` method in the stub, where the `SIPProcessor` facade calls the `SignalingClientManager`, which then delegates the request to the `SignalingManager`. If the registration is completed successfully, the facade receives the SIP message SIP 200 OK and responds to the stub with a success response.

Within composition contracts, operations of the communication cluster (`Signaling`, `Deliver`, and `Collaboration`) are not composed along with operations of other clusters, because they require a different programming model. Their event-driven nature separates them from the composition contracts of the other operations, which are more process driven. The responses of these operations are united on the presentation level in the Java-rich client. The client has a widget-like presentation, where a communication widget controls the communication service presentation.

After a communication leg is signaled, the media stream is initiated. The 320 `Delivery` operation is not implemented as a service contract/endpoint implementation pair. Rather, the 320 `Delivery` bypasses the service and application layer and is directed immediately to the client widget (such as a SIP soft phone), which encodes the media and presents the data, delivered as an encoded data stream, to the user.

## 8.5 **ConnectME, Application Division**

---

The application division at ConnectME is responsible for assembling services into applications. In the scenario, Activitylife is an application example of a travel community, able to provide content, communication, and control centered around the creation of itineraries. To keep the base data simple, itineraries are assembled user specifically from flights and activities.

### 8.5.1 Overview

The sample application `Activitylife` acts as a service consumer, looking up services in the `ServiceRegistry`, and invoking and binding services with the endpoint implementation provider component, the `ServiceExecution`. The application runs in its own execution environment, which is another app server based on Apache Tomcat. For the prototype, the functionality is limited to mediation between the client and the service layer. This mediation function includes data validation, assignment of incoming service requests to the service layer, service lookup, negotiation, invocation and binding.

The intent of this section is to demonstrate the use of the framework, in particular of the atomic and composed services from the application point of view.

### 8.5.2 Use Cases

The operations extracted from the SOTF model and used by `Activitylife` are: 210 Content Access, 300 Storage, 310 Signaling, 320 Delivery, 330 Collaboration, 410 UserContro, 420 SessionControl, and 430 MediaControl. Transparently, `Activitylife` also uses the 510 Composition and 600 System capabilities introduced in Section 7.5 and implemented in the MMCCS infrastructure.

The implemented service lifecycle activities are the lookup of services at the `ServiceRegistry`, including matchmaking, and the invocation and binding with the provider component, the `ServiceExecution`. The activities are simplified using a sample user, who searches for itineraries and communicates with other users.

#### Lookup Service

When a user searches for an itinerary, the user's client device invokes the application component. The application, as a mediator, then determines which services must be executed during this request, and the order of execution. In the prototype, this is accomplished using the Spring framework and its MVC architecture approach.<sup>22</sup> In more detail, an `ApplicationController` component receives requests, validates the data and creates a `Command` component based on the incoming parameters. The `Command` component is an-

---

<sup>22</sup>MVC refers to the Model View Controller architecture pattern. [(SD06)]

other POJO data container. Once the **Command** component is constructed, the data is handed over to a **ServiceAction** component, which handles the service execution and output formatting.

To facilitate the implementation, two design decisions were made:

- Each **ServiceAction** invokes exactly one service, and the result of multiple service executions will be assembled ultimately in the rich client.
- The service lookup is accomplished using a static routing table, consisting of key:value pairs, that maps service lookup keys to actual service identifiers. These are then handed over to the **ServiceRegistry** to query a service.

The actual lookup consists of three steps:

1. When the application mediator receives a **SearchItinerary(params)** request from the user client, the **ApplicationController** receives the request, validates it and creates a command object.
2. The command object is passed to a service action, which resolves the path of the service to be looked up and invoked.
3. The **ServiceAction** uses a **ServiceLocator**<sup>23</sup> to do the lookup of service contract data.

At this point, two handling alternatives arise. One alternative is that the action already has a service lease, determined by the **ServiceLocator**. In this case, the **ServiceAction** does not require any further lookup and proceeds directly with the invocation.

In the other case, the **ServiceLocator** determines that the **ServiceAction** has no valid service lease, which might be caused, for instance, by a lease expiration or a new start of service. That requires the **ServiceLocator** to do a full service layer lookup. The **Matchmaker** of the **ServiceRegistry** eventually receives this lookup request and queries the contract information using the matchmaking index. If the service contract for the **ServiceRegistry** path can be found, the **Matchmaker** delivers the full service contract to the

---

<sup>23</sup>The *ServiceLocator* pattern is another Java EE pattern [(SD06)] whose purpose is to hide the complexity of service client creation.

**ServiceLocator**. Additionally (though this is not implemented in the prototype) the **Matchmaker** can deliver service-specific lease information back to the **ServiceLocator**. The **ServiceLocator** then hands the information back to the **ServiceAction**, which hands over the command and contract information to the **ServiceInvoker** component.

The prototype handles the service lease by caching the contract information in memory, i.e. inside the Java Virtual Machine (JVM). Whenever the JVM terminates, the **ServiceLocator** has to look up the contracts again. Once cached, a lookup is no longer necessary.

Figure 8.3 illustrates the service lookup process.

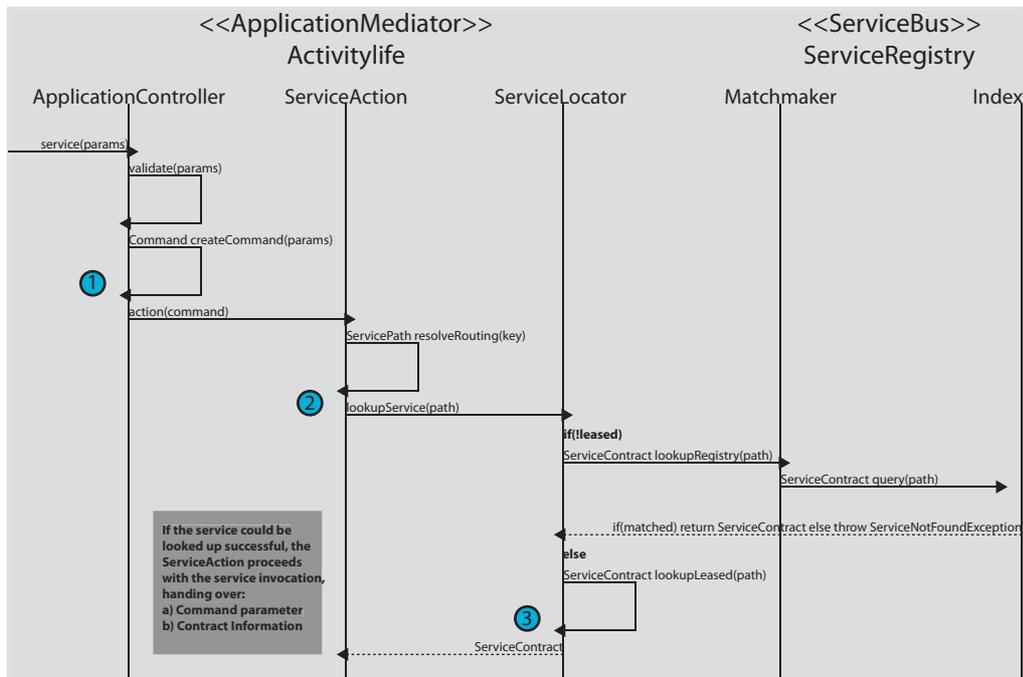


Figure 8.3: Sequence Diagram of Lookup Service Use Case

## Negotiate Service

Service negotiation refers to the process of finding the optimal atomic or composed contract for a given service consumer input. The input consists of parameters and constraints explicitly or implicitly exposed by the consumer. The difference is that explicitly exposed parameters are actively defined by the user (for example, the user enters Mongolia as a location for the `searchItinerary` method), while implicitly exposed parameters are those detected by the software environment. Implicit parameter detection can be seen, for example, in user-context recognition through sensor networks. For example, while triggering `searchItinerary`, the system detects user location information using GPS coordinates.

With this potential for recognizing users' parameters, a service consumer can begin finding the optimal service based on existing service contracts in registries known to that consumer. In the previous section we referred to this behavior as *dynamic service lookup*. Our current prototype dynamizes service lookup with a focus on quality parameters. In other words, if the contractually defined quality metrics cannot be met consistently during a service lease (as analyzed by the `QoS Broker`), the `Matchmaker` tries to find a one-to-one substitute for the service. This process is embedded in the service lookup and can be termed *substitution by quality*.

The limited negotiation capabilities included in the prototype are transparent to the user, and the criteria (the quality metrics) are assumed by the machine. This of course represents only a basic start on service negotiation. The service consumer, ideally, would actively participate in the negotiation, by declining or accepting contracts and methods based on the consumer's parameters, as in a real-world contract negotiation. We see this as an opportunity for a future enhancement of the prototype.

## Invoke Service

Service invocation is generally very complicated, because services are usually deployed in heterogeneous `ServiceExecution` environments, and clients use different technologies for invocation. To reduce this complexity, the prototype focuses only on certain types of invocation. However, the technical architecture of the invocation capability allows the addition of more technology types for the invocation of components. Invocation in the prototype is accomplished by the `ServiceAction` component of the application mediator. Once lookup has occurred, the `ServiceAction` hands over the command and

contract information to the `ServiceInvoker` component (developed for this project).

In general, services can be invoked remotely or locally, depending on the `invokeRemotely` contract flag. In the prototype, all services are invoked remotely. In the `ServiceAction`, the `ServiceInvoker`, after receiving the contract information through the `ServiceLocator`, triggers the invocation in four steps.

1. The `ServiceInvoker` extracts the type of invocation to be done, based on the type of contract (atomic WSDL document or composite XML contract) and on the type of binding to be invoked. The `ServiceInvoker` distinguishes between three types of service invocations:
  - HTTP.POST invocation: Atomic services, contractually described in WSDL documents using HTTP.POST bindings, are invoked by an endpoint address (URI:PORT) defined in the service contract. Input (i.e. `Command` objects) and output data is serialized and deserialized from the HTTP payload.
  - SIP invocation: Atomic services, contractually described in WSDL documents using SIP bindings, are invoked by a service specific proxy. Input/output is stored in SIP header fields. The payload is empty.
  - Composite invocation: Composite services, described in native XML documents, are invoked by a special `CompositionStub` component for all composite services. Input/output and the complete contract are serialized in the HTTP payload.
2. After determining the type of invocation, the `ServiceInvoker` creates an invoking object, a proxy for the invocation. HTTP.POST and Composite invocations use a single unified proxy object for all invocations, while SIP invocations instantiate the SIP method wrapper proxy according to the defined contract binding.
3. The `ServiceInvoker` calls the `extractContract` method of the proxy object.
4. Once contract extraction is successfully completed, `invokeService` is called to proceed with the actual invocation. To guarantee processability, each proxy service implements the interface `Invokable`, which defines both callable methods. If the service is synchronous,

the proxy handles the response including potential errors raised by the `ServiceExecution` environment.

`ServiceExecution` provides an interface/controller pair to be invoked for each type of invocation. The `ServiceControllers` are implemented as Java Servlets. `ServiceExecution` therefore provides an HTTP.REST servlet, a SIP servlet, and a Composite servlet.

Figure 8.4 illustrates the invocation procedure:

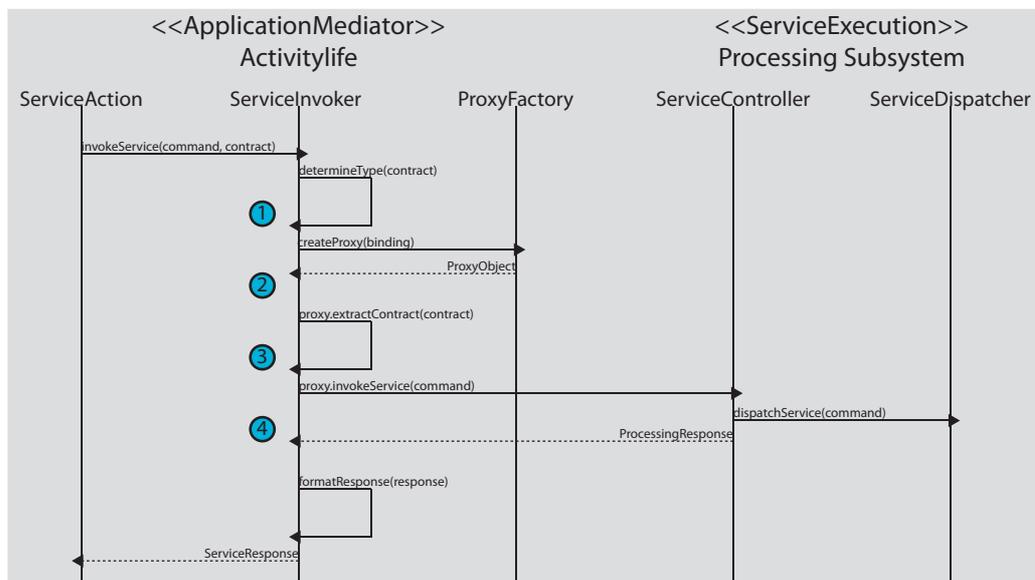


Figure 8.4: Sequence Diagram of Invoke Service Use Case

## Bind Service

The final service lifecycle activity necessary to start using an atomic or composed service is the service binding. Depending on the terms of the lease, an endpoint implementation service is bound to a `ServiceAction` for a specific amount of time. The binding is combined in the prototype with a caching capability, skipping the lookup after the initial bootstrap of the service contract by the `ServiceRegistry`. Once bound, a service can be invoked without looking up the `ServiceRegistry` again. From the perspective of the application (which, in the prototype, is the actual service consumer), the service is tightly coupled until the binding is released. The prototype has shown that it is useful to implement a binding release based on certain system events, such as changes to the service contract in the registry.

## 8.6 Summary

---

Based on the theoretical conceptualization of the telcom framework introduced in Chapter 7, in this chapter we have prototyped the actual use of the framework, implementing the business, data, and technology architecture of SOTF. The chapter described the infrastructure of the MMCCS project in detail, presenting the domain and utility services of the business architecture viewpoint developed in Section 7.5, and using the software components of the technology architecture conceptualized in Section 7.6. Interiorly, the prototype seamlessly mixes services from all clusters, demonstrating the use of the service paradigm in a service lifecycle equally applied to all types of service specifications and implementations.

