Formalization of a Converged Internet and Telecommunications Service Environment

von

Dipl. Informatiker (FH) Niklas Blum

Dissertation
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften

– Dr.-Ing. –

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam
am 15. September 2010

Betreuer:

Prof. Dr. Tiziana Margaria-Steffen	Universität Potsdam
Prof. Dr. Thomas Magedanz	Technische Universität Berlin

Ich erkläre hiermit, dass die Arbeit bisher an keiner anderen Hochschule eingereicht worden ist sowie selbständig und ausschließlich mit den angegebenen Mitteln angefertigt wurde.

Berlin, den 15. September 2010

Niklas Blum

Acknowledgements

I have been supported throughout the last years by several people. Without the valuable feedback in discussions and great team work, I would not have been able to reach my own goal of writing this thesis.

I owe Fraunhofer Institute FOKUS, University of Potsdam, and University of Applied Science in Leipzig (HTWK) a debt of gratitude for a great education and surrounding study and work atmosphere allowing me to proceed in my research and daily work based on values of freedom of thought and mind and trust in my abilities. I especially would like to thank Tiziana Margaria and Thomas Magedanz for their support and confidence in my person and work.

Several research projects sponsored by Deutsche Telekom and Nippon Telegraph and Telephone (NTT) allowed me to work on topics related to my thesis and collect necessary experience and knowledge. In these projects and my daily work I was supported by a team of nice, talented, and aspiring colleagues, especially Andreas Bachmann, Alexander Blotny, Irina Boldea, Irina Colgiu, Simon Dutkowski, Jan Kleeßen, Sebastian Lampe, Lajos Lange, Alice Motanga, and Florian Schreiner. I would also like to apologize at this point for moments of bad temper to Florian. Sometimes I was under a lot of pressure and I wish especially you to also reach the point to write the acknowledgements of your own thesis soon.

My parents have been very supportive in the last years helping me to reach this goal. I hope that I make you proud of your son with this achievement.

Special gratitude goes to my wife Eva who has been an incredible backing during all this time. Thank you for being so appreciative, especially during all these weekends I spent at my desk instead with you.

To my daughter.

Abstrakt der Dissertation

Formalisierung einer Dienstumgebung für konvergente Internet- und Telekommunikationsdienste

von Niklas Blum

betreut durch Prof. Dr. Ing. Tiziana Margaria-Steffen
und Prof. Dr. Ing. Thomas Magedanz

Das programmierbare Netz, das Ende des 20. Jahrhunderts in der Standardisierung und Forschung für das Intelligente Netz entworfen wurde, wird nun Realität in einem auf das Internet Protokoll basierendem Netz der nächsten Generation (Next Generation Network). Hierfür kommen Prinzipien aus der Informationstechnologie, insbesondere aus dem Bereich dienstorientierte Architekturen (Service-Oriented Architecture / SOA) für die Diensterstellung, -ausführung und -betrieb zum Tragen. SOA bietet hierbei die theoretische Grundlage für Telekommunikations-netze, vor allem jedoch für die dazugehörigen Dienstplattformen. Diese erlauben dem Telekommunikationsbetreiber seine Position in einem offenen Marktplatz der Dienste auszubauen. Dazu bedarf es allerdings möglichst flexibler Dienstumgebungen, die die Kooperation zwischen Dienstanbietern und Nutzern aus unterschiedlichsten Domänen durch Unterstützung geeigneter Werkzeuge und Mechanismen fördert.

Im Rahmen dieser Dissertation definieren wir aufbauend auf Forschungsergebnisse im Bereich Overlay-Netze, Netzabstraktion und Zugriff auf exponierte Dienste eine *Service Broker* genannte Dienstumgebung für konvergente Internet- und Telekommunikationsdienste. Dieser Service Broker stellt Mechanismen für die Komposition von Diensten und Mediation zwischen unterschiedlichen Dienstparadigmen und Domänenspezifika beim Dienstaufruf zur Verfügung.

Der Forschungsbeitrag dieser Arbeit findet auf unterschiedlichen Ebenen statt: Aufbauend auf einer Analyse und Klassifikation von Technologien und Paradigmen aus den Bereichen Informationstechnologie (IT) und Telekommunikation diskutieren wir die Problemstellung der Kooperation von Diensten und deren Komposition über Domänengrenzen hinweg. In einem zweiten Schritt diskutieren wir Methoden der Dienstkomposition und präsentieren eine formalisierte Methode der modellbasierten Diensterstellung. Der Schwerpunkt der Arbeit liegt auf der Spezifikation der Service Broker Dienstumgebung und einem zugrundeliegenden Informations- und Datenmodell. Diese Architektur erlaubt die Komposition und Kooperation von Diensten über Domänengrenzen hinweg, um konvergente Internet- und Telekommunikationsdienste zu realisieren. Hierfür wird ein auf Obligationspolitiken basierendes Regelsystem formalisiert, das Interaktionen zwischen Dienstmerkmalen während der Diensterstellung und -ausführung definiert.

Abstract of the Dissertation

Formalization of a Converged Internet and Telecommunications Service Environment

by Niklas Blum

under supervision of Prof. Dr. Ing. Tiziana Margaria-Steffen
and Prof. Dr. Ing. Thomas Magedanz

The programmable network envisioned in the 1990s within standardization and research for the Intelligent Network is currently coming into reality using IP-based Next Generation Networks (NGN) and applying Service-Oriented Architecture (SOA) principles for service creation, execution, and hosting. SOA is the foundation for both next-generation telecommunications and middleware architectures, which are rapidly converging on top of commodity transport services. Services such as triple/quadruple play, multimedia messaging, and presence are enabled by the emerging service-oriented IP Multimedia Subsystem (IMS), and allow telecommunications service providers to maintain, if not improve, their position in the marketplace. SOA becomes the de facto standard in next-generation middleware systems as the system model of choice to interconnect service consumers and providers within and between enterprises.

We leverage previous research activities in overlay networking technologies along with recent advances in network abstraction, service exposure, and service creation to develop a paradigm for a service environment providing converged Internet and Telecommunications services that we call Service Broker. Such a Service Broker provides mechanisms to combine and mediate between different service paradigms from the two domains Internet/WWW and telecommunications. Furthermore, it enables the composition of services across these domains and is capable of defining and applying temporal constraints during creation and execution time. By adding network-awareness into the service fabric, such a Service Broker may also act as a next generation network-to-service element allowing the composition of cross-domain and cross-layer network and service resources.

The contribution of this research is threefold: first, we analyze and classify principles and technologies from Information Technologies (IT) and telecommunications to identify and discuss issues allowing cross-domain composition in a converging service layer. Second, we discuss service composition methods allowing the creation of converged services on an abstract level; in particular, we present a formalized method for model-checking of such compositions. Finally, we propose a Service Broker architecture converging Internet and Telecom services. This environment enables cross-domain feature interaction in services through formalized obligation policies acting as constraints during service discovery, creation, and execution time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

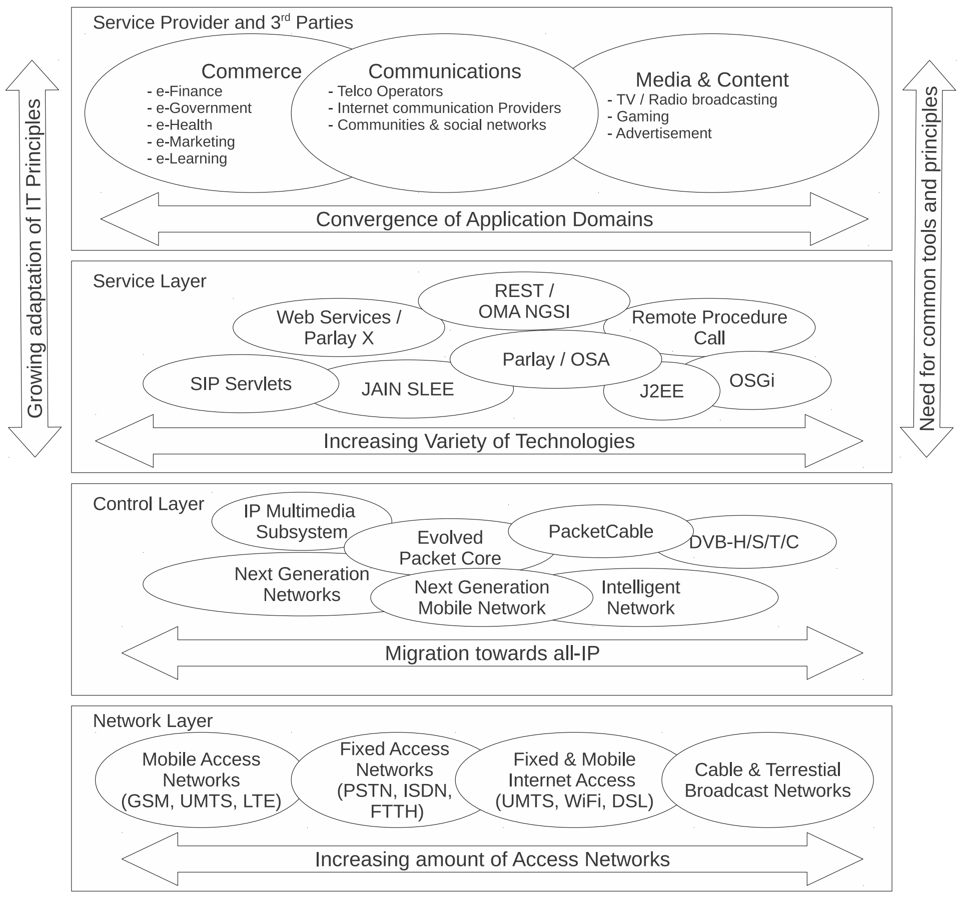
1.1 Background and Motivation

During the last few years, we can witness the introduction of Next Generation Networks (NGN) aiming at enhancing telecommunication systems from vertically separated application-specific networks towards a unified horizontal layered Internet Protocol (IP)-based network. This provides a homogeneous control layer to enable integrated Internet, telecommunications, and media services as well as network convergence (e.g., Fixed Mobile Convergence/FMC). Using IP as a common signalling protocol, information and telecommunication technologies are merged on a network control level. At present, NGN is known as a pioneer mean for achieving the *any where, any time* service providing promise.

As a result of the implementation of new control infrastructures and technologies for message signalling, new technologies for service provisioning and creation are needed, too. Traditionally, developing telecommunications applications and services were part of the network operator's domain, whose technical staff was intimately familiar with details of the network and proprietary interfaces of the equipment in use. Business processes using these applications were extensive and had multiple steps and many people were involved in the developing processes.

Within the last decade the marketplace for communications services has changed dramatically. Ready-to-deploy services can be acquired from several vendors and network operators can concentrate on the actual tasks to support and administrate their networks and services. Nevertheless, these services continue to act as isolated features that do not provide access for cross-domain interactions and integrated services. Furthermore, operators are facing challenges from multiple sides, as increased competition, the need to invest in new networks, platforms, and new services, which are eroding time-tried revenue streams like voice services, and regulation by public authorities. Simultaneously, new trends in the Web have come up which have changed the way people communicate (e.g., community portals, presence and location-based services).

Figure 1.1 illustrates this change of principles, technologies, and convergence of networks and platforms on the four levels of network layer, control layer, service layer, and for service providers and 3rd parties.



Service Provider and 3rd Parties
Commerce
- e-Finance
- e-Government
- e-Health
- e-Marketing
- e-Learning
Communications
- Telco Operators
- Internet communication Providers
- Communities & social networks
Media & Content
- TV / Radio broadcasting
- Gaming
- Advertisement
Convergence of Application Domains
Service Layer
REST / OMA NGSI
Web Services / Parlay X
Remote Procedure Call
Parlay / OSA
SIP Servlets
JAIN SLEE
J2EE
OSGi
Increasing Variety of Technologies
Control Layer
IP Multimedia Subsystem
Evolved Packet Core
PacketCable
DVB-H/S/T/C
Next Generation Networks
Next Generation Mobile Network
Intelligent Network
Migration towards all-IP
Network Layer
Mobile Access Networks (GSM, UMTS, LTE)
Fixed Access Networks (PSTN, ISDN, FTTH)
Fixed & Mobile Internet Access (UMTS, WiFi, DSL)
Cable & Terrestial Broadcast Networks
Increasing amount of Access Networks
Growing adaptation of IT Principles
Need for common tools and principles

Figure 1.1: Change of Platforms and Technologies across Layers

Alongside this evolution, we can observe the adaptation of platform principles from Information Technologies (IT) in the service layer. Formerly closed, proprietary platforms designed for the Intelligent Network (IN) make now use of industry standards and best practices for service creation, hosting, and execution originally designed for large enterprises and the World Wide Web (WWW). This is commonly referred to as the adaptation of Service Oriented Architectures (SOA) to the telecommunications service domain.

From a service provider and user perspective, we can furthermore witness the convergence of formerly isolated application domains into a converged triple /

quadruple play service offering, spanning across the communications and entertainment/content sector. The adaptation of IT principles based on SOA for Internet and communications services is also mostly visible reflecting the growing amount of reusable e-services in the Internet that also make use of real-time functionality.

To cope with this development, major telecommunications operators in Europe[1] have launched their own (proprietary) open development programmes offering communication related enabler as SMS, voice call and conferencing, location. The target of these portals is to expose simple interfaces for developers, who can combine these services to create web applications or integrate those telecommunications functions into existing modelled business processes and other products. Via these developer portals, 3^{rd} party service developers get access to core network functionalities using open Application Programming Interfaces (API) and Software Development Kits (SDK). Opening-up the network for service developers will affect all infrastructure levels of an operator and service provider ranging from provisioning of customer equipment, monitoring traffic of 3^{rd} party services, to the establishment of appropriate service enablers for exposed Telecom and non-Telecom services. Therefore, it is of great importance for the operator to implement effective mechanisms of control to allow the development of an open service market on top of the network infrastructure. The following figure 1.2 provides an overview of issues and concerns that operator and service provider need to cope with in a growing market of services.