After presenting this layered architecture, the chapter exemplified, in the prototype, an implementation of the service lifecycle from a service and application perspective. The potential was clearly revealed for applications, as potential mediators with limited computation capabilities, to leverage the reuse and agility of the service layer. In particular, the activity of composition has been demonstrated as an enabler of reuse in the service lifecycle. The prototype has shown that the quality of the semantics used heavily influences matchmaking and composition quality.

In the next chapter we will discuss three key issues and the possibilities of their computation and realization in the prototype: Non-functional service heterogeneity; inter-service, circular, and automatic invocation; and quality awareness.

# Chapter 9

## Prototype Validation and Discussion

*“I thought we might end this evening with a discussion. ”*

*Eisenheim, The Illusionist [2006]*

### 9.1 Synopsis

---

In Chapter 8 we demonstrated the behavior of an application (Activitylife) using a software service-oriented NGN environment, MMCCS. The infrastructure enabled a transparent use of the service lifecycle phases, including, also transparently, two service creation mechanisms, atomic creation and composition. In previous chapters, and in particular in the summary of related work in Section 5.8), we discussed three issues which we will explore more deeply in the context of our prototype demonstration:

- Non-functional heterogeneity of services in the telecommunication domain: Chapter 4 introduced seven non-functional properties of service computation. Chapter 6 assigned these properties to the semantic clusters of the domain and developed a formal cluster notation describing the behavior of the properties in the clusters. Chapter 8 demonstrated the characteristics of these properties during composition.

- Inter-service, circular, and automatic invocation: Chapter 4 introduced the concept of including composed services in another composition and raised the related issue of circular, endless looping. Section 4.7.6 discussed the potential for invoking services automatically, for example during dynamic composition. Chapter 7 developed an operation, `652 invokeAutomatically`, to implement this type of invocation.
- Quality awareness: Chapter 2 introduced quality of service in the network. Chapter 4 extended this discussion with the software-dominated layers of application, service and control. Chapter 6, Section 6.3.1 suggested a general quality model, consisting of the criteria of reactivity, availability, reliability and scalability, for quality assurance.

In the use cases in Chapter 8 we focused on the service lifecycle. In this chapter we will elaborate on the challenges listed above, and discuss the possibilities of their computation and realization in the prototype.

## 9.2 Non-functional Service Heterogeneity

---

Chapter 4 introduced a set of non-functional properties (state management, transaction handling, communication, quality assurance, invocation, security and trust and error handling) which are important to service lifecycle processing, especially to composition. Our framework and prototype development focused on the first five of these properties. Section 6.3 introduced a domain ontology that divided the domain into four major service clusters: community, content, communication and control. The semantic analysis, and the prototype implementation in Chapter 8, showed that the properties behave differently from cluster to cluster and in what ways they behave differently during composition.

In this section, we discuss the effects of these properties as manifested in the prototype implementation. We analyze the limits and constraints of our prototyped solution and propose computation alternatives.

### 9.2.1 State Management

In the prototype, state management for atomic and composed services is initiated by the consumer of the service layer, which in this case is the application mediator `Activitylife`. As described in Section 8.5.2, before looking up a service, the mediator component receives the incoming state, validates the incoming state information syntactically and creates a generic command object. This object contains the state information, which is bound to the extracted data types of the contract during invocation, using Java reflection. In this way, the handling passes the service state from the client to the mediator and then to processing in the service layer.

A major principle of the service paradigm is that services should be implemented statelessly, so as to allow flexibility during processing and composition. However, in the case of our prototype a majority of the services are handled statefully. There are two major reasons for this unorthodox implementation:

- Communication services rely heavily on user identity when connecting identities in pair or group scenarios. Only a valid user identity allows the initialization of the control layer. In the prototype we used a SIP URI for user identity so that a user session can be created in the `UserController`. This initiation information is passed to the service layer as service state information.
- Community and content services generally deal with persistent resources which must be personalized. Persistent resources integrate CRUDF operations to storage capabilities, and these must be passed to the service layer as service state. Furthermore, like communication services, personalized content requires user identity information, which must be handed over as service state as well.

In both cases, services must maintain conversational state. To this end, the universal `ServiceInvoker` component of the application mediator serializes the input state information in the service request, while the service implementations in the `ServiceExecution` accept and process the serialized input. This process becomes a bottleneck in service processing for two reasons: the input state information is serialized in the service layer request using the slow Java reflection mechanism, and the Spring framework dependency injection of the `ServiceExecution` environment uses the same reflection mechanism to connect the service input and the service facade. In addition, the dependency

injection binds validation classes, and the facade uses an XML configuration whose initial lookup also turns out to be performance-intensive. This effect interferes minimally with atomic service processing, but is significant in the case of composite processing, as each member invocation includes a state serialization and deserialization.

In order to avoid such data-binding effects, a future service layer implementation may contain a composition interface in the `ServiceExecution` to process compositions, and their state handover processes, entirely in the service layer. In this way a `CompositionController` component could accept complete composition descriptions as request input and deliver the composed output appropriately.

A disadvantage of this solution would be that the application mediator would lose control of the composition processing and would have to transfer its composed descriptions to the `ServiceExecution` environment. Another disadvantage is that the provider would have to provide a composition runtime capable of processing various formats of composition descriptions.

In addition, with this solution it would be difficult to integrate services of external providers. However, a provider could integrate a remote execution, though it will always prefer its own local implementation. Moving the composition execution in this way from the mediator to the provider, while still creating the composition at the mediator, needs further investigation, and should be considered as an alternative, depending on the service clusters involved and the location of the hosted member implementations of the composition.

## 9.2.2 Transaction Handling

Another property that operates differently in different clusters is transaction handling. Given the finding that a majority of services require stateful processing, the system must handle some of this state information in a transactive way, at least for the services containing a `Create`, `Update` or `Delete` (CUD) data transformation.

As described in Section 8.4.2, in the prototype transactions are distributed: atomic transactions are initiated and coordinated in the service layer (i.e. in the `ServiceExecution` environment), and composed transactions are initiated in the service layer but coordinated in the `ServiceExecution` component. A disadvantage of this implementation is that transaction compensation handling must be implemented in a distributed manner while rely-

ing on the central transaction coordinator, the JTA API, in the execution environment. In case of member failure in a composition, a transaction compensation must therefore communicate its failure to the execution environment to be compensated, and state changes (i.e. the content of the `CompositionDictionary`) must be transmitted to the execution environment to be synchronized with the transaction manager. In the same way, the distributed nature of composed transactions contains additional potential for failure, such as state loss in the `CompositionDictionary` or unavailability of the network. Since the coordination manager is not aware of these failures, they could cause an inconsistency in the system. This is a typical communication problem of distributed transactions and needs further research.

The prototype demonstrated that, during composition, content services usually read or find data (in the prototype, 75% RF and 25% CUD operations), and therefore can be handled non-transactively per default. When handling CUD behavior, which is often associated with enormous quantities of content data, transaction management can cause an overflow or congestion of the `CompositionDictionary`. Transaction handling of communication services is divided into signaling and delivery transactions. The SIP container, as part of the `ServiceExecution` component, supports signaling transactions and provides replication of the transactive data in case of transaction failure. This replication can be used for atomic services, but in the case of composition all transaction handling must take place in the mediator, requiring a specific transaction compensation mechanism.

This additional transaction mechanism has not been implemented in the prototype because it requires transactive handling of SIP sub-requests in the mediator, which would require a service-specific implementation in the `ServiceInvoker`. Currently, the `ServiceInvoker` is not able to store intermediate service state in the context of the transaction. This deficit could be corrected by an extension of the communication proxy objects, which could store the state of sub-requests in the composition dictionary. Another issue is that real-time communication requires an additional transaction compensation mechanism, one that would be more problematic, as real-time communication processing can involve a restart of the ongoing media transmission. These additional efforts led to the decision, not to apply transaction handling to composed real-time communication. Furthermore, in the prototype transaction handling could not be applied to the delivery of communication services, because they completely bypass the execution environment and transmit data between the `MediaDeliverer` and the client-side user agent.

In summary, data transformation (CUD) of atomic and composed commu-

nity and content services requires a central transaction coordinator instance, which we propose to implement in the `ServiceExecution` environment. During distributed data transformation (e.g. during composition), transactions should store state changes transiently and transmit those changes to the coordination manager for synchronization. The lightweight implementation introduced in Section 8.4.2 was sufficient for the synchronous short-term requests of the prototype, but is not able to handle loosely coupled long-term transactions. General service concepts addressing these issues may evolve from the LRT (Long Running Transactions) coordination protocol as part of BPEL standardization [OAS07], or from the Web Services Transaction Framework (WSTF) and its WS-Transaction specifications. [BS02]

Considering composition patterns, we observe that the mediator implementation reliably compensated transaction failures for the sequential and decisional patterns. Parallelism behavior, however, raised issues of concurrent data transformations, which could cause deadlock situations. These issues merit further investigation.

### 9.2.3 Communication

The communication mode was defined in Section 4.7.7 as the underlying interaction mode between provider and consumer, usually distinguishable into synchronous and asynchronous communication.<sup>1</sup> The prototype implements the request/reply pattern for synchronous communication and the passing pattern for asynchronous communication. These were defined on the method level in the service contract.

In the prototype, the communication mode is determined during contract extraction in the application/mediator layer. When starting invocation, the `ServiceInvoker` uses the communication mode information to invoke either a request/reply or a passing. By default, requests are handled synchronously. All community services are delivered in this way, as the rich client expects an immediate response to render the user interface. Real-time communication (for example, the delivery of media streams) is also highly dependent on synchronous delivery. Through highly abstracted architecture and embedded requests, the signalization of real-time communication almost reaches critical reaction times.<sup>2</sup>

---

<sup>1</sup>Communication mode terminology should not be confused with the *communication service cluster* that is part of the five-cluster ontology in Section 6.3.

<sup>2</sup>Further elaboration on the reaction times of real-time services can be found in Section 9.4.

The prototype analysis suggests that signaling via the SIP protocol, with its strong synchronous communication requirement, should only be used initially as mediator/registry abstraction and thus in a composition, further signalization is better accomplished using a direct connection between the user agent and `ServiceExecution` environment, thus trading off the principle of loose coupling between application and service processing for better synchronous performance.

In general, community and communication services (as described in Section 7.5) require a synchronous communication architecture to fulfill their strong response requirements. For that reason, asynchronous mechanisms such as publish/subscribe over a MOM (e.g. JMS), as synchronous communication protocol for operational services, are not used in the prototype. Asynchronous behavior is used exclusively for genuinely asynchronous services such as email messaging.

## 9.3 Inter-service, Circular, and Automatic Invocation

---

### 9.3.1 Inter-service Invocation and Circularity

Service composition enables inter-service invocation and thus provides a direct way to reuse services. In addition to this positive effect, however, inter-service invocation can cause performance issues and can lead to circularity. [RK03] Circularity, as a special undesirable case of inter-service invocation, describes the state of calling between two services with no logically defined exit or an invalid one, a state that can result in an endless loop. Circularity and endless loops can arise whether the composition is created manually or dynamically.

In the prototype, the `ServiceInvoker` serves as a central hub managing all atomic and composed invocations. Inter-service communication in composition scenarios were tested on community, content and communication services and combinations of those. Control layer services (such as the initialization of a user session) are realized via the `ServiceExecution` environment and are not called from the mediator/application directly. Those invocations are initiated automatically by the `ServiceExecution` environment, according to the incoming state information of the service request, such as user or signal-

izing information. For this reason, control layer services are not implemented as contract/implementation pairs, but rather serve as utilities for the other service clusters.

The proper handling of invocations in a composition scenario, in terms of inter-service invocation and circularity, can be verified during composition contract creation by the `ServiceComposer` and measured during the actual processing by the `ServiceInvoker`. Composing rules handed over to the `ServiceComposer` help to avoid circularity at the time of composition creation. The measurement of quality metrics by the `ServiceInvoker` facilitates the avoidance of quality deficits caused by inter-service invocation.

Consideration of composition rules and quality metrics led to a number of research findings, which are described below.

### Inter-service Invocation Control Findings

- Community services, as the most abstracted and composition-adaptable service cluster, have the lowest potential composition depth.<sup>3</sup> Tests of composition depth have shown that composed services that include transaction handling, such as 110 `UserHandling` and 130 `UserCommunication`, are particularly vulnerable to quality-decreasing behavior with increasing composition depth, as they require control layer communication. Each additional inter-service invocation causes a directly proportional (100%) increase in the reactivity of the member service. In contrast, compositions in the 120 `ContentIntegration` operation, in particular its non-transactive services, show a much smaller increase in reactivity: between 40% and 70% for each member candidate per inter-service invocation.
- The reactivity of 210 `Access` operations of the content cluster has proven less than proportional to increases in the composition depth, when composed with other 210 `Access` services. When combined with operations of 220 `Authoring` and 240 `Publishing`, however, their transactive behavior increases the effect of the composition depth and debases service reactivity.
- Communication services of the 310 `Signaling` operation are limited in our implementation to a single composition depth. This decision was

---

<sup>3</sup>The composition depth is the number of invocations of composed services for each member service.

made because the handling of the sub-requests caused by the SIP message flow create a complexity in the `ServiceInvoker` that cannot be handled using the general service invocation workflow.<sup>4</sup> Because communication services naturally include nested sub-requests, they would require sub-request handling in the member invocation, which would make the member dependent on the composition, thus violating the principle of composition transparency for a service invocation and requiring specific logic in the client-side proxy component. 320 Delivery operations cannot be tested against increasing composition depth because they completely bypass the `ServiceInvoker` component.

### Circularity Avoidance Findings

- During the prototype implementation, two types of circularity occurred: contract-based circularity (such as the case of a loop pattern where the termination constraint has been defined invalid) and implementation-based circularity (for example, when service implementations call other service implementations). Contract-based circularity can be checked during composition creation by the `ServiceComposer` before a service becomes published in the service registry. Processing circularity, which occurs more frequently, cannot be preemptively avoided and, to improve service quality, should be traced. A potential solution for this tracing might be a processing feedback engine that reports frequent service failures to the `ServiceRegistry` to trigger (possibly even automatically) a recomposition of the service.
- In general, to handle situations where circularity cannot be prevented, the `ServiceInvoker` should be configured with a timeout threshold for member processing which, when violated, will terminate and roll back the composition processing. This threshold should be included as a quality metric in the service contract, to be extracted by the invoking proxy object.

As indicated by these findings, the potential composition depth appears to be highly dependent on the semantics of a service and on the size of the logic included in the service. Heavyweight services that consume a significant quantity of computing resources (e.g. memory and CPU) should have a smaller composition depth than lightweight services.

---

<sup>4</sup>This workflow was introduced in Section 8.5.2.

Other factors that can affect the maximum suitable composition depth of a service are the importance of the service in a composition<sup>5</sup> and its scalability. An individual service becomes embedded in compositions and might embed other composed services. A composed service thus acts as a sort of node in a network of composition consumptions. The efficiency of this network must be maximized to produce a robust composition.

### 9.3.2 Automatic Invocation

Automatic composition was described in Section 4.7.6 as the fully dynamic creation of service compositions and the automatic invocation of all member services. Section 7.4.2 put that composition goal in the perspective of this thesis. The SOTF framework and the prototype propose an infrastructure that will evolve into dynamic composition creation and fully automatic invocation. During the framework and prototype development, various research findings were identified, as described in the following sections.

#### Cluster-Specific Findings for Automatic Invocation

The composition-creating entity (in the prototype, the `ServiceComposer`) must have complete information about all composition candidates. As outlined in the basic data model of a service contract in Section 7.6, the `Types` and `Methods` information, describing functionality and parameterization, is not sufficient for automatic composition creation. The previous section's discussion of the heterogeneity of the non-functional properties showed that the composability and addressability of a service as a member in a composition depends to a great extent on the quality of the description of its non-functional properties (both implicit and explicit). The capability to define such properties in service description standards will thus strongly influence the further development of automated service composition.

The prototype analysis showed that each cluster has its own requirements for the degree of semantic information necessary for automatic composition to succeed.

- Community services require the largest amount of semantic information to be automatically composed. On the one hand, goal information

---

<sup>5</sup>Consideration of the importance of a service in a composition is discussed in Section 6.4.1.

(such as user context information, functionality description, parameterization and constraints) must be complete, and on the other hand, the available content and communication member services must have fully described contract entities compliant with the SOTF data architecture, including, especially, their state, transaction, quality and communication information.

- Content services require input information as complete as that needed by community services, but the member service descriptions do not need to be as complete. Individual descriptions can be derived from the general specifics of the content cluster or from the incoming goal information: State information can be extracted from the service parameterization, the major read-find operations can be assumed to be non-transactive, the general quality model can be applied as described in Section 6.4.2 and the communication mode can be assumed synchronous by default.
- As a consequence of their tight coupling with the service client, communication services, to be automatically embedded in a service composition, require, at a minimum, user information (such as identity) and client information (such as device information). Real-time services, because of the complex nature of their invocation structure, require, in addition, a full description of their non-functional properties and a specific synchronous invocation handling. This synchronous invocation handling was not developed in the prototype.
- Control services, as the underlying utility layer, are fully controlled by the `ServiceExecution`. Their invocation was fully automated in the prototype, and they were loosely coupled with the `ServiceExecution` by means of dependency injection.<sup>6</sup> The separation of the control layer from the service contracts published in the `ServiceRegistry` completely hides the control layer from the service layer consumers, and in this way greatly reduces the complexity of service invocation.

---

<sup>6</sup>Except for the *MediaSignalizer*, each component of the control layer was decoupled from the *ServiceExecution*.

### General Design Rules for Automatic Invocation

In addition to our cluster-specific guidelines for automatic invocation, we can derive some general design rules to be considered if an automatic invocation implementation is planned.

- Seen from the service consumer's perspective, automatic and manual invocation in the service lifecycle should be handled in the same way, though referring to different endpoint addresses. The prototype suggests a generic `CompositionStub` component as an endpoint address.
- As all compositions can be processed using a unified description scheme (in the prototype, the SOTF data architecture), the invocation of all composed services can be encapsulated in a single component (in the prototype, the `CompositionStub`) to centralize control of the composition.
- The `ServiceExecution` environment does not need to be aware of the type of invocation (manual, semi-automatic, or automatic) or the type of service (atomic or composed).

## 9.4 Quality Awareness

---

Section 2.4 identified the awareness and assurance of quality metrics as one of the most important concepts in NGNs. Section 6.3.1 proposed a general quality model to measure the primary quality criteria of reactivity, availability, reliability and scalability.

In this section we use the quality model to measure the quality of service lifecycle activities, with an emphasis on the quality of manually and dynamically created service composition as these occur in the prototype implementation. The tests are made on a cluster of Linux and Windows XP-operated machines connected by a regular network. The components are deployed on several machines, while `ServiceRegistry` and `ServiceExecution` are running on a single machine each. The MMCCS control layer components, `UserController`, `MediaSignalizer`, and `MediaDeliverer`, are also installed on separate machines.

## Reactivity of Service Lifecycle Activities

The service lifecycle, from the application usage perspective, contains five major activities: lookup (including matchmaking), negotiation, invocation, processing and binding. Before entering into usage of the application, services must be specified, registered and published. Usable services can be either atomic or manually or dynamically composed.

For our current analysis we propose two services: `communicateToUser` and `searchItinerary`. Both services can be implemented with either atomic or composed contracts. In addition, the service descriptions composed by `searchItinerary` can be dynamically created by sequentially invoking the service methods `searchFlights`, `searchActivities`, and `createRoutes`. While the search methods provide the actual data search, the route creation generates potential traveling paths.

In our analysis of service lifecycle quality, we focus on the reactivity of the lookup, invocation and processing activities, because they are the ones that consume the most computing power. We have broken the activities down, for this analysis, into their computing steps, as introduced in the use cases:

1. Service lookup: `createCommand`, `resolveRouting` and `lookupService`
2. Service invocation: `determineType`, `createProxy` and `extractContract`
3. Service processing: `dispatchService`, `processService` and `formatResponse`

To distinguish among service specification types, we tested the activities against five types of registered and published service contract-implementation combinations:

- `searchItinerary` as an atomic service
- `searchItinerary` as a manual composed service consisting of the sequenced flow of `searchFlights`, `searchActivities` and `createRoutes`
- `searchItinerary` as a dynamically composed service with the input parameter of service names, creating a service composition on the fly and invoking the services automatically
- `communicateToUser` as an atomic service

- `communicateToUser` as a manually composed service using several SIP method invocations

The services are specified according to the procedure described in Section 8.4.2 (atomic services) and Section 8.4.2 (composed services). While the `searchItinerary` service behaves as a service of the content cluster, `communicateToUser` is a service of the communication cluster. These two services were chosen because of their dissimilar natures: Content operations are handled in the prototype sequentially (via `I:HTTP:REST`), while communication operations are event-driven (via `I:SIP`).<sup>7</sup> They differ not only in their programming model, but also, widely, in their meta, static and dynamic semantics. Their dissimilarity makes these services particularly useful in the investigation of quality specifics in the service lifecycle. Our testing focused on the reactivity of these services, arithmetically calculated, as described in Section 6.4.1, in milliseconds of time.

**Service Lookup** Service lookup, which is a mediation between the client and service layers, is the first computing activity of an application service consumption. When receiving a `service(params)` request, the service lookup begins with the validation of the incoming parameters and the creation of the command object through the interpretation of the incoming request parameters. While the `searchItinerary` service is requested via the `I:HTTP:REST` interface, the `communicateToUser` is requested using the `I:SIP` interface. The test results, shown in Figure 9.1, show a slightly slow validation/command instantiation of the `searchItinerary` because of the use of the Java reflection mechanism. The validation and command instantiation of the `communicateToUser` is done using the HTTP header variables directly and is therefore faster.<sup>8</sup>

After successful command creation, the `ServiceAction` tries to resolve the routing key of the service. Atomic and manually composed services are directly routed via a static routing table. Test results show that their routing is somewhat faster than the routing of the dynamic composed services, because in their case the `ServiceAction` must build a dynamic path of useful parameters to create a composition that can be executed. This dynamic path is used to create a dynamic composition request for the `ServiceComposer` in the `ServiceRegistry`.

---

<sup>7</sup>The content cluster is described in Section 6.4.2 and the communication cluster in Section 6.4.3.

<sup>8</sup>As described in Section 8.5.2, the SIP protocol transports parameters exclusively in the IP headers.

Next, the actual service lookup begins. The initial service lookup without a valid lease emerged as significantly slower than the lookup for a leased service. The registry lookup contains four steps:

1. Contacting the matchmaker<sup>9</sup>
2. Querying the index
3. Assembling the required contract data
4. Delivering the data

While steps 1, 2, and 4 remained largely constant among the tested services, the initial contract data assembly for composed services was observed to be faster than for atomic services, because of the workaround of describing implicit data in a separate document from the WSDL. After the initial lookup, composed services had to look up each member service, which made their lookup considerably slower than that of the atomic service, with just one lookup. The slowest registry lookup, however, was found when dynamically composing the `searchItinerary`, because of the use of the `ServiceComposer` to produce a composition definition using an algorithm that queries the index.

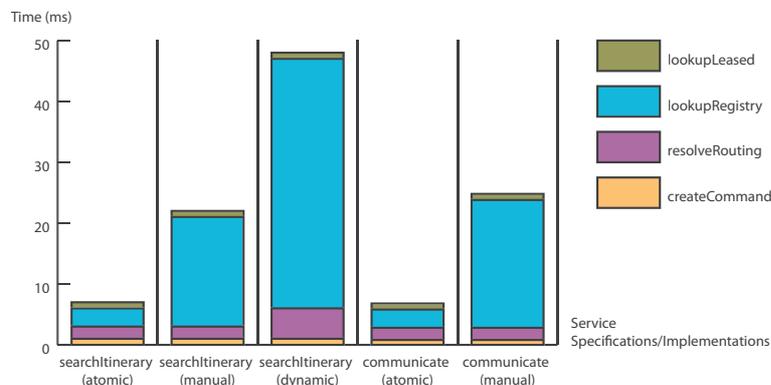


Figure 9.1: Quality of Service Lookup by Service Specification Type

It can be seen that the remote lookup of service contract data, especially during dynamic composition (as the composition must be created at runtime), is the most time-consuming activity in the overall service lifecycle.

<sup>9</sup>Dynamic compositions request the *ServiceComposer* before querying the index via the *Matchmaker*.

Lease and caching mechanisms must be optimized to scale service computation, depending on the semantics of the service. Dynamic composition can be cached based on user input parameters, but frequent context variation makes it almost impossible to define a fixed lease time. Heuristics or other predictive mechanisms could be employed to improve the accuracy and performance of a dynamic composition.

**Service Invocation** Following lookup, invocation begins with the determination of the service type, based on the received service contract data. This determination occurs in a negligibly small amount of time for all services tested. Nevertheless, SIP services can be recognized as somewhat faster, because the request to the `I:SIP` interface identifies the service type according to the interface receiving the service request.

The test results show that proxy creation for composed services is a little faster, as they are invoked by the `CompositionStub`, a unified proxy for all manually and dynamically created compositions. Atomic service proxies, on the other hand, must be extracted from the contract bindings, which is to say the endpoint address. The atomic `searchItinerary` service uses an `HTTP.POST` invocation stub, while the atomic `communicateToUser` stub refers to an individual proxy, which is able to process multiple SIP messages (`INVITE`, `REGISTER`, and `ACK`) within the Java logic.<sup>10</sup> The composed `communicateToUser` contract combines the various SIP message invocations<sup>11</sup> into a composition.

As the next step, the extraction of the contract data consumes more time than the preceding steps, because it involves parsing and analyzing the WSDL and XML contract data, and the extraction of the two documents containing atomic contracts requires more time than the extraction of the individual documents of composed contracts.<sup>12</sup>

Service invocation testing has demonstrated that invocation as a service life-cycle activity is uniformly implemented for atomic and composed service specifications and implementations. The prototype demonstrates similar initial invocation times for all services.

<sup>10</sup>In the example case, all three SIP messages must be invoked for each user to initiate the delivery of the RTP stream.

<sup>11</sup>In the prototype, each SIP message is implemented as a separate, atomic service.

<sup>12</sup>The asterisk labels in Figure 9.2 indicate that the composed implementation tests contain only the initial contract extraction. All other member contract extractions are omitted.

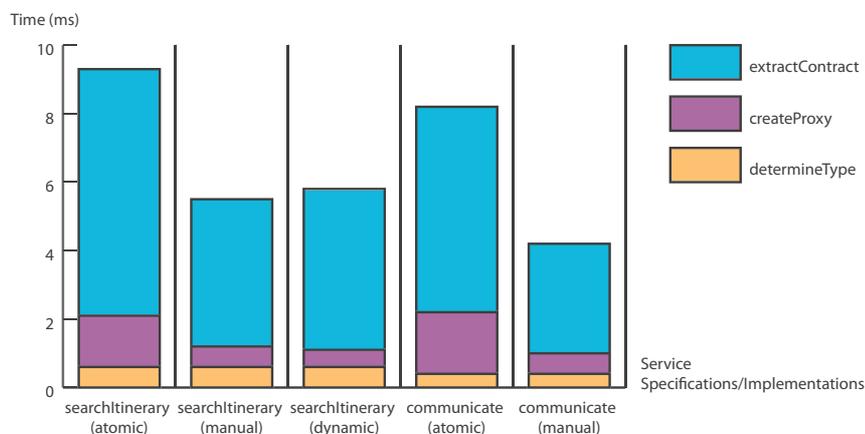


Figure 9.2: Quality of Service Invocation by Service Specification Type

**Service Processing** The final step in application service consumption is the processing of the services, starting with dispatching in the `ServiceExecution` environment. In the test analysis, dispatch times are an aggregate of the times from invocation through `ServiceController` processing and `ServiceProcessor` detection. It can be observed that in this analysis atomic services have a significantly lower dispatch time than composed services because, for atomic services, dispatching is done only once. Composite dispatching, on the other hand, involves multiple dispatches, exactly one per member.<sup>13</sup>

The actual processing of the service in its `ServiceProcessor` is significantly faster for atomic services, because their processing is done in a single implementation facade object. Composite services process multiple service implementations and hand over the results of the processing to the application. In addition, the processing of the `communicateToUser` service involves multiple remote and synchronous sub-requests to establish connectivity between users. For example, the service requests involve multiple responses among participants such as `100 Trying` or `180 Ringing`. Another reason for the high consumption of time is that the sub-requests are invoked remotely to connect two SIP-capable end-user devices. This remote invocation includes the lookup of IP addresses and waiting for the sub-requests to be processed successfully so as to proceed with the service execution.