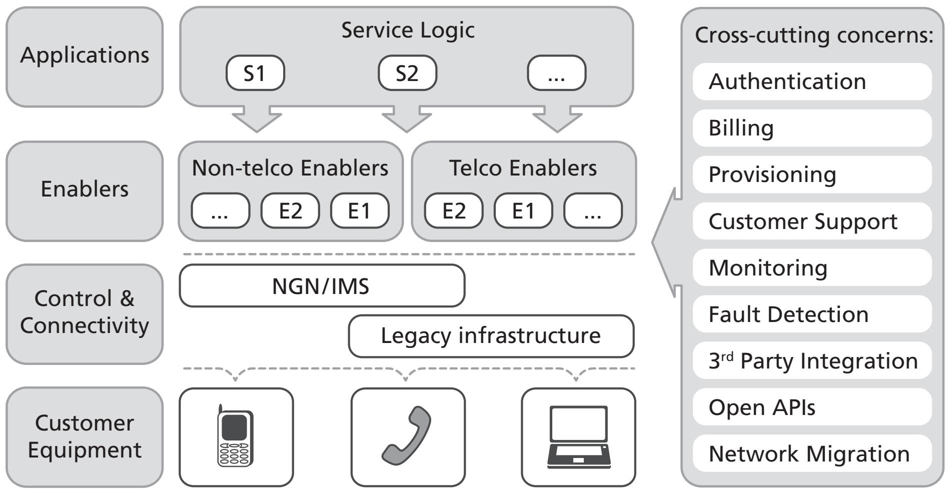


Figure 1.2: Areas of operator concerns in a growing service market

From a service developer perspective, we can witness a growing number of small and medium enterprises (SME) as well as integrators that are eager to create applic-

[1]see www.developergarden.com, www.o2litmus.co.uk, www.orangepartner.com, www.ribbit.com

ations for the converging telecommunications / Internet service space using high level APIs and programming languages with the aim to create new distributed services or integrate existing communication channels into business processes. The emerging digital marketplace presents an important opportunity to offer communications capabilities as services. Besides communication services like short message service (SMS), voice-call or location, further capabilities like billing, identity management, authentication or control of Quality of Service (QoS) are candidates as telecommunications enablers for the development of new applications.

Allowing the service designers and developers to effectively make use of the new converged service domain and its technologies, tools for service creation need to be provided that offer the creation of converged services across multiple application domains integrating a variety of service technologies on the one hand and hiding the complexity of network and platform related topics on the other hand.

1.2 Problem Statement

The above described evolution in technology ranging from the network layer to the service layer imposes changes for operators, service designers, and users. This thesis deals in detail with problems related to the increasing variety of technologies on service layer, the adaptation of IT principles for the telecommunications domain, and resulting the convergence of different application domains as depicted in the upper half of figure 1.1 from the perspective of network operators, service providers, service developers, and users.

Allowing the combination of features in a converged Telecom/Internet service space, well-defined and formalized programming paradigms in IT need to be applied to the telecommunications service domain and combined with fast evolving non-standardized, sometimes poorly described service principles in the Internet and WWW. Accordingly, the traditionally well-protected and limited opened telecommunications services need to be opened and exposed towards 3^{rd} parties for service creation and execution. Two main problem spaces open up in this context:

1. Cross-domain composition mechanisms for the creation of hybrid services consisting of multiple different service endpoints providing different techniques for description and service quality need to be applied in one service environment. A formalized description of such mechanisms is currently missing.

2. Constraints need to be applied during service execution and discovery time, especially:

    - Selection constraints that define which activities constitute a service,
    - Scheduling constraints that define when these activities are to be performed, both in terms of ordering as well as temporal dependencies,

- Resource constraints that define which resources are required to perform activities.

Such constraints need to be applied independently of underlying service and network technologies and platforms.

A general problem for developers, operators, and users are the vast amount of technologies and standards to cope with in this field. We provide classifications of service exposure API technologies, service composition/workflow expression languages, and policy expression languages in this thesis identifying technologies most suitable from the perspective of these three roles.

1.3 Scope of the Thesis and Contribution

We analyse technology trends in the economic important changing area of convergence of Internet and Telecom infrastructures and services. By examining relevant requirements for a target platform managing the interactions within a converged service layer and especially between 3^{rd} parties (service designers, developers, and users) and network operators, we identify a key component acting as a broker between the network-centric communications domain and the service-oriented application domain.

To allow the integration of (tele-)communications-specific service enablers into 3^{rd} party services, a formalism is defined to express constraints on access and usage of these enablers. We differentiate the constraints by the following purposes:

- Authorisation for service enabler access
- Service discovery
- Service access definition and control
- Service composition/orchestration

Whereas the first two points *authorisation* and *discovery* are not subject to discussion in this thesis, the latter issues related to the definition of fine-granulated service usage rules and the composition of several atomic service building blocks across multiple application domains to form complex services are the main subject of our research. A formalization of usage rules and their application to express and enforce (temporal) constraints is still missing and a common solution at the point of writing was not available. Furthermore, formalized methods for model-checking of service compositions in telecommunications is still on-going research [BMM 09].

The following figure 1.3 illustrates the role of a service broker function for the telecommunications service layer and its interaction with service providers and application domains.

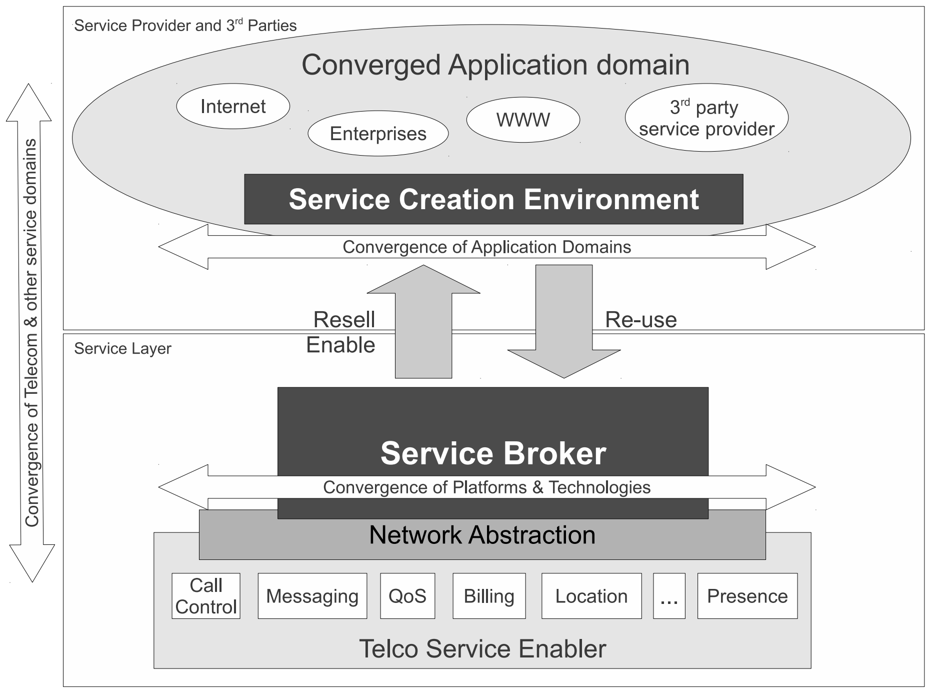

Service Provider and 3rd Parties
Converged Application domain
Internet
Enterprises
WWW
3rd party service provider
Service Creation Environment
Convergence of Application Domains
Resell Enable
Re-use
Service Layer
Service Broker
Convergence of Platforms & Technologies
Network Abstraction
Call Control
Messaging
QoS
Billing
Location
...
Presence
Telco Service Enabler
Convergence of Telecom & other service domains

Figure 1.3: Role of the Service Broker

The service broker function provides in an open telecommunications service environment vertical convergence between the business-driven service provider domain and the communications sector and horizontal convergence for platforms and technologies within a Telecom Service Delivery Platform (SDP).

Horizontal convergence of different application domains for cross-domain service composition is provided by a Service Creation Environment incorporating composition mechanisms for services from multiple domains. This needs to be tightly integrated with the service broker function to allow also cross-layer (vertical) convergence between specific application domains and the network-centric telecommunications domain.

Figure 1.4 illustrates the thematic focal points of the proposed service broker in this thesis:

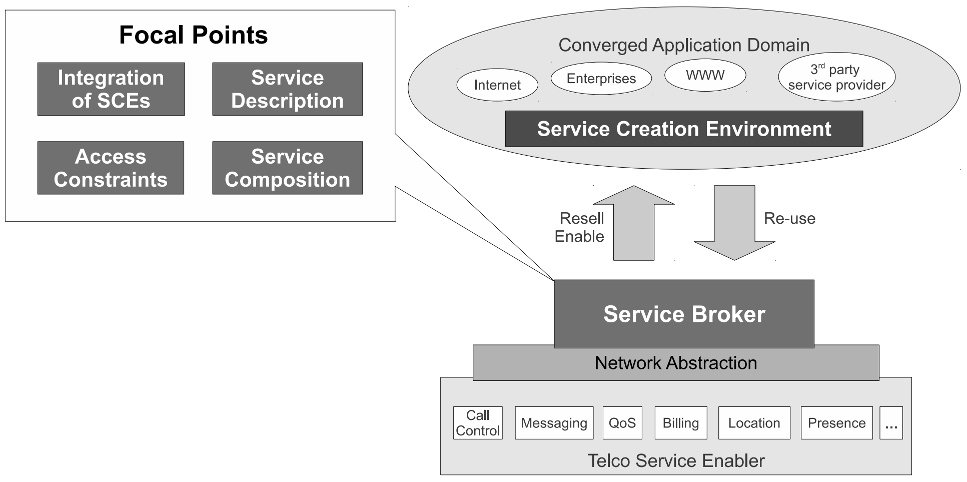


Figure 1.4: Major functional Service Broker components and research fields

Integration of Service Creation Environments
Considering the currently most prominent service execution environments for NGN, JAIN SLEE[2] and SIP Servlets, the service logic is tightly coupled with concrete technologies. In this way, services lack the ability to adapt quickly to the evolution and requirements in changing application domains. How to enhance both abstraction level of service development and re-usability of service logic is subject to research in the telecommunications field for many years (e.g., [GKM 03], [BMS+ 97], and [BHL+ 99]). This thesis deals with the integration of telecommunications specific services into Model-Driven Architecture (MDA)-based Service Creation Environments (SCE) formalising application logic as models. We discuss service creation principles and their evolution from a developer and operator perspective and present our approach for a formalisation of communications services.

Service Description
Complex services consist of a multitude of atomic features. In a distributed service environment, those features are available via Application Programming Interfaces (APIs) and protocols. We provide a classification of state of the art principles and technologies for distributed services and service description standards and best practices from a user's, developer's, and operator's perspective. Furthermore, we define a high-level Telecom API that allows integration of communication features into web-based mash-up with minimal efforts.

Service Composition
One of the main characteristics of a SOA is to expose resources transparently as

[2]Java API for Integrated Networks Service Logic Execution Environment

services to be consumed by other services. The general principles of a SOA are modularity, re-usability, service discovery, and the ability to compose services. Cross-domain composition mechanisms allowing convergence of multiple application domains are needed for the creation of hybrid services consisting of multiple different service endpoints. The Service Broker solution proposed within this thesis acts as a central control unit for service exposure and composition of telecommunications service enablers. We provide a classification of state of the art principles and technologies to compose services with a special focus on real-time capabilities required by communications services and heterogeneity of service end points. Furthermore, candidates identified to meet developer, operator, and user requirements are evaluated with the help of performance tests.

Constraints
Telecommunications operators have kept up with the *walled garden* strategy regarding limited access to services and network functionality from 3^{rd} party domains for a long time. But declining revenue and Over-The-Top (OTT) players are now forcing them to open up their networks for developers and service providers. Access to network and service capabilities on the service layer is based on constraints defined by the operator on a service provider or user-specific basis. Constraints defined above in section 1.2 can be differentiated either as permanent/-static or temporal constraints on multiple layers:

1. Model-checking within the SCE allows the verification of process and activity level constraints during the service composition process.
2. Policy enforcement at the Service Broker provides the enforcement of temporal constraints during execution time:
    - Selection constraints: Sequences of service and features to be executed, e.g., specific logging, audio/video announcements before call setup, etc.
    - Scheduling constraints: Temporal dependencies, e.g., enforcement of privacy rules based on location information, etc.
    - Resource constraints: Definition of required resources, e.g., network resources (QoS), service enabler selection, etc.

1.4 Methodology

This thesis will provide a fourfold approach addressing problems within the above described scope:

1. Analysis and classification of principles and technologies from IT and Telecoms to identify and discuss issues allowing cross-domain composition in a converging service layer.

2. Definition of requirements from the perspective the network operator, service provider, service developer, and user.

3. Discussion of service composition methods allowing the composition of converged services on an abstract level, in particular based on a formalized method for model-checking of such compositions.

4. Definition of a service broker architecture allowing converging Internet and Telecom services. This environment enables cross-domain composition mechanisms for the creation of hybrid services consisting of multiple different service endpoints and the application of constraints during service execution and discovery time.

Following the below depicted workflow, a service broker is specified as a central entity for providing cross-domain convergence of telecommunications and non-Telecom service domains:

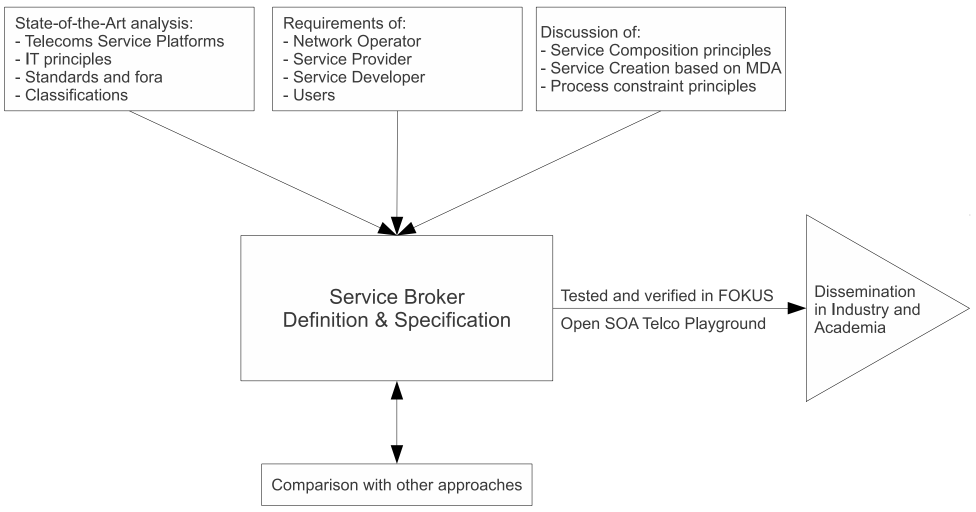

State-of-the-Art analysis:
- Telecoms Service Platforms
- IT principles
- Standards and fora
- Classifications
Requirements of:
- Network Operator
- Service Provider
- Service Developer
- Users
Discussion of:
- Service Composition principles
- Service Creation based on MDA
- Process constraint principles
Service Broker Definition & Specification
Tested and verified in FOKUS Open SOA Telco Playground
Dissemination in Industry and Academia
Comparison with other approaches

Figure 1.5: Workflow in this thesis

We define the following propositions to be discussed within this thesis:

I Technologies, mechanisms, and service paradigms in the Internet and Telecom follow different principles. Cooperative services spanning across both domains require an intermediate function for mediation between these domains.

II Policy-based service composition allows users, service providers, and network operators to express service constraints in high-performance, low-latency requirements in Telecom systems.

III Software synthesis and code generation based on formal models is a well-suited software development approach for an open Telecom systems.