Finally, the tests compare the time consumed by the formatting of the re-

<sup>13</sup>Measurement of dispatch times for composed services is limited to pure dispatch time, meaning the time required for invocation, controller processing and processor detection. Waiting times resulting from member idling, for example, are not considered.

sponse. The `searchItinerary` service delivers its response as HTTP.POST to the service consumer. The pure formatting time for all services is similar.<sup>14</sup>

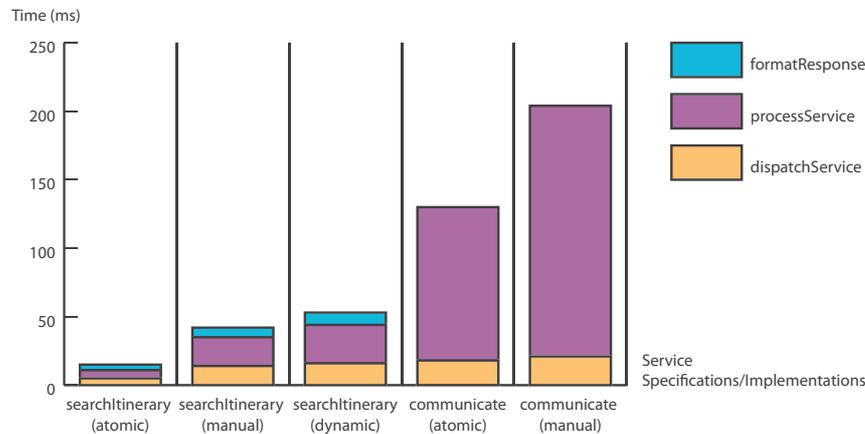


Figure 9.3: Quality of Service Processing by Service Specification Type

## 9.5 Prototype Enhancements

The prototype implementation introduced in Chapter 8 and evaluated in this chapter demonstrates the service lifecycle in the telecommunication domain, based on SOTF framework structures. The implementation, consisting of nearly 4,500 source files developed over almost 750 person days, proves the complexity of telecommunication service computing.

As an experimental system, the prototype still contains limitations to be resolved and the potential for future improvements. Some of these are itemized in the list, below, of optional future enhancements, arranged according to the software components implemented:

- **ServiceDesign:**
  - Development of a visual interface to facilitate manual composition capabilities

<sup>14</sup>The formatting time for the *communicateToUser* service was not measured, because it is part of the processing in which the service delivers the stream directly to the service consumer, bypassing the application.

- Extension of import capabilities to use ontologies in a composition context
- Definition of a unified service description format that can be used in the definition of atomic and composed services
- **ServiceRegistry:**
  - Development of a composition rule engine to define the mappings between consumer goal parameters and service contract data, to facilitate the dynamic creation of composition contracts
  - Extension of the **Matchmaking** engine to enable extended, complex queries (e.g. intersect or grouping) and semantic reasoning
  - Implementation of automatic and active service consumer negotiation capabilities
  - Extension of composition behavior to enable parallel and loop behavior in the composition flow definition
  - Enabling the reuse of composed services in other compositions
- **ServiceExecution:**
  - Monitoring of the quality metrics of service execution, followed by a more detailed analysis of the developed quality model
  - Dynamic improvement of service usage by, for example, automatic triggering of renegotiation in the case of potential improvement of SLAs between consumer and provider, or self-improvement of a service composition through the replacement of low-quality members
  - Decoupling of the **MediaSignalizer** from the **ServiceExecution** instance
  - Extension of security and error mechanisms, such as the implementation of service-specific, configurable mechanisms

## 9.6 Summary

---

The discussion at the beginning of this chapter established that the service clusters behave differently with regard to their non-functional properties, and that the ability to compose services (especially for automatic composition) is dependent on the quality of the descriptions in the atomic service contracts. We have determined that, among the service clusters, community services require the highest degree of input goal information and semantic contract description when they are trying to assemble content and communication services. The complexity of control services, however, can be invoked in a fully automatic manner by the `ServiceExecution` environment and thus completely hidden from the service layer consumer.

The quality and performance investigation carried out in this chapter illuminated the significant differences between the communication service cluster and other service clusters. It demonstrated that, although communication services can participate equally in the overall service lifecycle, they exhibit a completely different behavior from the other services during composition.

Focusing on service lifecycle activities, we have demonstrated that the performance of a service composition depends largely on efficient communication between service broker and service consumer. Once the consumer has leased a service, except in the case of invocation, a composed service and an atomic service behave similarly.

Our research exposed the invocation of services as the most complex lifecycle step, as the `ServiceInvoker` must be aware of composition transparency while additionally handling multiple technologies and formats during contract information extraction. It showed that, because the `ServiceInvoker` must be abstracted to handle all cases, the performance of invocation slows dramatically during composition processing, especially when handling communication services with multiple initialization requests among the client, application and service layers.

In Chapter 10 we will discuss what has been achieved in this work and consider the potential for future research and a future outlook.

Part V

**Conclusion**



# Chapter 10

## Contributions, Future Research and Outlook

*“It is not the end. It is not the beginning of  
the end. It is the end of the beginning. ”*  
*Millennium, seen from the authors  
perspective [1989]*

### 10.1 Synopsis

---

In our concluding chapter we will measure our thesis results against the research questions and objectives introduced in Chapter 1. We will elaborate on the contributions we intended to make and propose potential directions for future research. We will propose a future perspective concerning the complementarity of the underlying research concepts, Next Generation Networks, Service-Oriented Architecture (SOA) and service composition.

---

## 10.2 Answers to Research Questions

---

The central question posed in Chapter 1 was, "How can the software service paradigm supplement the emerging IP-based telecommunication infrastructures to improve agility through reuse?" We approached this question on several analytical levels, beginning with a conceptual deep dive into the complementarities between software services and IP-based infrastructures, followed by a software framework perspective to establish a sustainable architecture, through to an implementation perspective that validated and demonstrated the developed framework.

We have demonstrated that semantics-driven software service engineering (i.e. service composition) can deliver the agility and openness required by today's telecommunication infrastructures. The thesis clarified that service composition, a highly abstracted computing mechanism based on service reuse, should be considered an integral part of a service lifecycle facilitated by a loosely coupled software-component architecture.

In detail, the thesis proposed a uniform service lifecycle, computed in a four-layered software component architecture, constructed using the characteristics of the Next Generation Network and the principles of the software service paradigm. In the course of our research, we answered the questions asked in Chapter 1, Section 1.4 as summarized in the following paragraphs.

### **Conceptual Complementarity of Telecom Networks and the Software Service Paradigm**

NGN is an umbrella term for future IP-based telecommunication architectures. NGN's inherent service character and its decoupling of operation and control are ideally formed for accessibility via a software layer with open interfaces and decoupled components. Implemented between the application and network control layers, this *service layer* is conceptualized here as the reusable, operational software backbone of the network.

In the service layer, services are provided in a lifecycle where applications act as service consumers, *looking up* (matchmaking), *negotiating* and *binding* NGN software services from a broker in the service layer. Before the usage starts, a provider *specifies* and *registers* services to be *published* from a broker instance. Using a high degree of abstraction, service composition enables the provider to reuse its atomic services as another way to do service

specification, in addition to the common contract/endpoint implementation development. In this way composition is integrated, seamlessly for the service consumer, between specification and registration, as a service provider's lifecycle activity.

## Telecommunication Service Framework Development

Our answer to the framework development questions posed in Chapter 1 is embodied in a Service-Oriented Telecommunication Framework (SOTF), which we summarize here in three sections that correspond to the business (top), data (middle), and technology (base) perspectives.

**Business Perspective** To achieve a minimal semantic commitment of the service layer for application consumption, we have proposed four mandatory clusters of operational services: *community*, *content*, *communication* and *control*. Communities are user applications that invoke content, communication, and control services.

In their semantic specifications, the clusters have proven their heterogeneity by exhibiting different characteristics. The main differences have been observed in their non-functional definitions, particularly in quality (including reactivity, availability, reliability, and scalability), state, transaction, communication and invocation. In addition, the thesis suggested cohesion as a new criterion for assessing dependence between clusters. The control cluster has been identified as the most cohesive service aggregation and the most quality-dependent, but these services are also very composable. The clusters can be refined into collections of operations and further into business methods. The modeled outcome of this refinement is the business architecture.

**Data Perspective** SOTF introduced a data architecture, a common data model for the definition of services, as one of the key facilitators of a unified service lifecycle. The data architecture is valid for both composed and atomic services, and this commonality fosters the extended semantic expressibility that has been identified as one of the main challenges for today's service processing. We proposed a model based on service attributes and parameters that partitions an NGN service into the entities **Meta**, **Types**, **Properties** and **Methods**. Through this normalization of their definitions, services became uniformly matchable, negotiable and composable.

**Technology Perspective** At its base level, SOTF contains a technology architecture that includes a four-tier (domain, utility, control and persistence) component specification. Directly interfacing with the service consumer, `ServiceDesign`, `ServiceBus` and `ServiceExecution` were defined in depth, including their interfaces and sub-components. The central concept of a `ServiceBus` was worked out in the thesis by implementing components for matchmaking (component: `Matchmaker`), quality assurance (component: `QoSBroker`) and service composition (component: `ServiceComposer`).

### **Implementation of Service Composition in the Telecommunication Domain**

Our prototype development showed that, to harness the power of service composition, it is essential to analyze the domain and the software environment comprehensively. In our analysis of related work in Chapter 5, app servers and ESBs were identified as appropriate runtime environments, the Java EE language was proposed for the implementation of service logic, and the XML family, especially the web service framework stack, for the implementation of service contracts.

On the protocol level, the prototype implemented component interfacing using HTTP for non-real-time services, while real-time services used SIP/RTP messaging for signaling and delivery of media streams. For communication processing we adopted the IMS approach, using a SIP application server, acting as a back-to-back user agent, to signalize SIP messages.

A `ServiceBus` was implemented in the prototype to demonstrate the defined semantics, compose services in accordance with those semantics and assure correctness and quality sufficiency during composition. The semantics developed in this work enabled the `ServiceComposer` to interpret diverse service contracts and eventually match appropriate member candidates for either a manual or a dynamic composition request. To manage services universally, we proposed extracting the contract information into a service index, which would enable a common matchable data structure.

The `ServiceNormalizer` demonstrated the integration of diverse contract formats. Communication services were abstracted using WSDL contracts, mapping a single SIP message into an atomic contract. Since SIP signaling requests usually require multiple SIP sub-requests, various SIP stub components were developed to enable the integration of communication services in the service lifecycle.

Responding to the identified gaps of current service processing in the telecommunication domain, the prototype enabled the dynamic composition and automatic invocation of composed services in the lifecycle. In the prototype, dynamic composition used an algorithm to interpret incoming user parameters and create a composition goal service. This algorithm processing was seamlessly integrated into the lifecycle using parameter recognition. Based on implementational methods, appropriate member candidates were selected and a composition description generated.

## 10.3 Thesis Contributions

---

From the author's perspective, the most significant achievements of this work are its semantic domain conceptualization, the SOTF framework and the prototype implementation. These contributions are summarized in this section.

### **NGN Service Ontology**

A semantic interpretation of the domain is a prerequisite for service processing, especially composition. We have developed a telecommunication ontology to serve as a model for structuring the descriptive entities of an NGN service. Employing a meta syntax, four service clusters (community, content, communication, and control) aggregate homogeneous service capabilities, capturing the necessary functionality of a service layer. Structured, unified semantics can be mapped onto data entities, thus improving the lookup and negotiation capabilities of the service consumer and enabling the automation of composition.

Such semantics also facilitate the integration of services within and between business organizations. They demonstrate the level of cohesion between services, which can be used to define the services degree of composability. Our semantic conceptualization confirmed the importance of the definition of non-functional properties as main indicators of composability in a composition context. We have shown that, especially in the telecommunication domain, the definition of service-specific (or service cluster-specific) quality metrics constitutes a key parameter for efficient service processing.

## **Service-Oriented Telecommunication Framework**

We proposed a Service-Oriented Telecommunication Framework (SOTF), a telecommunication software framework that models the service layer of an IP-based network from the business, data, and technology perspective. SOTF brings together the structural characteristics of the NGN concept with the principles of the software service paradigm in a composable service architecture.

**SOTF Business Architecture** Working from our previous semantic analysis, SOTF presents a coarse-grained business architecture using operations and conceptual service methods as architectural building blocks. The model contributes a computable service map of the telecommunication domain, including the necessary non-functional and composition properties. Further, it suggests a unified nomenclature, which is useful in tracing and analyzing services and service transformations between the business and technology views of the architecture.

**SOTF Data Architecture** Based on the business architecture, SOTF offers a data model to establish a valid, domain-wide service data structure. The structure as developed, embedding the ontological entities of the semantic analysis, is appropriate for both atomic and composed services. The model is intended to serve as data template for the implementation of service contracts, facilitating sufficient service semantics to encourage accurate service matchmaking, which is necessary for efficient atomic service processing and composition.

**SOTF Technology Architecture** Computing business and data architecture, the SOTF technology model introduces a software component architecture that is loosely coupled through the use of open interfaces. The model specifies the components and their sub-components in detail according to their implementing software tiers, their underlying design principles, the tasks they perform and their outbound interfaces. These components, whose development was based on the specifics of the telecommunication domain, provide a foundation for future implementations of the service layer.

### Service Computation Prototype

Through the extensive prototyping carried out for this project, we established the continuous implementability of the service paradigm across all perspectives, from business to technical, of the telecommunication domain. Unlike any previous framework projects, this prototype exemplifies a generic service lifecycle implementation from the ontology model to the technology architecture. It embodies a way of thinking that validates the universal applicability of the service paradigm, which is one of the primary advantages the paradigm holds over its predecessors, objects and components.