1.5 Classification Properties of Key Interest

Throughout this thesis we classify technologies and architectural approaches in IT and telecommunications. This section describes the properties used to differentiate and classify architectural styles and technologies and are applied for benchmarking the developed solution. It is not intended to be a comprehensive list, we have included only those properties that are clearly influenced by the set of technologies taken into consideration. Note that not necessarily all criteria will be applied to every classification. The list of criteria extends the criteria for classifying network-based application architectures in [F 00].

1.5.1 Configurability

Configurability is related to both extensibility and re-usability in that it refers to post-deployment modification of components or configurations of components, such that they are capable of using a new service or data element types.

1.5.2 Customisability

Customisability refers to the ability to temporarily specialise the behaviour of an architectural element, such that it can then perform an unusual service. A component is customisable if it can be extended by one client of that component's services without adversely impacting other clients of that component. [MTS 03]

1.5.3 Modifiability

Service architectures are subject to change when new functionality is introduced. Modifiability is about the ease with which a change can be made to an application. [PC 07]

1.5.4 Network efficiency

An interesting observation about network-based applications is that the best application performance is obtained by not using the network. The design of distributed service architectures that use network APIs needs to take into consideration the amount and interval of data and requests to be sent over the network.

1.5.5 Powerfulness

Powerfulness refers to the ability or capacity to perform or act effectively through possession of controlling influence. In the context of this dissertation, powerfulness is applied to the complexity of operations that can be achieved and the degree of system governance to be enforced within platforms and architectures.

1.5.6 Scalability

Scalability refers to the ability of an architecture to support large numbers of components, or interactions among components, within an active configuration. Scalability is also impacted by the frequency of interactions, whether the load on a component is distributed evenly over time or occurs in peaks, whether an interaction requires guaranteed delivery or a best-effort, whether a request involves synchronous or asynchronous handling, and whether the environment is controlled or anarchic (i.e., can you trust the other components?).

1.5.7 Simplicity

Simplicity refers to the KISS principle: Keep it Simple, Stupid. It states that design simplicity should be a key goal and that unnecessary complexity should be avoided. Furthermore, simplicity is the property of a domain which requires very little information to be exhaustively described. If functionality can be allocated such that the individual components are substantially less complex, then they will be easier to understand and implement. Applying the principle of generality to architectural elements also improves simplicity, since it decreases variation within an architecture.

1.6 Outline

The thesis is structured in five main parts.

1. In Chapter 2, we provide an overview of the relevant technologies and standards in the context of this dissertation in regard of the evolution of telecommunications service environments.

2. Whereas Chapter 2 provides a rather Telecom centric perspective, Chapter 3 deals with general principles in IT, especially concerning Service Oriented Architectures. We discuss and classify principles and technologies for network-based service architectures, service composition, and the application of Model-Driven Engineering in telecommunications.

3. Chapter 4 presents our service broker architecture and the underlying information and data model. The platform goal is to encapsulate the network-driven service domain in telecommunications as services for developers and users while allowing the operator to dynamically manage access to its resources. We validate this platform with developer, network operator-, and user-oriented use cases and provide a performance analysis for high amount of requests.

4. In Chapter 5, we compare our proposed solution with other approaches in current and past research activities.

5. We summarize our work in Chapter 6 and describe the dissemination of the results of this work into other research activities regarding an open service market platform and cross-layer composition of network and service resources in the context of Future Internet (FI) initiatives.

Chapter 2

Service Principles in Telecommunications

This chapter provides an overview of the evolution of service principles and technologies in telecommunications and associated standards from various fora. Concluding, we will discuss the current state from the perspective of a developer, operator, and user point of view.

2.1 Service Evolution in Telecommunications

This section provides a chronological overview of the evolution of service platforms in telecommunications from a closed circuit-switched telephony system to an open, standardised IT multimedia infrastructure.

2.1.1 The Plain Old Telephone System

The Plain Old Telephone Service (POTS) is the voice-grade telephone service that remains the basic form of residential and small business service connection to the telephone network in most parts of the world. The name is a reflection of the telephone service still available after the advent of more advanced forms of telephony such as the Integrated Services Digital Network (ISDN), mobile phones and Voice over IP (VoIP). It has been available almost since the introduction of the public telephone system in the late 19th century, in a form mostly unchanged to the normal user despite the introduction of Touch-Tone dialling, electronic telephone exchanges and fiber-optic communication into the public switched telephone network (PSTN). While POTS provides limited features, low bandwidth, no mobile capabilities, and no standardized interfaces and platforms for developers, it provides greater reliability than other telephony systems (mobile phone, VoIP, etc.). Many telephone service providers attempt to achieve "dial-tone availability" more than 99.999% of the time the telephone is taken off-hook, equivalent to having a dial-tone available for all but less than five minutes each year.

The communications circuits of the PSTN continue to be modernised by advances in digital communications; however, other than improving sound quality, these changes have been mainly transparent to the POTS customer. In most cases, the function of the POTS local loop presented to the customer for connection to telephone equipment is practically unchanged and remains compatible with old pulse dialling telephones, even ones dating back to the early 20^{th} century.

2.1.2 The Intelligent Network

Based on the wish to establish a network independent, vendor independent, and service independent service provisioning and execution platform, the Intelligent Network (IN) was originally developed in the 1980s in the US by AT&T[1] and later Bellcore (nowadays Telcordia Inc.[2]).

The Intelligent Network is a network architecture intended both for fixed as well as mobile Telecom networks. It allows operators to differentiate themselves by providing value-added services in addition to the standard services such as PSTN, ISDN, and Global System for Mobile Communications (GSM) services on mobile phones. As illustrated in the following figure 2.1, in the IN, the intelligence is provided by network nodes owned by operators, as opposed to solutions based on intelligence in the telephone equipment, or in Internet servers provided by any part.

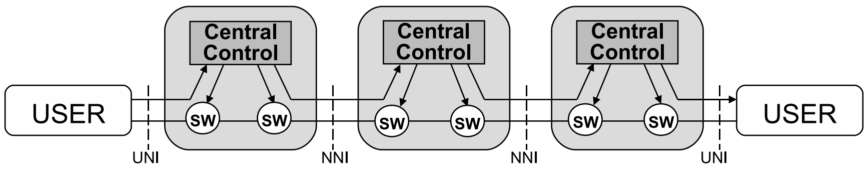


Figure 2.1: IN service switching principles

Principally the IN separates network signalling, i.e. the switching, from service signalling or the service control and thereby achieves network independence. In addition, the IN aimed basically at the delivery of value-added services (VAS) extending the POTS based on a set of reusable service building blocks and thereby providing a first step into service independence of the underlying networks. These services, such as the Customer Access Line Signalling Services (CLASS), Freephone, Premium Rate, Virtual Private Networks (VPN) provide primarily capabilities like flexible routing, flexible charging, screening, and user interactions.

[1]http://www.att.com
[2]http://www.telcordia.com/

Technically the IN demarcates the first relevant use of IT, namely the Remote Procedure Call (RPC) principle and central computers within the Telecom world. The idea was to define an overlay service architecture on top of a physical network and to extract the service intelligence from the legacy network switches into dedicated central computers, referred to as Service Control Points (SCPs). [M 93]

Service independence of the IN architecture was provided by the definition of reusable service components, so-called service independent building blocks (SIB) were able to be chained together for the realisation of new services in a Service Creation Environment (SCE), resulting in an executable program for the SCP. The International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) invented an iterative approach to the standardisation of the IN Q.1200 recommendation series [ITU1200] organised in phases, called IN Capability Sets (CS). Whereas the first CS was designed for the PSTN only, later CSs were targeted for ISDN, GSM, and IP-based networks.

Service independence was limited to a set of predefined reference services, which were decomposed into service features that could be part of several services. These features, describing for example one logical number, reverse charging, or user interaction were decomposed within a second stage into low level SIBs, representing some kind of programming approach for IN services. These SIBs, in a third step were mapped to IN functional components and their interactions, which finally in a fourth and final step led to the specification of the dedicated protocol Intelligent Network Application Part (INAP) between physical elements of the IN architecture. Therefore, an IN architecture is in principle only to a limited extent considered as service independent by the selection of benchmark services. The IN in the mobile world is referred to as Customised Applications for Mobile Enhanced Logic (CAMEL) [3G 99] and was standardised by the 3^{rd} Generation Partnership Project (3GPP) as a GSM phase 2+ feature from the mid 1990s onwards. The driving force for CAMEL was the need to provide Prepaid and VPN services to roaming users across country and network boundaries due to the success of GSM.

IN and CAMEL were successfully performing. However, the envisaged open market of services imagined originally in that time frame was not created. One main reason for this is that the SIB approach did not finally work in a standardised way and many vendors defined their own feature sets and INAP extensions. That limited the interoperability of products and vendor independence. In addition, and more important, the programmer's base for the IN was far too limited as the technology used was too Telecom specific and did not follow mainstream programming paradigms of the IT world. [M 96]

2.1.3 Telecommunication Information Networking Architecture Consortium

The Telecommunication Information Networking Architecture Consortium (TINA-C) was an attempt started in 1992 by several actors in the telecommunication world to define, design, and realise a software architecture for telecommunications infrastructures. The IN evolution expressed by the capability set approach was targeted towards a so-called IN Long-term Architecture (LTA). In the mid 1990s, programming languages like C++ and Java and especially middle-ware concepts, namely the Object Management Group's (OMG) CORBA (Common Object Request Broker Architecture) and the Java Remote Method Invocation (RMI) allowed implementing scalable and distributed service platforms and provided abstraction from the details of underlying network signalling and transport protocols [VZM 00]. The developed architecture was targeted for the future multimedia mobile broadband environment providing video conferencing and video on demand type of services in addition to existing POTS/ISDN services. The following figure 2.2 depicts the TINA-C service architecture with respect to the business model [TINA 97].

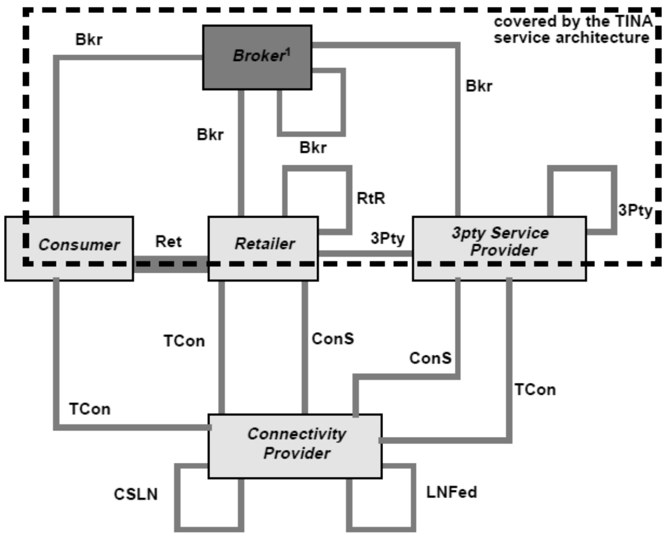


Figure 2.2: TINA-C service architecture

The TINA-C result has been a set of architectures, namely the Business Architecture, the Service Architecture, which was defined on top of a distributed processing environment representing an extended CORBA middle-ware, and a Network Resource Control Architecture enabling Quality of Service (QoS) based service provision. This architecture allowed a decoupling of networks from the service architecture and the flexible distribution of service platform components across different physical nodes. Most interesting at that time was the notion of a Service Retailer / Broker providing single sign-on capabilities for aggregated services by a set of service providers in a walled garden approach. Services were programmed by means of objects in an object-oriented manner, specific service components were not standardised.

TINA failed as the consortium specified the architectures mainly in a walled garden approach and lacked a clear migration path from existing IN architectures. Furthermore, the proposed middle-ware technologies did not meet the operational requirements of operators at that time. Nevertheless the work performed serves still as a basis for nowadays service architecture concepts.

2.1.4 Service Delivery for Next Generation Networks

In parallel to CAMEL, 3GPP started at the beginning of the 2^{nd} millennium, in regard of a new emerging core and access network evolution towards all-IP, the specification for a service control architecture: the IP Multimedia Subsystem (IMS) [3G 06].

The IMS was designed originally as an overlay on top of the General Packet Radio Service (GPRS) domain within 3GPP Release 5 specifications, published in 2003. Its main target was to provide an infrastructure for the provision of real-time multimedia services, but it has evolved since then as a general NGN control overlay network on top of any mobile, fixed, and even cable network. This means that today IMS represents the only common NGN service control standard. As such, IMS is being standardised by 3GPP for nearly every wireless access network, including Universal Mobile Telecommunications System (UMTS), Wireless Local Area Network (WLAN), Worldwide Interoperability for Microwave Access (WiMAX), and Long Term Evolution (LTE). The European Telecommunications Standards Institute (ETSI) TISPAN (TIPHON (Telecommunications and Internet Protocol Harmonization over Networks) and SPAN (Services and Protocols for Advanced Networks)) group [3] has defined additions for the fixed network domain within its Next Generation Network reference architecture definition, including PSTN/ISDN emulation [ETSI 09], IMS, and also interactive Internet Protocol based Television (IPTV) / streaming subsystems [ETSI 09a], as well as in the cable domain by CableLabs within their PacketCable 2.0 specifications [PacketCable 09].

IMS allows the integration of service platforms for efficiently implementing various multimedia communications applications, including VoIP, video calls and conferencing, instant messaging, presence, group lists, Push to Talk [BM 05a], and IPTV [ABM 07]. The architecture is based on a combination of IN/CAMEL and VoIP concepts deploying the major IP protocols for session signalling and Authorization, Authentication and Accounting (AAA), namely the Session Initiation Protocol (SIP) (it should be noted that SIP has been extensively enhanced for IMS purposes and is much more complex than basic SIP as defined in [RSC+ 02]) and Diameter [CLG+ 03], which is an evolution of the RADIUS protocol, standardised by Internet Engineering Task Force (IETF).

The major target of IMS is to provide a standardised and structured overlay network control architecture in contrast to the standard Internet for better security,

[3] http://portal.etsi.org/tispan/TISPAN_ToR.asp

QoS, and flexible charging within a single sign-on framework. Figure 2.3 illustrates the IMS architecture as part of the session control layer:

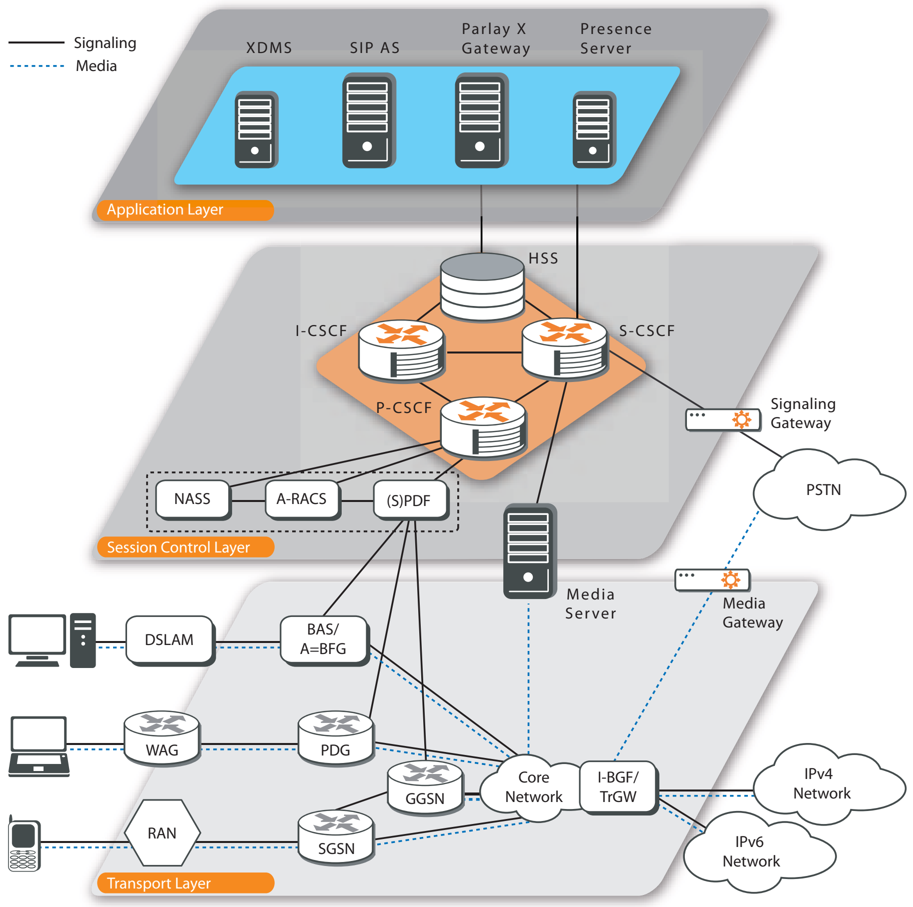


Figure 2.3: 3GPP IMS Architecture within session control layer

From an architectural point of view 3GPP defines an IMS core layer comprising a set of extended SIP servers, called Call Session Control Function (CSCFs), as well as signalling and media gateways to inter-work with fixed and mobile legacy circuit-switched networks and media servers. End users will use an enhanced SIP / IMS client to connect to the system. A specialised AAA server known as Home Subscriber System (HSS), or in case of fixed network access referred to as User Profile Server Function (UPSF) by ETSI TISPAN, stores the entire user and service related data which is exchanged with the SIP servers using the Diameter Base protocol. Application Servers are out of scope from the 3GPP perspective, but IMS provides standardized interfaces based on SIP and Diameter for the application layer as a single point of integration for service platforms.

2.1.5 Open APIs in Telecommunications

In face of the loss of momentum of TINA, which led to its official termination in 2000, a more pragmatic activity was started in 1998: the definition of Application Programming Interfaces (APIs) for telecommunications. These APIs allow the flexible implementation of services in application servers accessing the network functions via dedicated interface operations defined by the API. They are independent of the underlying network signalling protocols through the definition of protocol-specific resource adaptors and should thereby support network and vendor independence. Service independence is provided by the extensibility of an API. The major target of an API approach, however, is when publishing the API specifications openly to allow the creation of services by a 3^{rd} party community and thereby gaining more service innovation.

Using open communications APIs and inspired by the Computer Telephony Integration (CTI) environment, Private Branch Exchanges (PBXs) were among the first solutions programmed for implementing sophisticated enterprise telephony applications. These were quite similar to IN VPN services, however implemented at the edge of the network and much more flexible and innovative. One prominent example of the applied APIs was the Java Telephony Application Programming Interface (JTAPI) [JCP 02]. The Java community started with its development of Java APIs for integrated Networks (JAIN) [JCP 02a] at the end of the 1990s. In this framework Java classes for carrier grade call control (Java Call Control) were defined and Java containers were extended to support carrier grade performance, leading to the development of a Service Logic Execution Environment (SLEE) [JCP 04] which allowed the creation and execution of Telecom services based on Java Service building blocks on top of various signalling protocols, such as JAIN INAP or SIP.

In addition to these JAIN activities, the Parlay Group was founded in 1998, targeting the development of language independent and network independent Telecom service APIs. In its first version this API could be regarded as a pragmatically cut down version of the TINA Service Architecture, where applications in an application server on top of IN for fixed networks provided the services and CORBA provided the middle-ware functionality. With the extension of the Parlay scope towards mobile and IP networks and a growing number of members in the consortium, the APIs have been extended over the last ten years and feature telecommunications related capabilities, such as (multimedia) call control, conferencing, messaging, charging, location, presence, group definitions, application-layer QoS control etc. in a network independent way, i.e. hiding the signalling protocol details of the underlying telecommunications networks. Figure 2.4 provides a schematic overview of a how an API gateway based on Parlay may map application specific functionality to specific network resources.

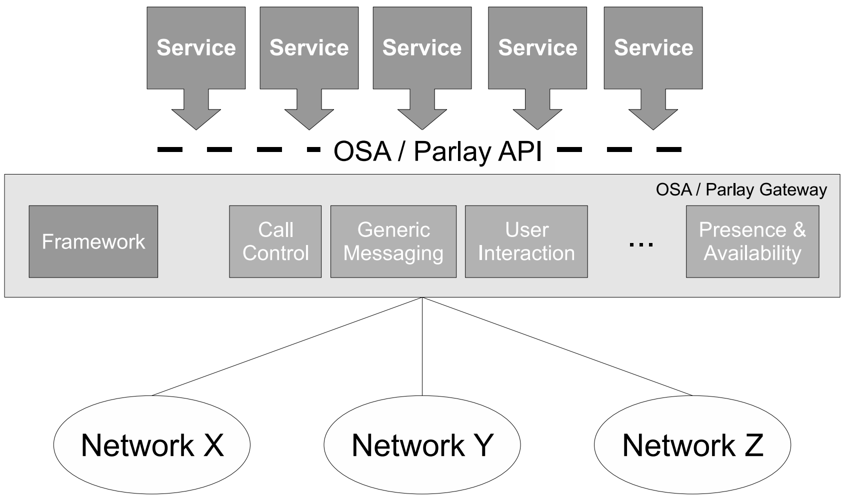


Figure 2.4: Parlay gateway for multiple network technologies

In principle, all these APIs make Telecom service implementation much easier and faster compared to the traditional IN approach by providing abstract service interfaces and leave the application developers the choice of using state of the art service development tools. As mentioned before, the basic idea of these APIs is that Application Servers (ASs) host the application logic. Developers access these APIs (via a secure network connection), which are provided by a dedicated network operator service gateway. This gateway maps the service interfaces to available resource interfaces in a flexible way not seen by the application. For example, an IN SCP could have been upgraded to become such a network gateway but also direct mappings of these APIs onto a network signalling protocol are possible. Corresponding interoperability specifications for the Parlay APIs were developed by ETSI TISPAN and should ensure the inter-operation of different application servers and gateways.

Whereas the Parlay APIs are based on CORBA, the Parlay group started in 2003 the definition of Web Services based APIs labelled as Parlay X [3G 09]. Parlay X is not considered as a successor of Parlay but rather an alternative, allowing the initiation of Telecom functionality via high-level functionality. On the one hand, this approach does not provide the developer with the functionality of creating sophisticated communications services with deep control of features. On the other hand, Parlay X hides most of the complexity of the Parlay APIs and can be considered as an offering for non-Telecom service developers that want to make use of Telecom features for their services.

Although this API technology is very promising, particularly as it provides some kind of Enterprise Architecture Integration (EAI) within a network operator infrastructure, market acceptance has taken a very long time of almost 15 years as most network operators at that time were still not ready to open up their networks to 3^{rd} parties. However, in face of changing network technologies, i.e. the migration from circuit-switched to all-IP networks, the value of OSA/Parlay APIs is fully recognised as they allow services to be provided on top of both networking domains in parallel.

2.2 Standards and Fora

This section discusses selected technical reports and emerging standards related to service brokerage in telecommunications environments, namely by 3GPP[4], the Java Community Process (JCP)[5], the Open Mobile Alliance (OMA)[6], and the TeleManagement Forum (TMF)[7] initiative. Figure 2.5 illustrates a classification of covered fora within a 4-tier horizontal model.

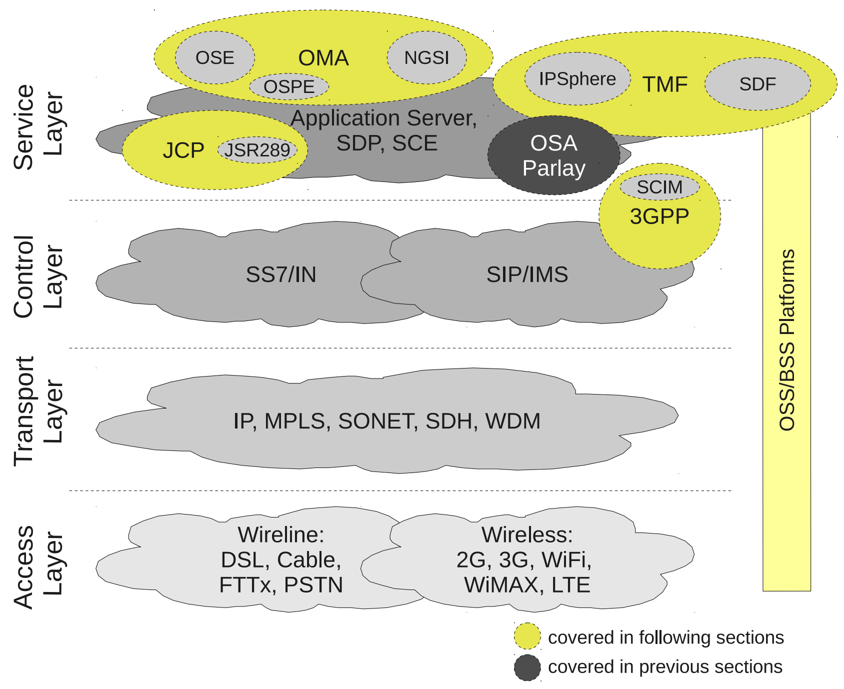

Service Layer
OSE
OMA
NGSI
OSPE
IPSphere
TMF
SDF
Application Server, SDP, SCE
JCP
JSR289
OSA Parlay
SCIM
3GPP
Control Layer
SS7/IN
SIP/IMS
Transport Layer
IP, MPLS, SONET, SDH, WDM
Access Layer
Wireline: DSL, Cable, FTTx, PSTN
Wireless: 2G, 3G, WiFi, WiMAX, LTE
OSS/BSS Platforms
covered in following sections
covered in previous sections

Figure 2.5: Relevant Standard Definition Organisations

[4]http://www.3gpp.org
[5]http://www.jcp.org
[6]http://www.openmobilealliance.org
[7]http://www.tmforum.org/

The mentioned standards are not exhaustive for this broad field but focus on the main working areas related to this thesis.

2.2.1 3rd Generation Partnership Project

3GPP has defined a function within the SIP application layer of IMS managing interactions between application servers. This function can be considered as a service broker for SIP services and has been labelled Service Capability Interaction Manager (SCIM) and defined in [3G 03]. However, the service interaction management functionalities of SCIM are not specified. Research in this field has been in progress over the last years [GC 07]. The following figure 2.6 depicts one of the first architectural classifications as part of release 5.

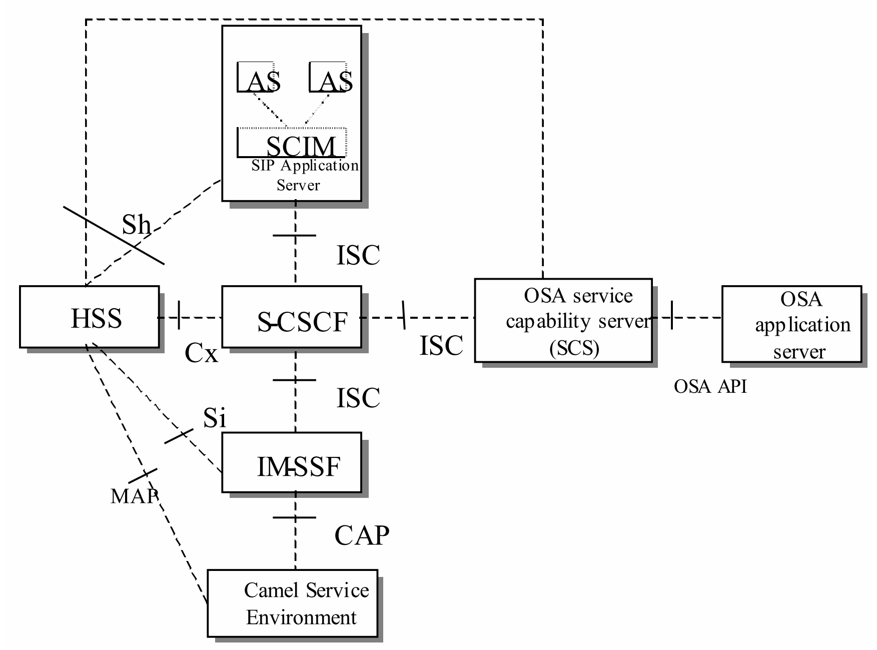


Figure 2.6: 3GPP Service Capability Interaction Manager

As part of release 8, 3GPP investigates [3G 08] impacts on defined control architectures through service brokering. A service broker is defined in this context as a logical function that manages service interactions among services hosted on single or multiple Application Servers. The main focus of the investigation is related to feature interaction between applications that are executed in a chain to form a complex service. Beyond the scope of [3G 08] are service integration between SIP and non-SIP applications available via the IMS service architecture and the support of service integration across multiple service providers. It is considered as a functional part of the IMS Service Control (ISC) reference point interfacing IMS Serving Call Session Control Functions (S-CSCF) and Application Server functions. These considerations by 3GPP extend the static invocation mechanism as part of the S-CSCF through initial Filter Criteria (iFC). 3GPP differentiates between centralised and distributed service broker functions as depicted in the following figure:

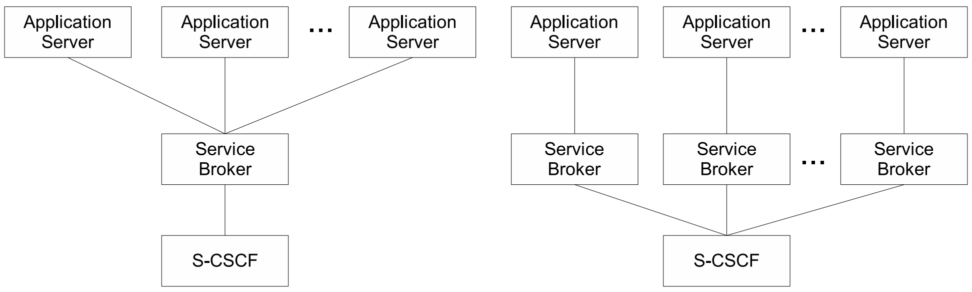


Figure 2.7: 3GPP centralised and distributed broker functions

2.2.2 Java Community Process

The Java Community Process (JCP) adopts the idea of a 3GPP SCIM as a so called Application Router (AR) as part of its SIP Servlet 1.1 specification JSR 289 [JCP 08]. The Application Router is defined as a function alongside the Servlet container. It is responsible for routing SIP requests to applications. As soon as the container receives a new request, it calls the Application Router; the AR determines which application should be selected and returns the name of that application to the container. The container then passes the request to the selected application. The following figure 2.8 is an illustration of request routing in JSR 289.