State management, transaction handling, communication, quality assurance and invocation were tested against the four major service clusters, and the results showed that efficient composition depends heavily on the semantic structures (clusters, in this case) to be assembled. While community services, to be used in a composition, required fully maintained member contract information, other clusters' information were derivable from cluster-wide descriptions. Control cluster operations were successfully hidden from the service consumer and invoked in a fully automatic way by the `ServiceExecution` environment. These findings show clearly that the manual and, especially, automatic composability of a service depends on the quality of the contractual semantics and the existence of domain-specific ontologies.

Our performance evaluation revealed that the service usage-initiating activities (such as lookup, matchmaking and invocation) between consumer and broker are the Achilles heel of the service lifecycle. We demonstrated that, especially in a real-time service computation, the complexity and reactivity of the invocation in a composition can jeopardize quality.

## 10.4 Suggestions for Future Research

---

The applicability of service composition in telecommunication infrastructures can be extended into all three of the areas on which we have focused: further refinement of the semantic conceptualization of the domain, functional extension of the service-oriented telecommunication framework (SOTF) and enhancement of the prototype implementation.

The definition of telecommunication semantics in the form of ontologies is a heavily discussed topic in current research. [HMT07] [PC07] As demonstrated in the thesis, ontologies can act as both a domain model and a foun-

dational component for a repository of reusable artifacts (services). [PGM04] As a domain model, the telecommunication ontology proposed in Chapter 6 can be extended in several ways. The telecom clusters could be complemented by a business service cluster to integrate the business operations (BSS) carried out in the domain.

Another way of refining the ontology would be an extension of non-functional property descriptions, especially those of quality and invocation. More predefined quality metrics could be used to improve service processing, especially for real-timeliness of communication services. The description of the invocation of services could be refined to achieve full independence between contract and invoking environment.

A refinement of non-functional properties could lead to a better prediction of composability between member services under a known composition goal. This composition goal could be described using a composition goal language, which would semantically describe the goals to be achieved, relationships within goals and constraints to be considered. [ML05] Composability between two services can be calculated in an algorithm, comparing non-functional properties and runtime quality metrics under the composition goal description.

Service software framework research can be extended within a general or telecom-specific scope. Generally, the service lifecycle-initiating activities of lookup, matchmaking, and invocation contain multiple further research opportunities. Lookup and matchmaking could be complemented by a generic service repository data model as a basis for the interoperability of service repositories. Additional research might target a *service query language*, a common idiom for requesting services from a repository, analogous to the Structured Query Language (SQL) for databases.

For the integration of existing services, the `ContractNormalizer` concept could be expanded for heterogeneous data and service models. As a functional extension of the `ContractNormalizer`, a rule engine could be developed to provide mappings of heterogeneous formats into a general repository data model.

The prototype validation in Section 9.5 introduced a number of potential enhancements. The seamless integration of ontologies in the `ServiceRegistry`, and the monitoring and runtime improvement of composition quality, are two significant tasks for future research. OWL-S ontology definitions offer the potential to split the current `Matchmaker` component into separate artifacts: a `QueryHandler`, a `QueryExecutor`, and a `QueryReasoner`, using SPARQL as

a query intermediation format. [AKC05] [EP07]

In addition, the runtime improvement of existing compositions could be enhanced with a feedback mechanism in which the `ServiceExecution` environment would report the quality of processed compositions to the `ServiceRegistry`, which would then use this data as input for the `ServiceComposer`, to improve the quality of future compositions.

## 10.5 Outlook

---

Domain-oriented service computation will improve and facilitate the openness, interoperability, and flexibility of today's telecommunication infrastructures. The mindset of the service paradigm and its use, as manifested in the SOA architectural style, will not only reshape existing organizations but will extend its influence to the entire market.

Inevitably, in the future, telecom infrastructure will no longer be just a network of hardware nodes. The probability is that it will evolve into a ubiquitous environment, openly accessible via a service layer and software interfaces provided by multiple market players. The mindset of software services will be more firmly established as the lingua franca of telecommunication transactions: from the economic perspective as billing and sales units, from the operational perspective as a control and maintenance target, and from the technical perspective as the building blocks of the architecture.

The role of service composition in future telecom infrastructures will depend on the quality of service descriptions available. It is unclear whether service composition will be used as a programming instrument within an operator's network or whether composition capabilities can evolve to be used by application providers to assemble the services of multiple registries, providers, and technologies. The profusion of legacy infrastructures and the diversity and enormous performance requirements of telecom services make the carrier grade service composition still a case for extensive research. As part of that effort, this thesis may facilitate the organization of service-oriented thought in the domain, by disclosing gaps in existing approaches and by putting the focus on a central future challenge: the creation of a programmable (composable), ubiquitous, loosely coupled software layer over existing legacy infrastructures, transforming the network into a programming environment.



# Part VI

## Appendices



# Appendix A

## A. Appendix of Theoretical Background

### A.1 BNF Notation

---

The Bacchus Naur Form (BNF) and the extended BNF (eBNF) are used to formalize on an abstract level the clauses in a composition solution. In accordance with the ISO standard 14977 (ISO-14977) the following symbols are used:

- $\dots$  – Non-terminal symbols
- $'\dots'$  – Terminal Symbol
- $''\dots''$  – Default Terminal Symbols
- $(\dots)$  – Dependencies
- $[\dots]$  – Optional Symbols
- $\{\dots\}$  – Symbols, repeated zero or more times
- $\{\dots\}^-$  – Symbols, repeated one or more times
- $:=$  – Definition

- ; – Rule Terminator
- I – Alternative
- , – Concatenation
- – – Except
- (\*...\*) – Comment
- ?...? – Special Sequence

# Appendix B

## B. Appendix of SOTF Framework Conceptualization

### B.1 Specification of Application Architecture Components

---

The following list provides a more detailed specification of the technology architecture introduced in Section 7.7. The specification introduces each main component modeled in the UML component diagram in Figure 7.7 and each subcomponent in the architecture, which consists of four layers: domain, utility, control and persistence.<sup>1</sup>

- **ServiceDesign**
  - **OntologyInterpreter** (Domain, ServiceDesign, API), Semantic Conceptualization, *Transformation of ontologies into service contract and implementations*
  - **ContractDesigner** (Domain, ServiceDesign, API), Composition Visualization

---

<sup>1</sup>As in the coarse specification developed in Section 7.7, the components are detailed by these criteria: **Component** (Layer, Parent Component, Component Type), Design Principle or Conceptual Requirement, *Tasks*.

- **ContractImpexer** (Domain, ServiceDesign, API), Contract Heterogeneity, *Import and export of contracts from and into certain other environments; transformation of standard or vendor-specific properties into formats supported by the targeted broker*
- **ServiceExecution - Processing Subsystem**
  - **ServiceController** (Domain, ServiceExecution, API), MVC Pattern (J2EE Pattern), Chain of Responsibility Pattern (GoF Pattern), Command Pattern (GoF Pattern), *Separation of invocation of diverse service implementations, depending on service type (domain or utility), cluster (e.g. content or community), requirements (e.g. real or non-real time) or implementation technology (e.g. HTTP, SIP)*
  - **ServiceDispatcher** (Domain, ServiceExecution, API), Business Delegate (J2EE Pattern), *Assignment of service implementation; handover of parameters; checking of invocation correctness*
  - **ServiceProcessor** (Domain, ServiceExecution, API), Service Lifecycle (SOA), *Processing of service implementations according to their contractually defined constraints; delegating of responsibility to runtime (e.g. failure compensation); logging of processing events*
- **ServiceExecution - Runtime Subsystem**
  - **StateManager** (Utility, ServiceExecution, API), State Management (Non-functional Requirement), *Service state management in cooperation with the service processor*
  - **TransactionManager** (Utility, ServiceExecution, API), Transaction Handling (Non-functional Requirement), *Service transaction management in cooperation with the service processor*
  - **QualityManager** (Utility, ServiceExecution, API), Quality of Service (NGN Principle), *Analysis of service metric correctness; storing of violation of metrics and reporting to broker*
  - **CommunicationManager** (Utility, ServiceExecution, API), Communication (Non-functional property), *Response handling according to the requested communication mode*
- **ServiceExecution - Communication Subsystem**
  - **SignalingClientManager** (Utility, ServiceExecution, API), Signaling/Delivery Processing, *Initiation of communication legs, forwarding to the MediaSignalizer*

- **DeliveryClientManager** (Utility, ServiceExecution, API), Signaling/Delivery Processing, *Forwarding signalized requests to the MediaDeliverer*
- **TransportAbstractionManager** (Utility, ServiceExecution, API), IP Foundation (NGN), Transport Abstraction (NGN), *Transparent use of access and aggregation networks*
- **ServiceExecution - Persistence Subsystem**
  - **PersistenceManager** (Utility, ServiceExecution, API), Object Relational Mapping, *Transformation of database relationships into objects*
- **ServiceBus**
  - **Matchmaker** (Utility, ServiceRegistry, API), Decoupling Contract from Implementation, Service Lifecycle Support, Brokerage, *Search capabilities to match contracts to input criteria*
  - **QoSBroker** (Utility, ServiceRegistry, API), Quality of Service, *Checking of quality metrics during matchmaking and composition to guarantee an efficient service lease and composability of member services*
  - **ContractComposer** (Utility, ServiceRegistry, API), Service Lifecycle Support, *Checking of composeability during manual and atomic composition; assembling services in a dynamic composition*
- **UserController**
  - **UserManager** (Control, UserController, API), Service Layering (NGN), Centralization of Control, *Central management of user information*
  - **SessionManager** (Control, UserController, API), Service Layering (NGN), Session Facade (J2EE Pattern), *Administration of all user access session-related information in the system*
- **MediaSignalizer**
  - **SignalingManager** (Control, MediaDeliverer, API), Signaling/Delivery Processing (NGN), Service Layering (NGN), *Establishment, modification and termination of communication legs*
- **MediaDeliverer**

- **DeliveryManager** (Control, MediaDeliverer, API), Signaling/Delivery Processing (NGN), *Central delivery of all real time media directly to the client application*
- **DataStore**
  - **DataStore**, (Persistence, RDBMS, PersistenceManagementSystem), *Central storage of all persistent data in the system*

# Appendix C

## C. Appendix of Prototype Implementation

### C.1 Definition of NGN Service Types

---

The following code demonstrates the implementation of the SOTF data architecture introduced in Section 7.6 and used in the prototype implementation. The schema and its sub-schemes have been used as a foundation to implement both atomic and composite service contracts.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3   xmlns:al="http://activitylife.com/schemas"
4   targetNamespace="http://activitylife.com/schemas"
5   elementFormDefault="qualified">
6   <xs:include schemaLocation="NGNServiceMetaTypes.xsd"/>
7   <xs:include schemaLocation="NGNServicePropertyTypes.xsd"/>
8   <xs:include schemaLocation="NGNServiceCompositeTypes.xsd"/>
9   <xs:include schemaLocation="NGNServiceMetaProperties.xsd"/>
10  <xs:element name="NGNService" type="al:NGNServiceType"/>
11  <xs:complexType name="NGNServiceType">
12    <xs:sequence>
13      <xs:group ref="al:MetaContractGroup"/>
14      <xs:group ref="al:TypesContractGroup" minOccurs="0"/>
15      <xs:group ref="al:MethodsContractGroup"/>
16      <xs:group ref="al:FlowContractGroup" minOccurs="0"/>
```

```

17     </xs:sequence>
18 </xs:complexType>
19 <xs:group name="MetaContractGroup">
20   <xs:sequence>
21     <xs:element name="MetaInfo">
22       <xs:complexType>
23         <xs:sequence>
24           <xs:element ref="al:Id"/>
25           <xs:element ref="al:Name"/>
26           <xs:element name="description" type="al:AttributeType"/>
27           <xs:element name="classification" type="al:AttributeType"/>
28           <xs:element name="Type" type="al:AttributeType"/>
29         </xs:sequence>
30       </xs:complexType>
31     </xs:element>
32     <xs:element name="Provider" minOccurs="0" maxOccurs="1">
33       <xs:complexType>
34         <xs:sequence>
35           <xs:element ref="al:Id"/>
36           <xs:element ref="al:Name"/>
37           <xs:element name="Classification" type="al:AttributeType"/>
38           <xs:element name="Location" type="al:AttributeType"/>
39         </xs:sequence>
40       </xs:complexType>
41     </xs:element>
42     <xs:element name="Cluster" minOccurs="0" maxOccurs="1">
43       <xs:complexType>
44         <xs:sequence>
45           <xs:element ref="al:Id"/>
46           <xs:element ref="al:Name"/>
47           <xs:element name="Classification" type="al:AttributeType"/>
48           <xs:element name="Cohesion" type="al:AttributeType"/>
49         </xs:sequence>
50       </xs:complexType>
51     </xs:element>
52     <xs:element name="MethodProperties"
53       type="al:PropertiesType" minOccurs="0" maxOccurs="1"/>
54   </xs:sequence>
55 </xs:group>
56 <xs:group name="TypesContractGroup">
57   <xs:sequence>
58     <xs:element ref="al:Id"/>
59     <xs:element ref="al:Name"/>
60   </xs:sequence>
61 </xs:group>
62 <xs:group name="MethodsContractGroup">
63   <xs:sequence>
64     <xs:element name="MethodInfo" minOccurs="1" maxOccurs="unbounded">
65       <xs:complexType>

```