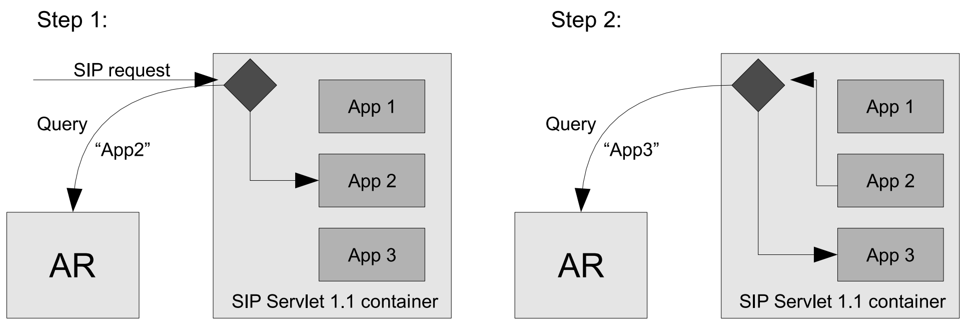


Figure 2.8: SIP Servlet 1.1 request routing based on Application Router

The AR is aware of what applications are deployed on the container, and it knows the request and the request context. Nevertheless, the AR is only limited to broker applications within a JSR 289 compliant container. The application is able to proxy the SIP request back out to the container, where it will again be passed to the AR to find the next application in a chain. The AR can tell from the request context passed to it by the container whether a request is new, or whether it has already been handled by one or more applications.

2.2.3 Open Mobile Alliance

The following subsections depict briefly standardisation efforts of the Open Mobile Alliance (OMA) that are related to work presented in this thesis, especially the Next Generation Service Interfaces (NGSI), OMA Service Environment (OSE), and the OMA Service Provider Environment (OSPE).

2.2.3.1 Next Generation Service Interfaces

The objective of Next Generation Service Interface (NGSI) [OMA 09] initiative is to define a set of new service APIs in order to stimulate the usage of various service enablers fostering the development of new services and applications. NGSI will both define extensions beyond today's Parlay X APIs (latest version: 3GPP Release 8 Parlay X 4.0 APIs [3G 09]) and define several new APIs as needed. In order to allow for advanced service creation based on multiple services/enablers, interface functionalities for identity federation and related obligations as well as configuration for the composition of services are included.

The main functionality and goal of NGSI may be described as follows:

1. OMA NGSI offers a way to applications to access network capabilities as well as to enhance existing communications with features offered via these applications.
2. Respecting device capabilities, service subscriptions, context, user profiles, privacy as well as their dynamic collaboration, the user experience can be enhanced with additional service value offered by the applications. These mechanisms are controlled and enabled via NGSI.
3. Leverage the enforced service capabilities from the applications, enhanced user experience and service consumption is provided. This step involves collaboration of network capabilities with features from the application environment.

The following figure 2.9 illustrates the scope of NGSI:

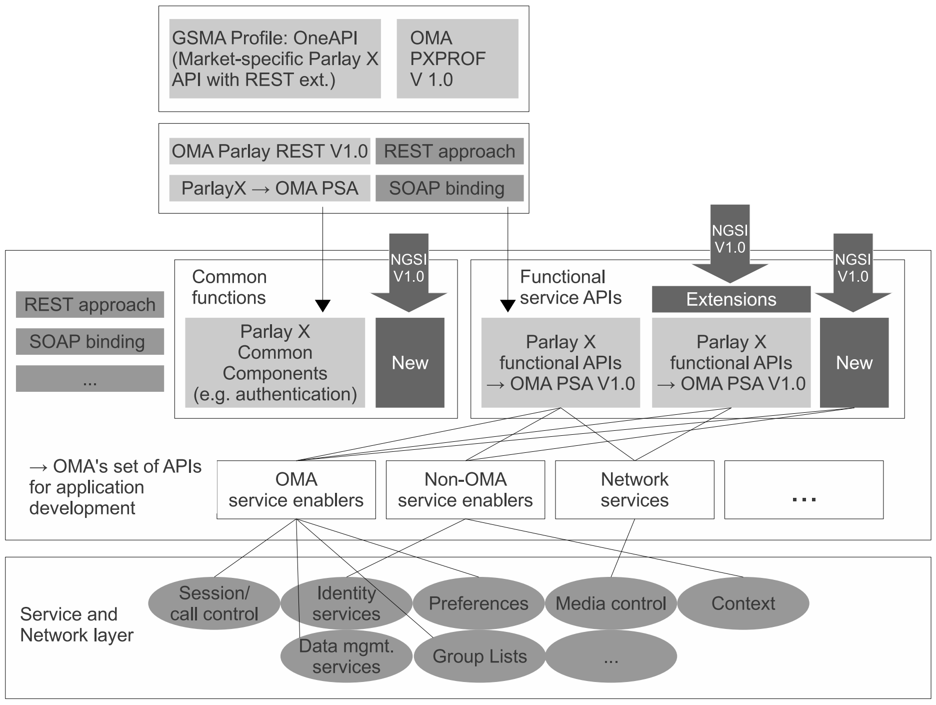


Figure 2.9: Scope of OMA NGSI 1.0

Service APIs are provided as part of the exposure layer of service delivery platforms to applications. While those APIs are offered via the service exposure layer, the service delivery platforms need to provide the necessary means to manage and govern the access of applications to network services. Thus, the set of APIs supports developers to implement their applications reusing offered network services while at the same time taking over the control and execution towards the underlying service and network layer.

2.2.3.2 OMA Service Environment

The definitions of several application service enablers by the Open Mobile Alliance and the need for a general access function for 3^{rd} party service access led to the specification of the OMA Service Environment (OSE) [OMA 07]. It defines an enabler layer which incorporates specific enabler components that offer northbound interfaces to applications that implement certain application logic. These applications either reside at the operator domain or are hosted at a 3^{rd} party domain. An enabler component can either be part of the OSE or the OSE can act as an application overlay that offers interfaces to other enabler functions. Figure 2.10 illustrates the proposed architecture by the OMA:

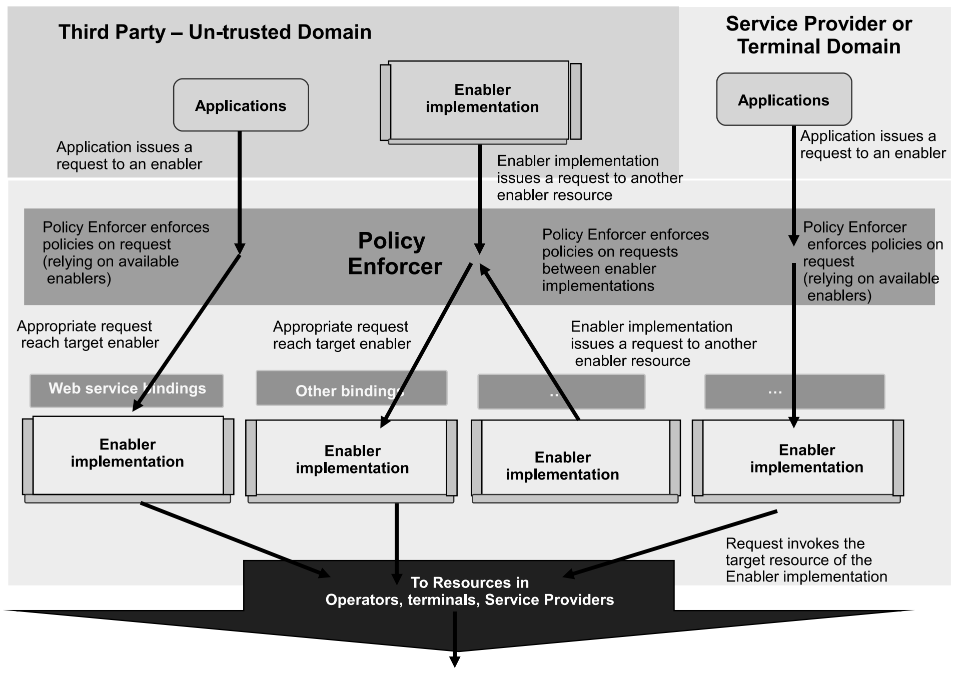

Third Party – Un-trusted Domain
Service Provider or Terminal Domain
Applications
Enabler implementation
Applications
Application issues a request to an enabler
Enabler implementation issues a request to another enabler resource
Application issues a request to an enabler
Policy Enforcer enforces policies on request (relying on available enablers)
Policy Enforcer
Policy Enforcer enforces policies on requests between enabler implementations
Policy Enforcer enforces policies on request (relying on available enablers)
Appropriate request reach target enabler
Appropriate request reach target enabler
Enabler implementation issues a request to another enabler resource
Web service bindings
Other bindings
...
...
Enabler implementation
Enabler implementation
Enabler implementation
Enabler implementation
Request invokes the target resource of the Enabler implementation
To Resources in Operators, terminals, Service Providers

Figure 2.10: OMA Service Environment Architecture

Basically, an OSE incorporates service enabler interfaces and translates service requests either directly into enabler logic or to an enabler specific protocol, in the case of IMS mostly SIP, Diameter and XCAP [R 07a] for XML document management. OMA, at the point of writing, does not standardise any mapping to a specific middle-ware messaging technology but leaves this open to the implementation of service environments. An enabler could as well be a non standardised implementation towards a specific telephony platform or an IN platform. An OSE consists according to OMA of several architectural elements that are described in the following:

- Enabler – A technology intended for use in the development, deployment or operation of a Service; defined in a specification, or group of specifications, published as a package by OMA. An enabler should specify one or more public interfaces. Examples of OMA enablers include Location or Presence.
- Enabler implementations – An element in the OSE representing an implementation of an enabler, e.g., either in a service provider domain or in a terminal domain. An enabler implementation can be viewed as a template that represents an implementation of any enabler (e.g., MMS or Presence) as defined by OMA.

- Enabler interface bindings – Interfaces must be specified in a neutral language manner. However, specifications may also define language specific bindings for the interfaces. Enabler interface bindings provide the specific formats (i.e. syntax and protocols used to access enablers using particular programming languages or network protocols).
- Policy Enforcer – An OSE architectural element that provides a policy-based management mechanism to protect resources from unauthorised requests and to manage the use of these requests.
- Applications – An implementation of a related set of functions that performs a certain application logic, often enabling one or more services.

Policy Evaluation, Enforcement and Management
The Policy Enforcer function of OSE or Policy Evaluation, Enforcement and Management (PEEM) as labelled within OMA standards can be used to intercept service requests and responses to and from a foreign domain as well as from any other service requester and apply certain rules (policies) on them that are stored at a repository. Furthermore, PEEM may be called explicitly by a resource for a specific policy evaluation result response to be executed as part of an enabler service logic. Policies in this context are formalisms used to express business, engineering or process criteria represented by a combination of policy conditions and actions. The following figure 2.11 provides an overview of the architecture of PEEM by OMA:

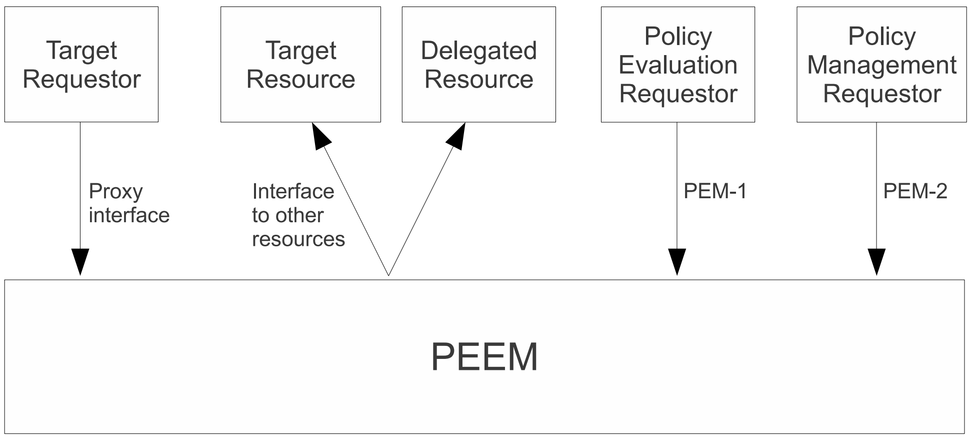

Target Requestor
Target Resource
Delegated Resource
Policy Evaluation Requestor
Policy Management Requestor
Proxy interface
Interface to other resources
PEM-1
PEM-2
PEEM

Figure 2.11: OMA Policy Evaluation, Enforcement and Management Architecture

PEEM supports two options for expressing policies:

- A rule set-based option: Each rule is evaluated as separate entity, and the combination of the results of the processing of all the rules in the rule set determines the policy outcome (notice that a precedence mechanism may be needed). The policy expression language specified is the XML Common Policy schema from IETF [STM+ 07].

- A workflow-based option: The entire policy is processed as a whole, following a flowchart approach, where at each node in the graph, a rule is being processed. The policy expression language specified is Web Services Business Process Execution Language (WSBPEL 2.0) defined by OASIS [OASIS 07].

2.2.3.3 OMA Service Provider Environment

The Open Mobile Alliance Service Provider Environment (OSPE) [OMA 05] is focused on standardising the service life-cycle management by addressing the main actors involved with application development, application deployment and application consumption:

1. End user
2. Service Provider
3. Application Developer

Independent deployment of services may lead to inconsistent user experience. From the *end user* perspective, the benefits of OSPE are supposed to be primarily that of improved service experience, i.e. improved services consistency and coherency. By re-using and leveraging components such as a common user-profile it is possible that individual services offered by a Service Provider will avoid, for example, the need for an end-user to repeatedly supply their personal service preferences and settings for each individual service that is offered by the Service Provider.

From a *service provider* perspective, OSPE is supposed to provide reduction in integration efforts for new services and facilitate integrating, deploying, managing, monitoring, upgrading, adding and removing components by standardising the components and their management interfaces. This should also encourage multi-vendor and hence component plug and play and component re-usability.

By standardizing the environment/infrastructure for developing and deploying services, OSPE is supposed to eliminate the need for the *application developer* to develop proprietary and operator specific mechanisms for the deployment of new services. This should be achieved by promoting modularity and re-use of components.

2.2.4 TeleManagement Forum

TeleManagement Forum (TMF) focuses on solving systems and operational management issues and is mostly known for its de-facto standards Business Process

Framework (eTOM), Information Framework (SID). Applications Framework (Telecom Application Map or TAM), and New Generation Operations Systems and Software (NGOSS) programme. The following subsections depict in more detail the emerging work of IPShpere and Service Delivery Framework related to the scope of this thesis.

2.2.4.1 IPSphere

The TeleManagement's Forum IPSphere working group works on specifications for inter- and intra-service provider working modes with a special focus on utilising service-specific management systems linked to service-specific network technology. The IPSphere service management approach focuses on extending existing frameworks to handle inter-provider and other multi-stakeholder interactions and defines a framework for properties as organizational roles, scope of business practices, service structuring, service & technology independent. The following figure 2.12 illustrates the functional scope of the IPSphere framework in more detail:

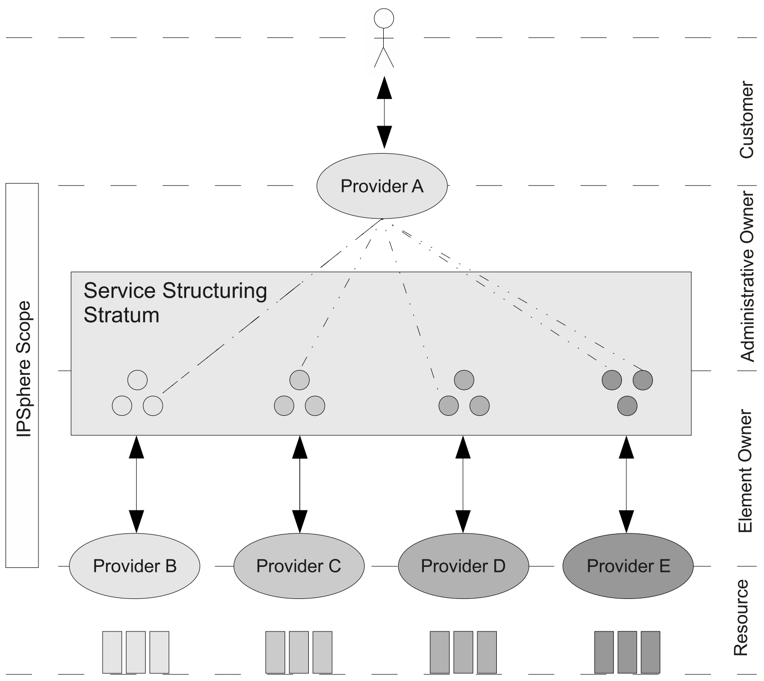
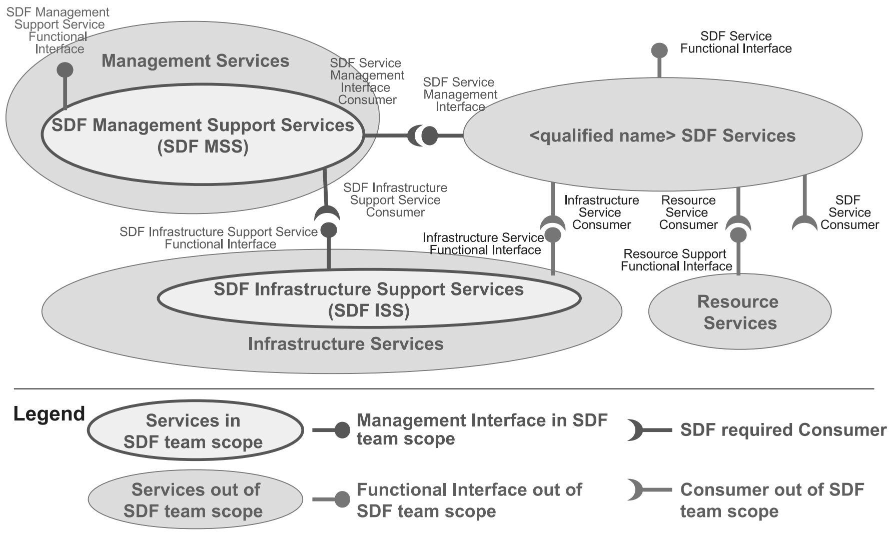


Figure 2.12: IPSphere Framework Context

The scope of the IPSphere Framework is based on two key aspects namely, service abstraction and service decomposition. The former is concerned with providing a mechanism to classify a service offer in terms of practical and business constraints and the latter with mapping a service offer to resource commitment across one or more providers. The framework is concerned with identifying the functional components required to facilitate service structuring within and across provider domains. It deals with defining an information model to capture the requirements of a structured service, providing flows for distributing the configuration information to the participating elements, so they can be configured to deliver the service, receiving fault and performance information from the underlying resources, and generating events to support auditing, notification, billing and settlement, etc. The *Service Structuring Stratum* is the focus of the IPsphere Framework and the place where service abstraction and decomposition takes place. It translates the abstract service requests to concrete resource commitments across providers and between providers and customers. The IPSphere framework is supposed to be responsible for defining interfaces to these systems and entities to facilitate participation in the Service Structuring Stratum and to identify the core functional components and interfaces that need to be standardised to enable systems, both within and across service, network, and content provider domains to collaborate in the delivery of services.

2.2.4.2 Service Delivery Framework

The goal of the TeleManagement's Forum Service Delivery Framework (SDF) programme [TMF 08] is to define a generic management framework for next generation services regardless of the software or network technologies used to implement those services. This management framework is aimed at addressing the full lifecycle of services. In order to scope the problem space, a SDF reference model is defined as depicted in the following figure 2.13:



SDF Management Support Service Functional Interface
Management Services
SDF Management Support Services (SDF MSS)
SDF Service Management Interface Consumer
SDF Service Management Interface
SDF Service Functional Interface
<qualified name> SDF Services
SDF Infrastructure Support Service Consumer
SDF Infrastructure Support Service Functional Interface
Infrastructure Service Consumer
Resource Service Consumer
SDF Service Consumer
Infrastructure Service Functional Interface
Resource Support Functional Interface
SDF Infrastructure Support Services (SDF ISS)
Infrastructure Services
Resource Services
Legend
Services in SDF team scope
Management Interface in SDF team scope
SDF required Consumer
Services out of SDF team scope
Functional Interface out of SDF team scope
Consumer out of SDF team scope

Figure 2.13: TM Forum Service Delivery Framework Reference Architecture

In the context of the SDF work, services are defined as components exposing their capabilities via one or more interfaces. A specific SDF compliant service exposes one ore more SDF Service Management Interface(s) (SMI). A SDF service may itself rely on capabilities exposed by other services, such as those provided by an integration infrastructure, by network or IT resources, or by other SDF services. The consumption of such external capabilities by the SDF service is graphically represented by a consumer (a “half moon”) in figure 2.13.

The IPSphere framework and its overlap with evolving SDF service life cycle management will be integrated in future SDF work.

2.3 Discussion

The evolution of Telecom technology has come a long way from POTS to digital networks as ISDN and the currently on-going paradigm shift from circuit-switched to packet-switched based Next Generation Networks. The result of this development is a change of paradigms on all layers. Nevertheless, the realisation of this evolution in standardisation can also be considered as a continuous effort by the classic Telecom industry to protect the network and network services domain from openness and therefore limit the amount of options for (non-Telecom) 3^{rd} parties to gain footage.

Whereas great achievements are visible regarding network technologies (e.g., fixed data networks and wireless communication networks), the service domain has remained rather static within the last decades concerning the evolution of new services and features. Interestingly, architectural principles defined initially for TINA-C are now applied on top of NGN service architectures as part of the standards definition by OMA and the TeleManagement Forum. Furthermore, we witness a similar technology shift in paradigms performed on the control layer happening on the service layer, too. Proprietary service platform with limited to none openness for developers and 3^{rd} parties are being replaced with platforms providing well-defined APIs towards network-specific functionality. Besides OSA/Parlay and Parlay X (now under the label of OMA NGSI) these standardization efforts take place within activities and fora not associated to the well-established Telecom standardization fora, but at organizations as the Java Community Process that have their roots in defining principles for service creation and execution in IT. A similar observation can be made regarding signalling protocols for the NGN where the Internet Engineering Task Force is the leading forum, compared to the ITU-T for circuit-switched network technology.

From the point of view of the *service developer* the evolution from proprietary service platforms requiring detailed knowledge of network protocols and platform specific tools for service creation is overdue. Technology and principles have been defined to allow cross-operator and cross-network development of services that can be exposed again to other services and developers for re-use. The current state in standardization and best practices may be considered as a well-defined eco space for service development and deployment compared to fragmented solutions with limited access realised for the IN [BT 08]. The definition and deployment of network abstraction APIs offer now for service developers access to program communication-centric services for Telecom networks ([BMS 07], [BMS+ 10]). Nevertheless, the amount of technologies, standards, and fora to be considered have significantly increased and impose a significant increase in complexity for the service developer.

From an *operator* perspective, the change in technologies has been significant, too. Monolithic vertical service/network silos realised for POTS and IN have been replaced by a well-defined horizontal NGN standard providing a set of distributed functions connected via specified protocol-based reference points. Most outstanding is the paradigm shift from circuit-switched signalling towards a packet-switched all-IP infrastructure. Adopting IT principles for service life-cycle management, the complexity of the service layer increased tremendously [BMS+ 09]. Non-deterministic behaviour of services is a challenging result of this evolution and clear guidelines for service creation and the management of once-deployed services are missing at the point of writing by standardization fora. Ironically, the definition of a new, well-structured network control and access architecture based on well-defined standards leads to a higher amount of complexity for network and service management. The openness of the service layer to expose network service functionality towards 3^{rd} parties via network abstraction allows from a business perspective new partnerships and possibilities for service integration. From a technical perspective the accompanying variety and complexity of technologies and standards on the one hand and on the other hand loosing control of service execution in the case of 3^{rd} party services impose high risks in operations for network operators.

From a *user's* point of view, the opening of the network allows the integration of communications features into a variety of service options. Nevertheless, this evolution has just started. Coming along the migration to all-IP for communications services, so called Over-The-Top (OTT) service provider already offer competing communications services to classic operator offerings, currently with a lower degree of Quality of Experience (QoE) for the end user and in some cases neglecting most of applicable standards described. Nevertheless, we expect communications features to be available for the end user in many service contexts through the availability of open APIs for developers ([BM 05], [BDM 07], [BLM 08], [BLM+ 09]). Still missing in 2010 and subject to research are tools and building blocks for users to create and configure their own services to meet their specific needs [BC 08].

Chapter 3

Service Principles in Information Technologies

The preceding chapter presented principles and standards enabling the technical inter-operation of distributed applications in telecommunications. One obvious notice when looking at the development of standards for service architectures in telecommunications in the last two decades is the adoption of principles from Information Technologies (IT) for the telecommunications service domain. This chapter deals with general principles of Service Oriented Architectures (SOA), composition of complex services, Model-Driven Engineering (MDE), and the formal expression and enforcement of service constraints during service creation and execution context in the context of this thesis.

We argue that this application of distributed service technologies and the service creation methodology in MDE is a novel application of existing technology for telecommunications that creates a robust, scalable, and flexible federation of principles that are needed in the next generation of large-scale SOA deployments to allow convergence of communications and information services.

3.1 Service Oriented Architecture Principles

A key concept of the SOA model is the publication and accessibility of services to create more complex, orchestrated services, and to allow the re-use of services through open access and well-defined service descriptions. The following sections describe open access and service invocation principles and technologies needed to realise a SOA.

3.1.1 Main Characteristics

According to the SOA reference model (SOA-RM) specification in [OASIS 06], SOA is a paradigm for organising and utilising distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. The SOA-RM specification bases its definition of SOA around the concept of "needs and capabilities", where SOA provides a mechanism for matching needs of service consumers with capabilities by service providers.

This concept is based on an architectural style that defines an interaction model between three primary parties:

1. a service provider, who publishes a service description and provides the implementation for the service,
2. a service consumer, who is either able to use the uniform resource identifier (URI) for the service description directly or to find the service description in a service registry and to bind and invoke the service,
3. a service broker providing and maintaining the service registry.

The service broker may define interaction patterns as depicted in the following diagram 3.1:

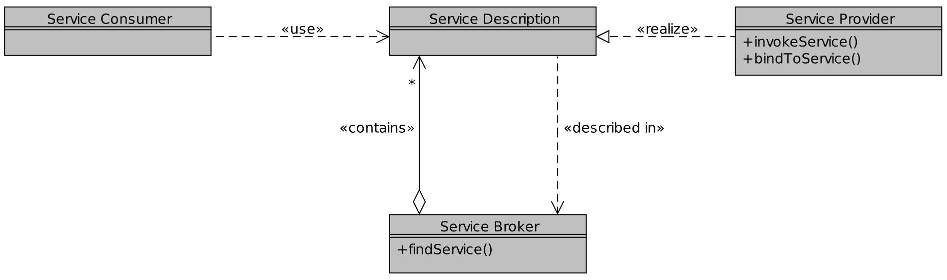


Figure 3.1: Conceptual model of a SOA architectural style

In the following are principal concepts that the reference model defines around services. Visibility, Interaction, and Real World Effect address the dynamic aspects of services (interactions with services), while the remaining concepts address static aspects:

- *Service Description*: The information needed in order to use, or consider using, a service. The purpose of description is to facilitate interaction and visibility, particularly when the participants are in different ownership domains, between participants in service interactions.

- *Visibility*: The capacity for those with needs and those with capabilities to be able to interact with each other. This is typically done by providing descriptions for such aspects as functions and technical requirements, related constraints and policies, and mechanisms for access or response.
- *Interaction*: Refers to the interaction between service providers and consumers. Typically mediated by the exchange of messages, an interaction proceeds through a series of information exchanges and invoked actions. The result of an interaction is a real world effect.
- *Real World Effect*: The actual result of using a service. This may be the return of information or the change in the state of entities (known or unknown) that are involved in the interaction.
- *Execution Context*: The set of technical and business elements that form a path between those with needs and those with capabilities and that permit service providers and consumers to interact. All interactions are grounded in a particular execution context, which permits service providers and consumers to interact and provides a decision point for any policies and contracts that may be in force.
- *Contract & Policy*: A policy represents some constraint or condition on the use, deployment or description of an owned entity as defined by any participant, while a contract represents an agreement by two or more parties.

As a SOA provides means for the definition of complex distributed services environments by a well-defined approach based on above service principles and interactions, the fragmented Telecom service domain consisting of proprietary platforms (e.g., IN SCPs) and open service platforms (e.g., SLEEs, SIP Servlet containers) as depicted in the previous chapter seems to be one among many Enterprise Architecture Integration (EAI) efforts to be solved by classical IT. Nevertheless, the following criteria differentiate from our view point the Telecom sector from other service sectors (e.g., enterprise):

- Services to be executed are network-centric services with low latency (e.g., call establishment, signal routing)
- Interactions between services are based on asynchronous behaviour (e.g., notifications of network events)
- Events in the network are communicated via signalling protocols (e.g., SIP) and need to be translated into platform specific APIs, resulting in multiple layers of translation between control and service layer
- Extreme high availability of services (e.g., 99.999%)
- Great variety of implementations for services (e.g., C++, Java, proprietary languages)

Taking into consideration that protocol adaptors and most legacy services have been “hard-wired” within network-centric platforms and these platform will only be replaced within an operator’s infrastructure incrementally, the transformation of the Telco service layer into a SOA and the following option of opening the operator’s service domain for 3^{rd} party access requires the implementation of a well-defined network abstraction layer allowing the communication between services within the service platform of an operator by a common communication principle to also allow the efficient exposure of those services.