```
66     <xs:sequence>
67       <xs:element ref="al:Id"/>
68       <xs:element ref="al:Name"/>
69       <xs:element name="Input" type="al:ParameterType"
70         maxOccurs="unbounded"/>
71       <xs:element name="Output" type="al:ParameterType"
72         maxOccurs="unbounded"/>
73       <xs:element name="EndpointAddress" type="al:AttributeType"/>
74     </xs:sequence>
75   </xs:complexType>
76 </xs:element>
77 <xs:element name="MethodProperties" type="al:PropertiesType"
78   minOccurs="0" maxOccurs="1"/>
79 </xs:sequence>
80 </xs:group>
81 <xs:group name="FlowContractGroup">
82   <xs:sequence>
83     <xs:element name="Flow">
84       <xs:complexType>
85         <xs:sequence maxOccurs="unbounded">
86           <xs:element ref="al:Id"/>
87           <xs:element ref="al:Name"/>
88           <xs:element name="Connector" type="al:ConnectorType"
89             minOccurs="0" maxOccurs="unbounded"/>
90           <xs:element name="ServiceReference" type="al:ServiceReferenceType"
91             minOccurs="0" maxOccurs="unbounded"/>
92           <xs:element name="FlowCondition" type="al:FlowConditionType"
93             minOccurs="0" maxOccurs="unbounded"/>
94           <xs:element name="Transition" type="al:TransitionType"
95             minOccurs="0" maxOccurs="unbounded"/>
96           <xs:element name="Behavior" type="al:BehaviorType"
97             minOccurs="0" maxOccurs="unbounded"/>
98           <xs:element name="SequenceBehavior" type="al:SequenceBehaviorType"
99             minOccurs="0" maxOccurs="unbounded"/>
100         </xs:sequence>
101       </xs:complexType>
102     </xs:element>
103   </xs:sequence>
104 </xs:group>
105 </xs:schema>
```



# Glossary

The glossary entries have been extracted from different sources: [Con07a], [Con04], [Ins06d], [Uni06], and [For06].

**.NET** A programming development and execution environment based on the Microsoft Windows operational system and the idea of a Common Language Runtime (CLR).

**3GPP** Abbreviation for *3rd Generation Partnership Project*: A group of international standards bodies and wireless carriers as well as vendors with the responsibility of standardizing mobile communications based on the WCDMA technology.

**3GPP2** Abbreviation for *3rd Generation Partnership Project 2*: A standardization body defining mobile communications based on the CDMA2000 technology.

**ACID** Abbreviation for *Atomicity - Consistency - Isolation - Durability*: The concepts refers to the main capabilities to be guaranteed from a software system when processing a transaction.

**AIN** Abbreviation for *Advanced Intelligent Network*: A conceptual successor of the Intelligent Network with enhanced feature support (e.g. Voice Recognition).

**API** Abbreviation for *Application Programming Interface*: A set of conventions defining how a service is invoked by a consumer.

**ASM** Abbreviation for *Abstract State Machine*: Synonymous with Finite State Machine (FSM), a software system-modeling methodology illustrating a software system as an aggregate of a finite number of possible states.

- ATM** Abbreviation for *Asynchronous Transfer Mode*: Switching technology that uses virtual channels instead of dedicated circuits to carry data.
- BNF** Abbreviation for *Backus-Naur Form*: A metalanguage used to formally describe the syntax of other computer languages.
- BPM** Abbreviation for *Business Process Management*: A knowledge field that considers the modeling and transition of business processes into IT and the measuring and management of the processes in line with business economies.
- BPML** Abbreviation for *Business Process Modeling Language*: A metalanguage for the modeling of business processes.
- CDMA** Abbreviation for *Code-Division Multiple Access*: A cellular telecommunications access technology using direct sequence spread spectrum techniques.
- CE** Abbreviation for *Customer Equipment*: Equivalent to CPE, physical devices that are connected to a telecommunication network.
- Circuit Switched Networks** A telecommunication network technology that establishes a dedicated physical communications connection between endpoints through the network for the duration of the communication session.
- CPE** Abbreviation for *Customer Premises Equipment*: Equivalent to CE, physical devices that are connected to a telecommunication network.
- CRM** Abbreviation for *Customer Relationship Management*: Business process of managing a companies customer assets.
- CS** Abbreviation for *Capability Set*: A release cycle of the Intelligent Network which defines certain capabilities of the release.
- CSCF** Abbreviation for *Call Session Control Function*: An integral part of the IMS specification defining several roles of SIP servers in the control layer.
- DAML** Abbreviation for *Darpa Agent Markup Language*: A descriptive programming language that provides a rich set of constructs with which to create ontologies and to mark up information so that it is machine-readable and understandable.

- 
- DBC** Abbreviation for *Design by Contract*: A software development methodology based on the design of computation obligations and benefits between a supplier and a client.
- Delay** In telecommunications the time elapsing when a transmission packet is traveling from sender to receiver.
- ERP** Abbreviation for *Enterprise Resource Planning*: Software-supported set optimization of business processes to manage inventory and resources across departments in an enterprise.
- ESB** Abbreviation for *Enterprise Service Bus*: A distributed middleware that facilitates integration and interoperation across systems.
- ETSI** Abbreviation for *European Telecommunications Standardization Institute*: A major European telecommunications standardization body.
- FE** Abbreviation for *Functional Entity*: The nodes of the Functional Plane (DFP) of an Intelligent Network (IN), e.g. SSF (Service Switching Function) and SCF (Service Control Function).
- GPRS** Abbreviation for *General Packet Radio Service*: A packet data transmission protocol used in wireless networks.
- GSM** Abbreviation for *Global System for Mobile Communications*: A digital cellular network standard protocol.
- HCI** Abbreviation for *Human Centered Computing*: A concept and practice of designing computations and computational artifacts in alignment with human behavior.
- IDE** Abbreviation for *Integrated Development Environment*: A software system combining an editor and a compiler as well as other useful tools in one package.
- IETF** Abbreviation for *Internet Engineering Task Force*: A standardization body that defines standard Internet operating technology such as TCP/IP.
- IMS** Abbreviation for *IP Multimedia Subsystem*: A telecommunication architecture that allows the use of IP multimedia applications within a mobile 3G system.

- IN** Abbreviation for *Intelligent Network*: A telecommunication system that decouples delivery from control and enhances a unified service creation methodology (UFM) as well as conceptualizes an encapsulated format for a telecommunication service.
- INCM** Abbreviation for *Intelligent Network Conceptual Model*: A model consisting of four layers intended to provide a framework toward IN engineering standardization.
- IP** Abbreviation for *Internet Protocol*: One of the core network communication protocols specifying the format of packets (datagrams) and their addressing scheme.
- ISDN** Abbreviation for *Integrated Services Digital Network*: A high-speed data and media communication system providing transmission of analog and digital services.
- ITU** Abbreviation for *International Telecommunications Union*: An international standards organization established for the standardization and regulation of telecommunications.
- J2SE** Abbreviation for *Java Standard Edition*: A basic distribution of the Java programming API.
- Java EE** Abbreviation for *Java Enterprise Edition*: A Java platform designed for the mainframescale computing typical of large enterprises.
- JBI** Abbreviation for *Java Business Integration*: A Java based SOA framework based a component framework and normalized message routing as well as a management framework.
- JCP** Abbreviation for *Java Community Process*: An open organization of international Java developers and licensees who develop and revise Java technology specifications and reference as well as implementations and technology compatibility kits through a formal process.
- Jini** A network architecture for the construction of distributed systems in the form of modular cooperating services.
- JTAPI** Abbreviation for *Java Telephony API*: A programming API facilitating the access to call control services.
- LI** Abbreviation for *Lawful Intercept*: The interception of telecommunication by law enforcement agencies in accordance with public regulations.

---

**Loss** The aggregation of all lost packets of a transmission.

**Mobile Phone** A long-range and portable electronic device for mobile communication.

**MOM** Abbreviation for *Message Oriented Middleware*: A distributed software infrastructure which facilitates the interoperability and flexibility of systems based on messaging.

**OMA** Abbreviation for *Open Mobile Alliance*: A standardization body aiming to develop mobile service enabler specifications which support the creation of interoperable end-to-end mobile services.

**OMG** Abbreviation for *Object Management Group*: A computing industry collaboration to promote object-oriented interoperability among heterogeneous computing.

**OOP** Abbreviation for *Object-Oriented Paradigm*: A computation pattern relying on the use of objects that describe the attributes and behavior of a software system.

**OSE** Abbreviation for *OMA Service Environment*: A software runtime environment for software enablers defined by the OMA standardization body.

**OSS** Abbreviation for *Operational Support Systems*: Systems that handle procedures to support the daily operation of a telecommunication operators infrastructure.

**OWL-S** Abbreviation for *Ontology Web Language - Services*: An ontology based on the OWL language providing a core set of markup language constructs for describing the properties and capabilities of web services.

**PAIS** Abbreviation for *Process-Aware Information Systems*: Umbrella term for software systems concerned with the organization and implementation of business processes.

**PDA** Abbreviation for *Personal Digital Assistant*: A small mobile handheld device that provides computing and information storage and retrieval capabilities for personal or business use.

**PE** Abbreviation for *Physical Entity*: Term used for the definition of the nodes of the Physical Plane (PP) of an Intelligent Network (IN), e.g. SSP (Service Switching Point).

- POTS** Abbreviation for *Plain Old Telephone Service*: Analog telephone service deployed in most of the existing public telecommunication networks.
- PSTN** Abbreviation for *Public Switched Telephone Network*: The public circuit-switched analog telephone network.
- QoS** Abbreviation for *Quality of Service*: The ability to ensure specific and quantifiable levels of performance in a network.
- RG** Abbreviation for *Residential Gateway*: A hardware device which provides connectivity from a home network to a wide area network.
- SaaS** Abbreviation for *Software as a Service*: Software license model characterized by outsourced hosting and maintenance used by multiple tenants.
- SDP** Abbreviation for *Service Delivery Platform*: A software framework classification delivering telecommunication services compliant to the service paradigm.
- SF** Abbreviation for *Service Feature*: A capability of a telecommunication service which used e.g. in combination with Intelligent Networks (IN).
- SIB** Abbreviation for *Service-Independent Building Block*: An encapsulated architectural concept of a telecommunication service used in the Intelligent Network (IN).
- SLA** Abbreviation for *Service Level Agreement*: A contract between a service provider and a consumer that specifies in measurable terms what services the network service provider will furnish.
- SLEE** Abbreviation for *Service Logic Execution Environment*: A software-based service execution environment that centralizes and standardizes the execution of service in a single instance.
- SOC** Abbreviation for *Service-Oriented Computing*: Synonym for Service-Oriented Development, a software paradigm evolved from objects and components focused on the computation of the software service lifecycle.
- SOTF** Abbreviation for *Service-Oriented Telecommunication Framework*: The software framework developed in this thesis.

---

**TISPAN** Abbreviation for *Telecoms and Internet converged Services and Protocols for Advanced Networks*: A standardization body that specializes in fixed networks and Internet convergence and strongly supports the 3GPP IMS standardization committee.

**TMF** Abbreviation for *Telemanagement Forum*: A standardization body dedicated to operations systems support (OSS) and communication management issues in telecommunication.

**TOGAF** Abbreviation for *The Open Group Architectural Framework*: A detailed method and set of supporting resources for developing an enterprise architecture.

**UA** Abbreviation for *User Agent*: A user endpoint (e.g. a software application) of the communication network.

**UFM** Abbreviation for *Unified Functional Methodology*: A unified service creation methodology used in the Intelligent Network (IN).

**UML** Abbreviation for *Unified Modeling Language*: A formal language for the specification and documentation of the components of software systems.

**UMTS** Abbreviation for *Universal Mobile Telecommunications System*: Mobile broadband and packet-based transmission system of multimedia content at high data rates.

**VoIP** Abbreviation for *Voice over IP*: A system enabling voice data to be delivered using the IP (Internet Protocol).

**WIN** Abbreviation for *Wireless Intelligent Network*: A concept that combines wireless technology and Intelligent Networks with the intention of providing Intelligent Network (IN) capabilities in roaming networks.

**WSMO** Abbreviation for *Web Service Modeling Ontology*: A conceptual model for describing various aspects of semantic web services whose purpose is to solve the application integration problem.

**XML** Abbreviation for *eXtensible Markup Language*: A specification for markup languages which facilitates the separation of structure, format and content of data,



# Bibliography

- [2107] Interface 21. Spring framework documentation. <http://www.springframework.org/documentation>, 2007.