3.1.2 SOA versus Web 2.0

The relationship between Web 2.0 and SOA has received an enormous amount of coverage through the notion of complexity-hiding and reuse, along with the concept of loosely coupling services. Some argue that Web 2.0 and SOAs have significantly different elements and thus can not be regarded as parallel philosophies [1]. Others, however, consider the two concepts as complementary and regard Web 2.0 as the global SOA.

Tim O’Reilly coined the term Web 2.0 to describe a quickly growing set of Web-based applications. SOA is considered the approach of encapsulating application logic in services with a uniformly defined interface and making these publicly available via discovery mechanisms as depicted above. O’Reilly identifies major characteristics inherent to the Web 2.0 philosophy: the Web is considered a platform for building systems that are “tied together by a set of protocols, open standards, and agreements for cooperation" [2]. The exploitation of collective intelligence of Web users, ownership of mission-critical data, and the end of the software release cycle are quoted as central characteristics as well. The use of lightweight programming models that allow for loosely coupled applications, the use of diverse media and devices for the consumption of Internet-based applications, and the realization of rich user experiences represent further paradigms inherent to the concept of Web 2.0.

According to [HMS+ 06], we classify Web 2.0 applications as follows:

- Communities that aim to unify their users by means of a common ideal such as social networking or knowledge sharing.
- Platforms or tools that help users create and share content with a broad audience (e.g., Web logs and online directories). Mash-up platforms let users retrieve content or functionality from arbitrary sources, mix it with other resources, and expose it for further reuse by other applications.

[1] see http://edgeperspectives.typepad.com/edge_perspectives/2006/04/soa_versus_web_.html

[2] T. O’Reilly, 2005, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

- Online collaboration tools support users in collaboratively performing certain tasks, such as maintaining time schedules or processing text online.

In comparison, the following are ways we can differentiate SOA use cases from Web 2.0:

1. SOAs allow for a cross-organizational integration of services. By adhering to common standards for the description of their service interfaces, corporations are enabled to setup loosely coupled electronic business transactions with other companies and thus automate business transactions in a quickly changeable fashion.
2. SOAs facilitate the intra-organizational integration of disparate services. On the basis of a central integration layer (often referred to as an Enterprise Service Bus / ESB), heterogeneous applications can be encapsulated and composed to a seamlessly integrated IT landscape
3. SOA-based application development significantly reduces development time thanks to the availability of reusable application building blocks.

The first major analogy between product design in the fields of Web 2.0 and SOA is the notion of reusing and composing existing resources. Both concepts let users reuse, compose, and enrich existing resources and components to new and potentially higher-level applications. The second commonness is the affinity to collaboration and coupling of remote resources or services. Both Web 2.0 and SOA applications enable the loose coupling of distant and possibly heterogeneous resources. A third similarity between Web 2.0 and SOA is the shared principle of agility and the support of permanent structural change. Web 2.0 and SOA also have divergent elements. First of all, many Web 2.0 applications incorporate a social aspect, as they facilitate human interaction and also mainly deal with human-readable content, such as text and pictures.

In contrast, conventional SOAs merely aim at interconnecting dispersed business functionality and facilitating seamless machine-machine collaboration. Web 2.0 is clearly about presentation and user interface integration, whereas SOA deployments are more abstract and less visible to its users. Furthermore, the degree of ex ante determination and involved governance is a key differentiator between Web 2.0 and SOA [M 05] . Because of their frequent implementation in the corporate context, SOAs are subject to requirements that do not necessarily exist in the case of most Web 2.0 applications and thus underlie well-defined governance mechanisms [WR 04].

In general, the philosophies of Web 2.0 and SOA serve different user needs and thus expose differences with respect to the design and used technologies of real-world applications. However, recently numerous novel use cases [SJ 07] demonstrate the great potential of combining the technologies and principles of Web 2.0

and SOA. One major example of the convergence of the two philosophies is the emergence of a global SOA that we refer to as Internet of Services (IoS) as depicted in figure 3.2.

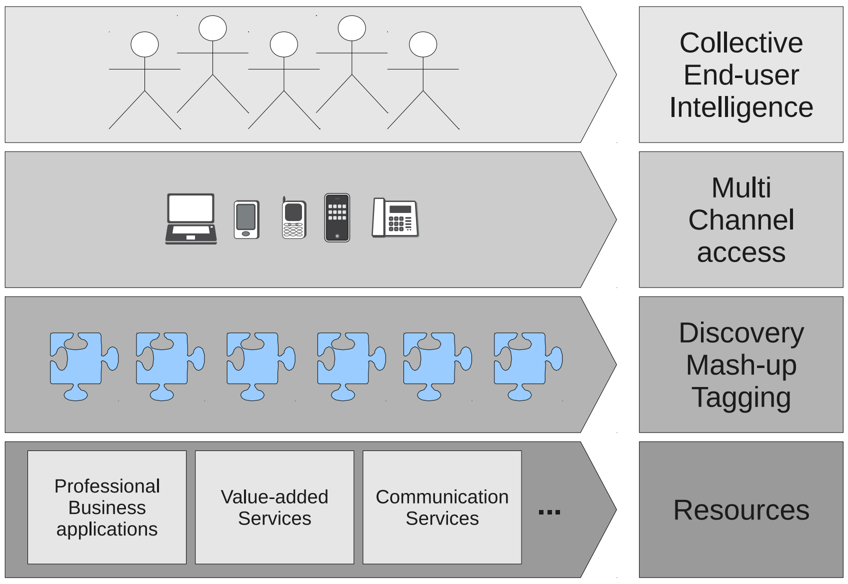

Collective End-user Intelligence
Multi Channel access
Discovery Mash-up Tagging
Professional Business applications
Value-added Services
Communication Services
...
Resources

Figure 3.2: Internet of Services basic architecture

Up to 2010, the average Internet users with little IT sophistication have not been able to easily retrieve and use certain services. This is because many of these services mostly reside within company boundaries and are only accessed for professional use in a corporate context. However, the provision of easily accessible services for end users might drive a novel generation of Internet use and allow for the ad-hoc setup and configuration of IT-supported business models.

3.1.3 Classification of Service Invocation Technologies

Service Oriented Architecture is found at multiple levels within software systems. This section examines the highest level of abstraction in software architecture, where the interactions among components are capable of being realised in network communication. In the following, we classify technologies related to distributed service principles in IT, namely Remote Procedure Call, the evolved Web Services standard and Representational State Transfer (REST) by applying criteria described in section 1.5.

3.1.3.1 XML-RPC and Web Services

XML-RPC is a remote procedure call protocol which uses eXtensible Markup Language (XML) to encode its calls and Hypertext Transfer Protocol (HTTP) as a transport mechanism. Figure 3.3 depicts the basic mechanism:

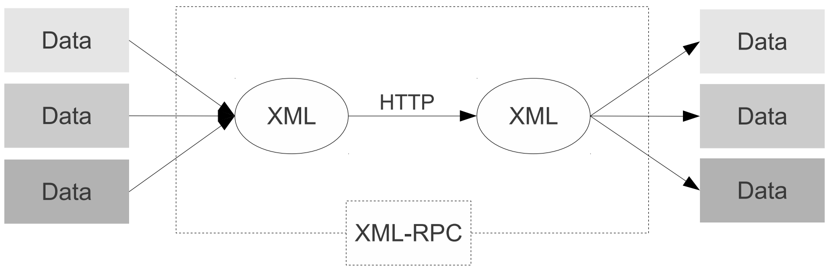


Figure 3.3: XML-RPC mechanism

As new functionality was introduced, the standard evolved into what is now SOAP [LJD 01]. SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in networks and is standardised by the World Wide Web Consortium (W3C) [W3C 07]. SOAP can form the foundation layer of a Web Services protocol stack, providing a basic messaging framework upon which Web Services can be built. The following figure 3.4 depicts the basic Web Service model:

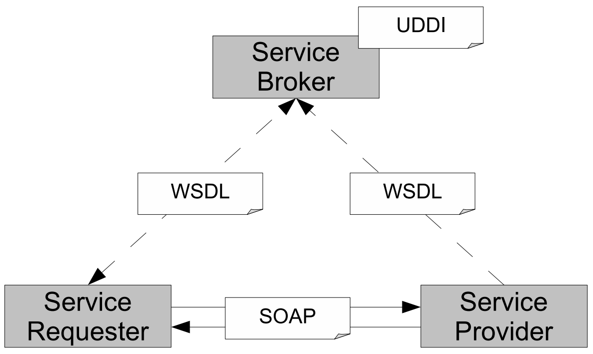


Figure 3.4: Basic Web Services model

3.1.3.2 REST and RESTful APIs

Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web (WWW). The terms

’representational state transfer’ and ’REST’ were introduced in [F 00] and they describe a method for building distributed services as web services in the WWW.

The following figure 3.5 illustrates some of the entities of REST and how they might participate within a simple enterprise system.

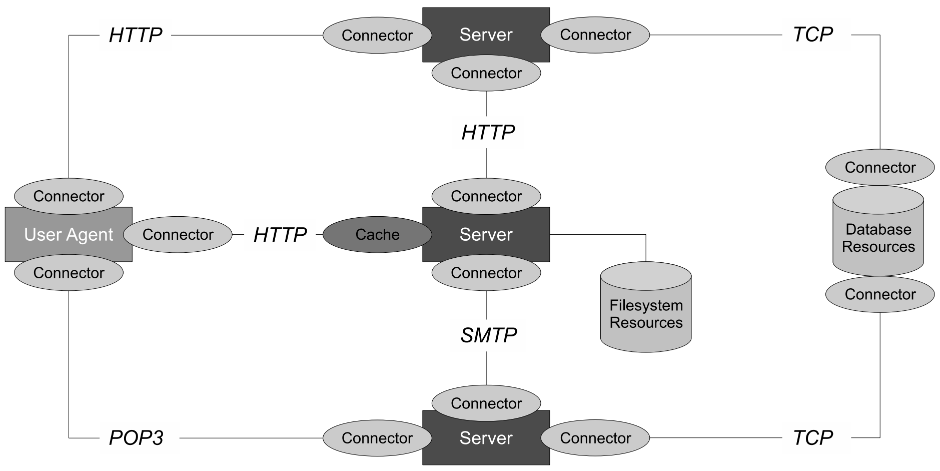


Figure 3.5: REST in a simple enterprise system

An essential concept in REST is the existence of resources (sources of specific information/functionality), each of which is referenced with a global identifier (e.g., a URI) in Hypertext Transfer Protocol (HTTP). In order to manipulate these resources, components of the network (user agents and origin servers) communicate via a standardised interface (e.g., HTTP) and exchange representations of these resources (the actual documents conveying the information). A RESTful web API is a simple web service implemented using HTTP and the principles of REST. Such a web service can be viewed as a collection of resources. In a system like the one above, connectors would be embodied as ports to enable communication among the components over each given protocol.

3.1.3.3 JSON-RPC

JSON-RPC is a remote procedure call protocol encoded in JavaScript Object Notation. JSON [C 06] is a lightweight computer data interchange format, text-based and human-readable for representing simple data structures and objects.

At the point of writing JSON-RPC is specified by the JSON-RPC Working Group[3]. It is a very simple protocol similar to XML-RPC, defining only few data

[3]http://groups.google.com/group/json-rpc/web/json-rpc-1-2-proposal

types and commands and in contrast to XML-RPC or SOAP, it allows for bidirectional communication between the service and the user agent client, treating each more like peers and allowing peers to call one another or send notifications to one another. Figure 3.6 depicts the basic mechanism:

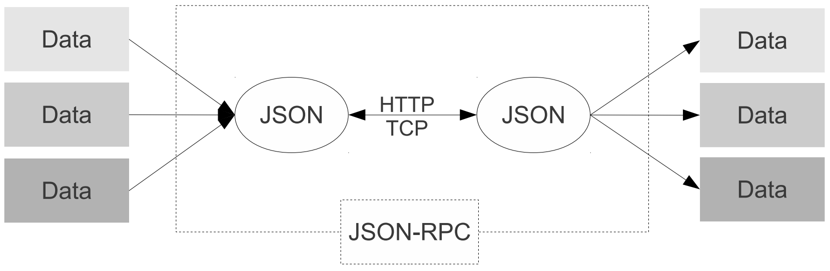


Figure 3.6: JSON-RPC mechanism

JSON-RPC allows multiple calls to be sent to a peer which may be answered out of order. Besides using HTTP for transport, one may use TCP/IP sockets to create much more responsive web applications with JSON-RPC, compared to polling data from a service with JSON-RPC over HTTP.

3.1.3.4 Classification

The following table 3.1 provides a classification of the above depicted service invocation methods according to the defined criteria in section 1.5:

	XML-RPC	Web Services	RESTful API	JSON-RPC
Configurability	+	+	++	+
Customizability	+	+	++	++
Modifiability	/	/	+	/
Network efficiency	-	+	+	++
Powerfulness	+	++	-	+
Scalability	+	++	++	+
Simplicity	/	-	+	++

Table 3.1: Classification of network-based application architectures and technologies

Configurability refers to the ability that a service might be re-configured during run-time without influencing clients and users. As REST is fully based on URIs and makes only use of the four verbs of HTTP, it is fairly easy to add new functionality through the definition of new URIs without influencing other services that make use of the providing service. Therefore it receives the highest benchmark.

Customizability in this context refers to the amount of effort a service developer needs to invest to change specific service behaviour. As all service request paradigms are considered as high-level network APIs compared to Common Object Request Broker Architecture (CORBA) or Java Remote Method Invocation (RMI), we consider all highly customisable through relatively less efforts. RESTful APIs and JSON-RPC are considered with a slightly higher degree, as data structures do not need to be defined in XML and therefore data parsing and consistency constraints are less complex.

Modifiability – As RESTful APIs do not incorporate complex data structures (usually XML or JSON transported within the HTML body) but resources based on URIs, it is fairly easy to modify underlying service logic by keeping resources consistent. Other APIs need to assure consistency in data, structured to achieve modifications not conflicting with remote service logic.

Network efficiency refers to amount and size of requests needed to achieve basic network-based distributed functionality. As stated above JSON-RPC allows the direct usage of TCP sockets and is not bound to high-level protocols as HTTP. Furthermore JSON objects are not XML encoded and require therefore less bandwidth as XML documents and XML tag overhead does not exist. The same applies for RESTful APIs but those do not support an active notification model as incorporated into JSON-RPC and follows the basic HTTP request/response paradigm as Web Services and XML-RPC do.