- [Aa004] Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype", 2004.
- [AA02] Ender Ayanoglu and Nail Akar. B-ISDN (broadband integrated services digital network), May 2002.
- [Aal03] W.M.P. van der Aalst. Don't go with the flow web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.
- [ABJ<sup>+</sup>06] Jaime Arguello, Brian S. Butler, Elisabeth Joyce, Robert Kraut, and Kimberly S. Ling. Talk to me: foundations for successful individual-group interactions in online communities. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 959–968, New York, NY, USA, 2006. ACM Press.
- [Acc99] Centre For Telecommunications Access. Evolution of standards for advanced telecommunications services and network management, August 1999.
- [ADHF06] Marek J Sergot Andrew D H Farrell. Formalising workflow: A ccs-inspired characterisation of the yawl workflow patterns. *Hewlett Packard Lab Press*, 2006.
- [ADK<sup>+</sup>05] Vikas Agarwal, Koustuv Dasgupta, Neeran M. Karnik, Arun Kumar, Ashish Kundu, Sumit Mittal, and Biplav Srivastava. A service creation environment based on end

- to end composition of web services. In Allan Ellis and Tatsuya Hagino, editors, *WWW*, pages 128–137. ACM, 2005.
- [AE02] Jan Øyvind Agedal and Earl F. Ecklund, Jr. Modelling QoS: Towards a UML profile. *Lecture Notes in Computer Science*, 2460, 2002.
- [AHK05] Patrick Albert, Laurent Henocque, and Mathias Kleiner. Configuration-based workflow composition. In *ICWS*, pages 285–292. IEEE Computer Society, 2005.
- [AJ98] C Storey A Johne. New service development: A review of the literature and annotated bibliography. *European Journal of Marketing*, 1998.
- [AKC05] Juan Ye Lorcan Coyle Simon Dobson Paddy Nixon Adrian K. Clear, Stephen Knox. Integrating multiple contexts and ontologies in a pervasive computing framework. 2005.
- [All06] Open Mobile Alliance. Oma ad service environment. <http://www.openmobilealliance.org/>, 22 Jun 2006.
- [Arn04] Karine Arnout. *From Patterns to Components*. PhD thesis, ETH Zurich, 2004.
- [Ars05] Ali Arsanjani. Toward a pattern language for service-oriented architecture and integration, part 1: Build a service eco-system. 13 Jul 2005.
- [Ast07] Asterisk open source telephony engine. <http://www.asterisk.org>, 2007.
- [AZ03] Bjoern Althun and Martin Zimmermann. Multimedia streaming services: specification, implementation, and retrieval. In *MIR '03: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 247–254, New York, NY, USA, 2003. ACM Press.
- [AZV] C. D. Anagnostakis, Th. B. Zahariadis, and I. S. Venieris. Guidelines for the evolution of core network capabilities for the support of advanced multimedia services.

- 
- [Bag04] Anindo Bagchi. Digest of technical information. Technical report, Telcordia Technologies Inc., 2004.
- [Bak01] David E. Bakken. *Middleware*. Kluwer Academic Press, 2001.
- [Bea07a] Bea weblogic server and weblogic express documentation. <http://e-docs.bea.com/wls/docs81>, 2007.
- [Bea07b] Bea weblogic workshop 8.1. <http://edocs.bea.com/workshop>, 2007.
- [Bea07c] Weblogic sip server. <http://e-docs.bea.com/wlcp/wlss21>, 2007.
- [BG04] Annika Hinze Bryan Genet. Open issues in semantic query optimization in relational dbms. <http://www.cs.waikato.ac.nz/~hinze>, 2004.
- [BGP06] Sami Bhiri, Claude Godart, and Olivier Perrin. Transactional patterns for reliable web services compositions. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 137–144, New York, NY, USA, 2006. ACM Press.
- [Bre05] Breeze documentation. <http://www.adobe.com/support>, 2005.
- [Bro91] M. Broy. Declarative specification and declarative programming. In *Proceedings of the 6th International Workshop on Software Specification and Design*, pages 2–11. IEEE Computer Society Press, 1991.
- [BS02] Microsoft Corporation Inc. BEA Systems, International Business Machines Corporation. Web services transaction (ws-transaction). <http://dev2dev.bea.com/pub/a/2004/01>, 2002.
- [CCF<sup>+</sup>00] Rost S. Cost, Ye Chen, Tim Finin, Yannis Labrou, and Yun Peng. Using colored petri nets for conversation modeling. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 178–192. Springer-Verlag: Heidelberg, Germany, 2000.

- [CDA<sup>+</sup>06] Edward Clarkson, Day, Jason A., Foley, and James D. An educational digital library for human-centered computing. In *Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems*, volume 2 of *Work-in-progress*, pages 646–651, 2006.
- [CGB05] Rafael Ceballos, Rafael Martinez Gasca, and Diana Borrego. Constraint satisfaction techniques for diagnosing errors in design by contract software. In *SAVCBS '05: Proceedings of the 2005 conference on Specification and verification of component-based systems*, page 11, New York, NY, USA, 2005. ACM Press.
- [CJ01] D. Chakraborty and A. Joshi. Dynamic service composition: State-of-the-art and research directions, 2001.
- [CKMRM03] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: a critical review and considered forecast. 2003.
- [CLC06] Chi-Hung Chi, Lin Liu, and Choon-Keng Chua. Web content consistency through explicit ownership definition. In *IEEE SCC*, pages 531–532. IEEE Computer Society, 2006.
- [CLL06] Wu Chou, Feng Liu, and Li Li. Web service for telecommunication. In *AICT/ICIW*, page 88. IEEE Computer Society, 2006.
- [Coc02] J.-Y. Cochenec. Activities on next-generation networks under global information infrastructure in itut. *Communications Magazine, IEEE Volume 40, Issue 7, July 2002 Page(s)98 - 101 Digital Object Identifier 10.1109/MCOM.2002.1018013*, 2002.
- [Con97] TINA Consortium. Tina-c service architecture. Technical report, TINA Consortium, 1997.
- [Con04] World Wide Web Consortium. Glossary. <http://http://www.w3.org/TR/ws-gloss/>, 2004.
- [Con07a] Atlassian Confluence. Sail glossary. <http://www.telscenter.org>, 2007.

- 
- [Con07b] World Wide Web Consortium. Resource description framework (rdf). <http://www.w3.org/RDF>, 2007.
- [CRM91] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *Proc. ACM Conf. Human Factors in Computing Systems, CHI*, pages 181–188. ACM, April 1991.
- [CSLK22] D. Chae-Sub Lee; Knight. Realization of the next-generation network. *2005, Communications Magazine, IEEE Volume 43, Issue 10, Oct. 2005 Page(s)34 - 41 Digital Object Identifier 10.1109/MCOM.2005.1522122*.
- [Cyb06] Dieter Cybok. A grid workflow infrastructure. *Concurrency and Computation: Practice and Experience*, 18(10):1243–1254, August 2006.
- [DD04] Remco Dijkman and Marlon Dumas. Service-oriented design A multi-viewpoint approach. December 2004.
- [DF90] UC Berkeley D. Ferrari. Rfc-1821, integration of real-time services in an ip-atm network architecture. <http://www.faqs.org/rfcs/rfc1193.html>, November 1990.
- [DK07] Carsten Bether Daniel Kellmerein, Brenden Lane. Redefining service portfolios. *Detecon Management Report*, 2007.
- [DPC11] W3C DAML Program Consortium. Owl-s: Semantic markup for web services v. 1.1. <http://www.daml.org/services/owl-s/1.1/overview/>, 2004-11.
- [DS05] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. August 2005.
- [EC01] Greg Meredith Sanjiva Weerawarana Erik Christensen, Francisco Curbera. Web services description language (wsdl) 1.1. <http://www.w3.org/TR/wsdl>, 2001.
- [EG05] M. Naci Akkk Ere Goektuerk. Paradigm and software engineering. <http://heim.ifi.uio.no/nacia>, 2005.

- [Eld97] C.A. Eldering. Customer premises equipment for residential broadband networks. *Communications Magazine, IEEE Volume 35, Issue 6, June 1997 Page(s)114 - 121 Digital Object Identifier 10.1109/35.587714*, 1997.
- [Elf02] P Elfatatry, A.; Layzell. Software as a service a negotiation perspective. *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International 26-29 Aug. 2002 Page(s)501 - 506 Digital Object Identifier 10.1109/CMP-SAC.2002.1045054*, 2002.
- [EP07] Andy Seaborne Eric Prud'hommeaux. Sparql query language for rdf. <http://www.w3.org/TR/rdf-sparql-query>, 2007.
- [Fau02] F. Le Faucheur. Rfc 3270 - multi-protocol label switching (mpls) support of differentiated services. <http://www.faqs.org/rfcs/rfc3270.html>, May 2002.
- [FdGGN<sup>+</sup>02] R. A. Falbo, dr. G. Guizzardi, A. Natali, G. Bertollo, F. Ruy, and P. Mian. Towards semantic software engineering environments. page 477. ACM Press, 2002.
- [FM04] George Feuerlicht and Sooksathit Meesathit. Design framework for interoperable service interfaces. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 299–307, New York, NY, USA, 2004. ACM Press.
- [For06] The Telemanagement Forum. Telecom glossary 2000. <http://www.atis.org/tg2k/t1g2k.html>, 2006.
- [GA006] *MW<sub>4</sub>SOC '06 Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW<sub>4</sub>SOC 2006)*, New York, NY, USA, 2006. ACM Press. Conference Chair-Karl M. Gaeschka and Conference Chair-Schahram Dustdar and Conference Chair-Frank Leymann and Conference Chair-Stefan Tai.
- [GB01] Jeff Carpenter Guy Bieber. *Introduction to Service-Oriented Programming*, 2001.

- 
- [GBBB05] M González-Berges, F Bernard, and R Barillère. Frameworking, 2005.
- [GC06] Miguel-Angel Garca-Martn Gonzalo Camarillo. *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds, 2nd Edition*. John Wiley and Sons, Chichester, UK, February 2006.
- [GHJV00] Gamma, Helm, Johnson, and Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, Massachusetts, 2000.
- [GK97] Uwe Glässer and Rene Karges. Abstract state machine semantics of SDL. *J. UCS*, 3(12):1382–1414, 1997.
- [Gn05] Selda Gner. Service oriented architecture. Master’s thesis, Technical University Hamburg Harburg, Software System Institute, 2005.
- [Gro01] C. Gronroos. *Service Management and Marketing A Customer Relationship Management Approach, 2nd edition*. John Wiley and Sons, Chichester, UK, 2001.
- [Gro02] D. Grossman. Rfc 3260 - new terminology and clarifications for diffserv. <http://www.faqs.org/rfcs/rfc3260.html>, April 2002.
- [Gro04] ITU-T Rec. A.7 Focus Group. Itu-t rec. y.2011, general principles and general reference model for next generation networks. Technical report, 2004.
- [Gro06] The Open Group. *TOGAF, Version 8, Documentation*, Nov 2006.
- [Gro03] The Object Management Group. Unified modeling language: Superstructure. <http://www.omg.org/docs/formal/07-02-03.pdf>, 2007-03.
- [Gru08] Tom Gruber. Ontology. *Encyclopedia of Database Systems*, 2008.

- [GS03] Jack Greenfield and Keith Short. Software factories: assembling applications with patterns, models, frameworks and tools. In Ron Crocker and Guy L. Steele Jr, editors, *OOPSLA Companion*, pages 16–27. ACM, 2003.
- [HB03] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *Proceedings of the Fourteenth Australasian Database Conference (ADC 2003)*, Adelaide, Australia, 2003.
- [HKHA06] B. Hirsch, T. Konnerth, A. Hessler, and S. Albayrak. A serviceware framework for designing ambient services. In A. Mana and V. Lotz, editors, *Developing Ambient Intelligence (AmID'06)*, pages 124–136. Springer France, 2006.
- [HM02] Berners-Lee Tim Hendler, James and Eric Miller. Integrating applications on the semantic web. *Journal of the Institute of Electrical Engineers of Japan, Vol 122(10)*, 2002.
- [HMT07] M. Evenson (eTel) E. Szczekocka (TP) H. Ksiezak (TP) H.Kupidura (TP) R. Belecheanu (OU) P. Krysteva(Nexcom) D. Roman (LFUI) P. Todorova (ONTO) O. Lopez (iSOCO) L.Cicurel (iSOCO) H. Munoz (TID), N. Prez (TID). *Telecom Vertical Domain and Process Ontologies*, 2007.
- [Hua97] Thierry Hua. Personal communications services through the evolution of fixed and mobile communications and the intelligent network concept, jun 1997.
- [Hua05] J. Huang. [http://www.bptrends.com/publicationfiles/01-05\\_eTOM and ITIL - huang.pdf](http://www.bptrends.com/publicationfiles/01-05_eTOM_and_ITIL_-_huang.pdf), 2005.
- [IFT05] Donald F. Ferguson Jeffrey Frey Seve Graham Tom Maguire David Snelling Ian Foster, Karl Czajkowski and Steven Tuecke. Modeling and managing state in distributed systems: The role of ogis and wsrf. *Proceedings of the IEEE Volume 93, Issue 3, Mar 2005 Page(s):604 - 612 Digital Object Identifier 10.1109/JPROC.2004.842766*, 2005.
- [Ins06a] European Telecommunications Standards Institute. Osa overview, 5.2, 1st draft es 203 915-1 v1.3.1.

- 
- <http://portal.etsi.org/docbox/TISPAN/Open/OSA>, 11, 2006.
- [Ins06b] European Telecommunications Standards Institute. Osa overview, 5.2, 1st draft es 203 915-1 v1.3.1. <http://parlay.org/imwp>, 11, 2006.
- [Ins06c] European Telecommunications Standards Institute. Osa framework, 6.0, draft etsi es 204 915-3 v0.0.2 (2006-12). <http://portal.etsi.org/docbox/TISPAN/Open/OSA>, 12, 2006.
- [Ins06d] The Balanced Scorecard Institute. Glossary. <http://www.balancedscorecard.org/basics>, 2006.
- [Ios06] P.; Pouwelse J.; Epema D. Iosup, A.; Garbacki. Correlating topology and path characteristics of overlay networks and the internet. *Cluster Computing and the Grid Workshops, 2006. Sixth IEEE International Symposium on Volume 2, 16-19 May 2006 Page(s)10 - 10*, 2006.
- [ISO96] Information technology — Syntactic metalanguage — Extended BNF. Technical Report 14977, ISO/IEC JTC1/SC22, 1996.
- [IT93] Study Group XVIII ITU-T. Recommendation i-112 vocabulary of terms for isdns, 1993.
- [IT04a] ITU-T. Itu-t rec. y.2001, general overview of ngn. Technical report, 2004.
- [IT04b] Study Group 13 ITU-T. <http://www.itu.int/ITU-T>, 2004.
- [Jav07] Java ee apis and docs. <http://java.sun.com/javaee>, 2007.
- [JB06] F. Baker J. Babiarez, K. Chan. Configuration guidelines for diffserv service classes. <http://tools.ietf.org/html/rfc4594>, August 2006.
- [Jen98] K. Jensen. A brief introduction to colored petri nets. In *Proc. Workshop on the Applicability of Formal Models, 2 June 1998, Aarhus, Denmark*, pages 55–58, 1998.

- [JF07] Holger Lausen Joel Farrell. Semantic annotations for wsdl and xml schema. <http://www.w3.org/TR/sawsdl>, 2007.
- [JM03] Michael Stevens Sunil Mathew James McGovern, Sameer Tyagi. *Java Web Services Architecture*. Morgan Kaufmann; Book and CD-Rom edition, April 25, 2003.
- [Joh97] R. Johnson. Frameworks = (components + patterns). *Communications of the ACM*, 40(10):39–42, 1997.
- [JS05] A. Jaimes and N. Sebe. Multimodal human computer interaction: A survey. In *Computer Vision in Human-Computer Interaction*, page 1, 2005.
- [KB96] Randy H. Katz and Eric A. Brewer. The case for wireless overlay networks. In Tomasz Imielinski and Henry F. Korth, editors, *Mobile Computing*, pages 621–650. Kluwer Academic Publishers, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1996. <http://daedalus.cs.Berkeley.edu/>.
- [Kel98a] Wolfgang Kellerer. Dienstarchitekturen in der telekommunikation - evolution, methoden und vergleich. 1998.
- [Kel98b] Wolfgang Kellerer. Multimedia service architectures - an overview. 1998.
- [Kel02] Wolfgang Kellerer. *Serverarchitektur zur netzunabhaengigen Dienssteuerung in heterogenen Kommunikationsnetzen*. PhD thesis, Technische Universitaet Muenchen, Fakultaet fuer Elektrotechnik und Informationstechnik, 2002.
- [Ker96] Thomas David Kern. The realization of the advanced intelligent network. Master’s thesis, University of Colorado, 1996.
- [Kes05] Thomas Kessler. Ena - extended network architecture. Technical report, Detecon International GmbH, 2005.
- [Kot88] Philip Kotler. *Marketing Management Analysis, Planning, Implementation, and Control*. Prentice Hall International, 6th edition, 1988.

- 
- [Kru92] Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, 1992.
- [Kru98] Philippe Kruchten. Modeling component systems with the unified modeling language. 1998.
- [KS05] Alexander Knapp and Harald Störrle. Unified modeling language 2.0. In *VL/HCC*, page 9. IEEE Computer Society, 2005.
- [Kuh96] T. Kuhn. *The Structure of Scientific Revolutions*, 3rd. The University of Chicago Press, Chicago, IL, 1996.
- [KvDK06] Rick Kazman, Arie van Deursen, and Rainer Koschke. Introduction to the special on software architecture reconstruction and modeling. *Automated Software Engineering*, 13(2):199–200, 2006.
- [Lab07] JBoss Labs. Hibernate – relational persistence for idiomatic java. [http://www.hibernate.org/hib\\_docs](http://www.hibernate.org/hib_docs), 2007.
- [Ler03] Reuven M. Lerner. At the forge: content management. *Linux J.*, 2003(108):11, 2003.
- [LG06] Kitty Wilson Jarrett Lillian Goleniewski. *Telecommunications Essentials, Second Edition: The Complete Global Source*. Addison Wesley Professional, October 10, 2006.
- [Li94] J. Li, G.; Bacon. Supporting distributed real-time objects. *Parallel and Distributed Real-Time Systems, 1994. Proceedings of the Second Workshop on 28-29 April 1994* Page(s):138 - 143, 1994.
- [Llo95] J. W. Lloyd. Declarative programming in escher. Technical Report CSTR-95-013, Department of Computer Science, University of Bristol, June 1995.
- [LRPF04] Rubén Lara, Dumitru Roman, Axel Polleres, and Dieter Fensel. A conceptual comparison of WSMO and OWL-S. In Liang-Jie Zhang and Mario Jeckle, editors, *Web Services — Proceedings of the 2004 European Conference on Web Services*, volume 3250 of *Lecture Notes in Computer Science*, pages 254–269, Erfurt, Germany, September 2004. Springer-Verlag.

- [LRS02] Frank Leymann, Dieter Roller, and Marc-Thomas Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.
- [Luc07] Apache lucene. <http://lucene.apache.org/java/docs>, 2007.
- [Med04] Brahim Medjahed. *Semantic Web Enabled Composition of Web Services*. PhD thesis, Virginia Tech, 2004.
- [Mey88] Bertrand Meyer. *Object-oriented Software Construction*. Prentice-Hall, 1988.
- [Mil91] R. Milner. The polyadic pi-calculus: a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, October 1991.
- [Mil02] Mark A. Miller. *Building the Converged Network*. MT Books, 2002.
- [MK04] Susan Bishop Alan Hopkins Sven Milinski Chris Nott Rick Robinson Jonathan Adams Paul Verschueren Martin Keen, Amit Acharya. *Patterns: Implementing an SOA Using an Enterprise Service Bus*. IBM, July 2004.
- [ML05] Jianfei Yin Manshan Lin, Heqing Guo. Goal description language. 2005.
- [MWK05] Thomas Magedanz, Dorota Witaszek, and Karsten Knuettel. The IMS playground @ fokus - an open testbed for next generation network multimedia services. In *TRIDENTCOM*, pages 2–11. IEEE Computer Society, 2005.
- [MyS07] Mysql documentation. <http://dev.mysql.com/doc>, 2007.
- [NB04] Klara Nahrstedt and Wolf-Tilo Balke. A taxonomy for multimedia service composition. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 88–95, New York, NY, USA, 2004. ACM Press.

- 
- [NB05] Marc Fiammante Keith Jones Rawn Shah Norbert Bieberstein, Sanjay Bose. *Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap*. IBM Press, October 25, 2005.
- [OAF88] Kiyoshi Ono, Mikio Aoyama, and Hiroshi Fujimoto. Data management of telecommunications networks. In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 201–201, New York, NY, USA, 1988. ACM Press.
- [OAS04] OASIS. Universal description, discovery and integration (uddi) v, 3.0.2. <http://www.oasis-open.org>, 2004.
- [OAS06] OASIS. Web services security: Soap message security v. 1.1. <http://www.oasis-open.org>, 2006.
- [OAS07] OASIS. Web services business process execution language (bpel) v. 2.0. <http://www.oasis-open.org>, 2007.
- [PC03] Claus Pahl and Michael Casey. Ontology support for web service processes. In *ESEC / SIGSOFT FSE*, pages 208–216, 2003.
- [PC07] Micha Rj Tomasz Rybicki Pawe Cielak, Jarosaw Domaszewicz. *A proof-of-concept ontology of telecommunication services, 1st version*, 2007.
- [PCM06] PCMag. [http://www.pcmag.com/encyclopedia\\_term](http://www.pcmag.com/encyclopedia_term), 2006.
- [PGM04] Ricardo de Almeida Falbo Paula Gomes Mian. Building ontologies in a domain oriented software engineering environment. 2004.
- [Pro98] Telecommunications Industries Analysis Project. The electronic apple pie deploying advanced telecommunications capabilities to all americans, 1998.
- [Pry89] Martin De Prycker. Evolution from ISDN to BISDN a logical step towards ATM. *Computer Communications*, 12(3):141–146, 1989.

- [PSM03] Rudolf Pailer, Johannes Stadler, and Igor Miladinovic. Using parlay apis over a sip system in a distributed service platform for carrier grade multimedia services. *Wirel. Netw.*, 9(4):353–363, 2003.
- [PSWW04] D. J. Palmer, S. K. Shrivastava, S. M. Wheeler, and S. J. Woodman. Notations for the specification and verification of composite web services, 2004.
- [RB94] S. Shenker R. Braden, D. Clark. Rfc 1633 - integrated services in the internet architecture: an overview. <http://www.faqs.org/rfcs/rfc1633.html>, June 1994.
- [RK03] Sanjiva Weerawarana Rania Khalaf, Nirmal Mukhi. Serviceoriented composition in bpel4ws, 2003.
- [RMB00] William A. Ruh, Francis X. Maginnis, and William J. Brown. *Enterprise Application Integration: A Wiley Tech Brief*. John Wiley & Sons, Chichester, England, October 2000.
- [Rob91] II; Robrock, R.B. The intelligent network-changing the face of telecommunications. *Proceedings of the IEEE, Volume 79, Issue 1, Jan. 1991 Page(s)7 - 20, Digital Object Identifier 10.1109/5.64379*, 1991.
- [Rot07] Arnon Rotem. *SOA Pattern*. Manning Publications, 2007.
- [RZ03] Simone Rottger and Steen Zschaler. CQML+: Enhancements to CQML, 2003.
- [Sae05] M Saeed, M.; Jaffar-Ur-Rehman. Enhancement of software engineering by shifting from software product to software service. *Information and Communication Technologies, 2005. ICICT 2005. First International Conference on 27-28 Aug. 2005 Page(s)302 - 308*, 2005.
- [SC94] C.D. Sharp and K. Clegg. Advanced intelligent networks - now a reality. *Electronics, Communication Engineering Journal Volume 6, Issue 3, June 1994 Page(s)153 - 162*, 1994.
- [Sch05] Prof. Dr. Alexander Schill. Service-oriented architectures: Potential and challenges.

- 
- [http://www.rn.inf.tu-dresden.de/scripts\\_lsrn](http://www.rn.inf.tu-dresden.de/scripts_lsrn), 2005.
- [(SD06)] Sun Developer Network (SDN). Core j2ee patterns. <http://java.sun.com/blueprints>, 2006.
- [SDfNGNGBY06] E.; Deschrevel J.P.; Crespi N.; Service Definition for Next Generation Networks Grida Ben Yahia, I.; Bertin. Service definition for next generation networks. *Networking, International Conference on Systems and Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on 23-29 April 2006 Page(s)22 - 22 Digital Object Identifier 10.1109/ICNICONSMCL.2006.194*, 2006.
- [Seb89] Robert W. Sebesta. *Concepts of Programming Languages*. Benjamin/Cummings, Redwood City, Calif., 1989.
- [Sip03] Jsr 116: Sip servlet api. <http://jcp.org/en/jsr>, 2003.
- [SKY03] Zhou Su, Jiro Katto, and Yasuhiko Yasuda. Replication algorithms to retrieve scalable streaming media over content delivery networks. In *MIR '03: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 255–261, New York, NY, USA, 2003. ACM Press.
- [SPH04] E. Sirin, B. Parsia, and J. Hendler. Composition-driven filtering and selection of semantic web services, 2004.
- [SR06] Arnaud Simon and Thomas Rischbeck. Service contract template. In *SCC '06: Proceedings of the IEEE International Conference on Services Computing (SCC'06)*, page 511, Washington, DC, USA, 2006. IEEE Computer Society.
- [SS97a] J. Wroclawski S. Shenker. Rfc 2215 - general characterization parameters for integrated service network elements. <http://www.faqs.org/rfcs/rfc2215.html>, September 1997.
- [SS97b] R. Guerin S. Shenker, C. Partridge. Rfc 2212 - specification of guaranteed quality of service. <http://www.faqs.org/rfcs/rfc2212.html>, September 1997.

- [Sta05] Christian Stahl. A petri net semantics for BPEL, 2005.
- [Sun99] Sun Microsystems. *Jini Architectural Overview – Technical White Paper*, January 1999. <http://www.sun.com/software/jini/whitepapers>.
- [Szy] Clemens Szyperski. Component technology what, where, and how? In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pages 684–693. IEEE Computer Society, 2003.
- [Tec99] Information Society Technologies. Information society technologies 1999 workprogramme. Technical report, European Commission, 1999. Online at <http://www.cordis.lu/ist/home.html>.
- [Tom06] Apache tomcat 6.0. <http://tomcat.apache.org>, 2006.
- [Tsa05] W.T Tsai. Service-oriented system engineering a new paradigm. *Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop 20-21 Oct. 2005 Page(s)3 - 6 Digital Object Identifier 10.1109/SOSE.2005.34*, 2005.
- [TvS02] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, Upper Saddle River, NJ, 2002.
- [Uni] Princeton University. Wordnet search - 2.1.
- [Uni06] Carnegie Mellon University. Glossary. <http://www.sei.cmu.edu/opensystems/glossary.html>, 2006.
- [uSSG06] Z-Punkt Gmbh und Sinus Sociovision GmbH. Kommunikationstreiber und sinus milieus. Technical report, T-Online International GmbH, 2006.
- [vdAADtH04] Wil van der Aalst, Lachlan Aldred, Marlon Dumas, and Arthur ter Hofstede. Design and implementation of the YAWL system. January 2004.
- [VID03] Luca Vollerero, Giulio Iannello, and Francesco Delfino. An open software architecture for structured data elaboration and, 2003.

- 
- [W3C00] W3C. Simple object access protocol (soap) v. 1.1. <http://www.w3.org/TR/soap>, 2000.
- [W3C01] W3C. Web services description language (wsdl) v. 1.1. <http://www.w3.org/TR/wsdl>, 2001.
- [W3C04] W3C. Web services choreography description language v. 1.0. <http://www.w3.org/TR/2004>, 2004.
- [W3C06] W3C. Web services policy 1.2 - framework (ws-policy). <http://www.w3.org/Submission/WS-Policy>, 2006.
- [WA01] Jim Waldo and Ken Arnold. *The Jini specifications*. Addison Wesley, pub-AWadr, second edition, 2001.
- [Wal06] Kevin Wallace. *CCVP QOS Quick Reference Sheets*. Cisco Press, 2006.
- [Woe06] Dr. Wolfgang Woelker. Next generation service and system management. DTAG Internal Project, 2006.
- [WY06] ShaohuaTang Wenya Yang. A solution for web services transaction. : *Hybrid Information Technology, 2006. ICHIT'06. Vol 2*, 2006.
- [WZZJ05] Xinming Wang, Gansen Zhao, Xin Zhang, and Beihong Jin. An agent-based model for web services transaction processing. In *EEE*, pages 186–189. IEEE Computer Society, 2005.
- [XZH06a] K.; Schlichting R.D. Xianan Zhang; Hiltunen, M.A.; Marzullo. Customizable service state durability for service oriented architectures. *Dependable Computing Conference, 2006. EDCC '06. Sixth European Oct. 2006 Page(s):119 - 128 Digital Object Identifier 10.1109/EDCC.2006.8*, 2006.
- [XZH06b] K.; Schlichting R.D. Xianan Zhang; Hiltunen, M.A.; Marzullo. Customizable service state durability for service oriented architectures. *Dependable Computing Conference, 2006. EDCC '06. Sixth European Oct. 2006 Page(s):119 - 128 Digital Object Identifier 10.1109/EDCC.2006.8*, 2006.

## Bibliography

---

- [Yee06] G. Yee. Personalized security for e-services. *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on 20-22 April 2006* Page(s):8 pp. Digital Object Identifier 10.1109/ARES.2006.92, 2006.
- [YL05] Tao Yu and Kwei-Jay Lin. A broker-based framework for qoS-aware web service composition. In *EEE*, pages 22–29. IEEE Computer Society, 2005.
- [ZB96] V. Zeithaml and M. J. Bitner. *Services Marketing*. McGraw-Hill, New York, NY, 1996.
- [ZBD<sup>+</sup>03] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM Press.
- [Zha04] J. Rhea S.C.; Joseph A.D. Kubiawicz J.D. Zhao, B.Y.; Ling Huang; Stribling. Tapestry a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on Volume 22, Issue 1, Jan. 2004* Page(s)41 - 53 Digital Object Identifier 10.1109/JSAC.2003.818784, 2004.