Powerfulness refers to the complexity of operations to be achieved by a request. In this regard, Web Services through the possibility of defining complex APIs as part of WSDL clearly receive the highest benchmark opposed to REST that is limited to the four HTTP verbs POST, GET, PUT and DELETE. Furthermore, as Web Services provide a well-defined description language (WSDL), automated generation of service end points is possible, too.

Scalability – RESTful APIs and Web Services scale well by design. In the case of RESTful APIs, it can be stated the more services are added to a distributed network architecture, the higher the degree of scalability. Also, considering that RESTful services end points usually make use of web servers as underlying service provider software, a cheap and scalable infrastructure can be achieved with rather few efforts.

Simplicity – RESTful APIs and JSON-RPC are definitely the most simple API constructs. The simplicity of JSON objects even adds on in favour of JSON-RPC and RESTful services that make use of JSON-based data structures. Web Services in contrast are more complex regarding the creation of services and software libraries required and also regarding the execution. JSON-RPC furthermore allows a very simple way of creating responsive web applications, compared to polling data from a service with XML-RPC or Web Services.

Concluding, it can be stated that RESTful APIs achieve through their simplicity a comparable high benchmark and have not without reason become the favourite way of achieving distributed network-based service architectures on the Internet. Considering the complexity in operations that is needed for advanced enterprise architectures and services, the efforts needed to achieve such functionality are rather high on the other hand and justify the usage of Web Services. Both XML-RPC and JSON-RPC may also be applied for RESTful services and a combination is advisable when more complex operations and data structures need to be incorporated. It should clearly be stated that Web Services based on SOAP define the most mature standard as the standard provides a well-defined communication pattern and service description language allowing automated generation of service end points.

3.2 Service Composition Principles

Service Oriented Architectures expose resources transparently as services. The mechanism of combining two or more (basic or composite) services into a complex service is known as service composition. The following subsections provide an overview of service composition principles and how they form the basis of business logic expression languages and modern service creation environments. We depict why composition mechanisms play an important role in converged Telecom and Internet service environment and provide the basis for modern service creation principles. We classify the most prominent composition/workflow expression languages according to their suitability in the context of this thesis.

3.2.1 Overview

The domain of service composition can be grouped into three main parts of characteristics differentiating how complex services are created: static, semi-automatic, and fully-automated service composition. Other characteristics are possible, but the complexity of the individual approaches can be roughly grouped based on the following characteristics for combining services:

- Combination of services manually in a fixed order as a static composition.
- Semi-automatic composition by the user with the help of tools
- Fully-automated composition of services, meaning that services are added or removed dynamically and can be aggregated to new complex value-added services. In this case the process of discovery and composition is automated and realized by a component performing the composition.

Generally, due to the openness of the Internet and the continuously growing amount of available services, dynamic service composition becomes a decision problem in regard of which services should be selected by optimizing the user's end-to-end QoS requirements. Considering the limited magnitude of telecommunications services within an operator's domain, selection problems can be reduced to an acceptable complexity by defining sets of applicable services for specific scenarios. Basically, this can be considered as an application of automated composition within a pre-defined space of action. This allows service providers and operators to adjust service behaviour according to users and their needs.

From an *operator* perspective service composition can be applied as a solution fulfilling the following targets:

- Application for multiple business concepts within one service scenario. E.g. one service might be composed with different features, depending on user-/service preferences, privacy or context information
- Selection of appropriate network services. E.g., selection of closest call control enabler to reduce interconnection costs in inter-domain or roaming scenarios, selection of alternative service enabler due to operations as maintenance down-time.
- Provisioning of service creation/composition tools to developers to leverage new service development for existing platforms.

From the point of view of a *service developer* an integrated service composition environment for service creation allows the definition of workflows and service logic on a very high abstraction level without touching or altering the actual code of service features. This provides on the one hand tools and mechanisms to non-experts hiding non-relevant programming support details. On the other hand facilitating the rapid creation of new communication services should leverage the magnitude of available services for further re-use in complex compositions and to end users.

From a *user's* point of view the combination of the above described targets is expected to provide a greater variety of services and a higher degree of personalisation of features and service behaviour.

The following subsection provides a general introduction into the principles of service workflows and composition.

3.2.2 A Generic Service Composition Model

The following Figure 3.7 illustrates the components of a generic service composition reference model defined in [ILeM 09]. The major components which can be identified are: Translator, Generator, Evaluator and Builder.

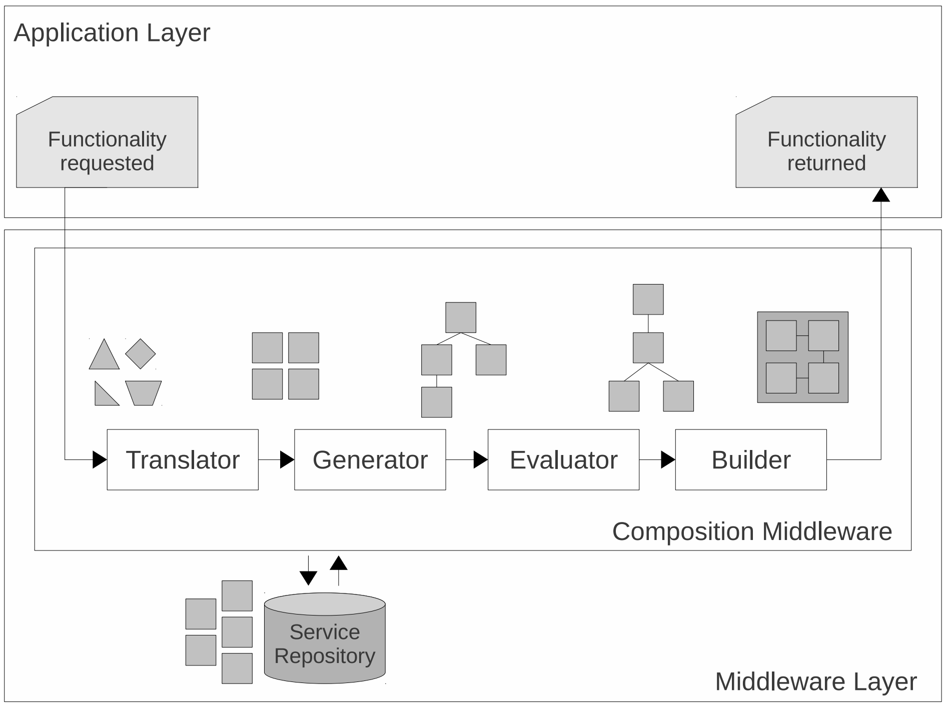


Figure 3.7: Service Composition Model

The composition middle-ware interacts with the application layer by receiving functionality requests from users or applications. It needs to respond to the functionality requests by providing services that fulfil the demand, these services can be atomic or composite. The Service Repository represents all the distributed service repositories where services are registered. The composition middle-ware interacts with the Service Repository to choose services to compose. Focusing on synthesis, mechanisms range from static service compositions to semi-automatic to fully-automated service compositions; static and automatic composition approaches are compared in [DS 05]. Whereas static service composition requires manual generation of choreographies (similar to workflows), fully-automated service compositions are capable of discovering components and synthesizing composite services at run-time.

The process of service composition includes the following phases:

1. **Translation Phase**
   Applications specify their needed functionalities by sending requests to the middle-ware. These requests may be described by diverse languages or techniques. Most systems distinguish between external specification languages and internal ones. The external ones are used to enhance the accessibility with the outside world, commonly users or other services. Internal specific-

ation corresponds to a formal way of expressing things and uses specific languages, models, and logics, usually for SOA a generic service model. [M 07] provides a translation mechanism of the available service technologies and service descriptions into one model. Authors in [CFB 04] propose a wrapper to provide a uniform access interface to services. These middleware usually realize transformation from one model to another or from one technology to another. If new technology models appear in the environment, the *Translator* will need to be expanded to take these technologies into consideration. Other approaches ([CJF+ 05], [SHP 03]) abandon the Translator module as they use a common model to describe all the services of the environment. They use common description languages based on semantics as OWL-S [OSC 03] for describing atomic services, composed services and user queries.

2. **Generation Phase**
   Once translated, the request specification is sent to the Generator. It will try to provide the needed functionalities by composing the available services and functionalities and generates one or several composition plans with the same or different services available in the environment. Composing service is technically performed by chaining interfaces using a syntactically or semantically method matching. The interface chaining is usually represented as a graph or described with a specific language, representing the semantic matching between the input and output parameters [FS 05]. A number of languages have been proposed to describe data structure in general and functionalities offered by devices in particular. While some languages are widely used, such as XML, and generic for multiple use cases, others are more specific to certain tasks as service composition, orchestration or choreography such as Business Process Execution Language (BPEL) [OASIS 07] and State Chart XML (SCXML) [W3C 09].

3. **Evaluation Phase**
   The Evaluator chooses the most suitable composition plan for a given context. This selection is done from all plans provided. The evaluation depends strongly on many criteria like the application context, the service technology model, the quality of the network, the non-functional service QoS properties, etc. Two main approaches are commonly used: rule-based planning ([CIJ+ 00], [GPZ 04], [PF 02]) and a formal methods approach ([KKS 07], [MBE 03], [M 07]. The first one evaluates whether a given composition plan is appropriate or not in the actual context by applying rules. A major problem of the rule-based evaluation approach is the lack of tools verifying the correctness (functional and non-functional aspects) of the composition plan. This aspect is at the main advantage of a formal method approach.

4. **Build Phase**
   The Builder executes the selected composition plan and produces an implementation corresponding to the required composite service. It may apply a

range of techniques to realize the effective service composition, depending strongly on the service technology model to be composed and on the context. Once the composite service is available, it may be executed by the application that requires its functionality. Literature distinguishes different kinds of builders provided by the service composition middle-ware. Some builders are very basic and use only simple invocation in sequence to a list of services [ILeM 09]. These services need to be available otherwise the composition result is not certain. Others [U 00] provide complex discovery protocols adapted to the heterogeneous nature of the pervasive environments.

3.2.3 Classification of Composition Expression Languages

The generic service composition model presented in the previous section defines a model applicable for manual to fully-automated approaches. The *Generator* has the task to create a workflow of services and features that can be directly executed or compiled to machine-executable code. Within in the last decade XML-based languages for expressing such workflows of compositions, orchestrations or choreographies of services have been standardized to allow the execution of such services across multiple platforms. This section outlines the most prominent approaches in the context of this thesis to enable complex services including communications features. It provides a qualitative classification of different languages based on our experiences during the last years and a comparative literature study.

3.2.3.1 Business Process Execution Language

BPEL, short for Web Services Business Process Execution Language (WS-BPEL) was standardised in 2004 by OASIS after collaborative efforts to create the language by BEA Systems, IBM, Microsoft, SAP and Siebel Systems. We refer to the latest version WS-BPEL 2.0 [OASIS 07], released in January 2007.

It is a language that combines and replaces IBM's WSFL (Web Services Flow Language) and Microsoft's XLANG specification, relying on WSDL, XML schema and XPath. Basically, it is an executable XML-based language, for specifying interactions between Web Services, which can be executable business processes and abstract business processes. The first category refers to the behaviour of a participant in a business interaction, while abstract processes represent some partially specified services that are not intended to be executed, but serve a descriptive role. BPEL supports the specification of an executable process, the message exchange with other systems involved, and finally the coordination and management of the entire process. A BPEL process provides a Web Services (SOAP) interface, but behind the scenes it calls other Web Services to actually realise service specific functionality; those services are called partners. The relationship between them is called a partner link and it is defined by operations that each partner provides to the other. Thus, BPEL orchestrates Web Services by specifying the order in which it is meaningful to call a collection of services, and assigns responsibilities for each of the services to partners. [M 03] The following figure 3.8 depicts the general BPEL mechanism:

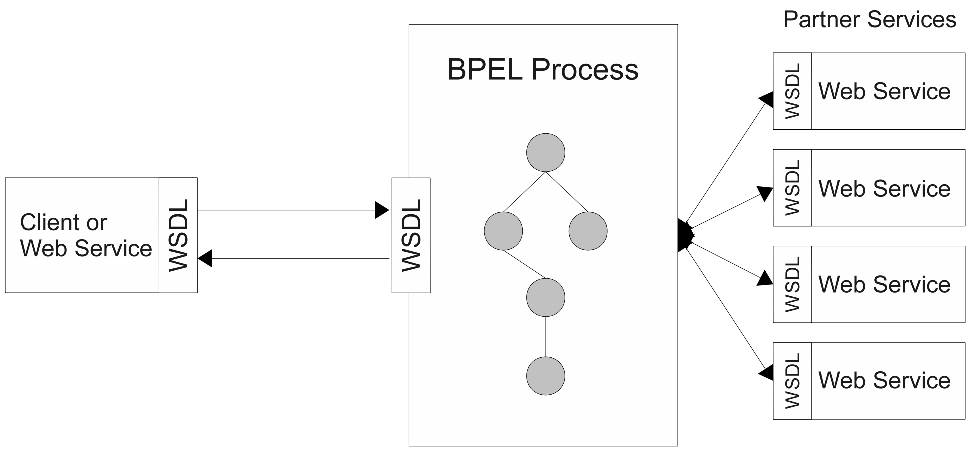


Figure 3.8: General BPEL mechanism

The design of Web Services, including BPEL processes, generally follows a number of best practices for SOA ([S 06], [RA 04]). A well designed service should expose a coarse-grained interface to the functionality it supports. Client-service communication should also be minimized by exchanging coarse-grained data types and doing composite tasks together where possible. This helps to keep response time within acceptable limits, what is an essential feature in the context of near real-time services like call control, since Web service invocations tend to have more overhead due to underlying tasks such as encoding and decoding of request and response messages in XML, and logging and monitoring for error recovery [B 07].

3.2.3.2 Call Control XML and Voice XML

The Voice eXtensible Markup Language (VoiceXML) [W3C 07a] and Call Control eXtensible Markup Language (CCXML) [W3C 07b], are XML-based although separate frameworks, that have been proposed by W3C and complement each other.

CCXML is an XML-based language for marshalling telephony servers such as dialogue systems and conference bridges to service telephone calls. Since the actual work is performed by external servers, this can be seen as a domain specific form of composition. The motif in CCXML is that during the course of a call, a number of objects are connected together. These objects raise events; a CCXML script reconfigures the connections in response. In contrast, the motif in BPEL is that of an assembly line: a BPEL script moves a request through a sequence of steps. CCXML can be used with a dialogue system such as VoiceXML, but the call control model in CCXML has been designed to be sufficiently abstract so that it is able to accommodate all major definitions including Parlay or JAIN SLEE. CCXML is capable of receiving events and messages from external computational entities (e.g., via HTTP POST), interacting with an outside call queue, or placing calls on behalf of a document server. Call legs are considered as audio sinks and sources which can be combined to form arbitrary networks, in support of sophisticated conference call features.

VoiceXML supports the creation of Interactive Voice Response (IVR) services. It is designed for creating audio dialogues that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed initiative conversations. A VoiceXML document/script specifies each interaction dialogue to be conducted by a VoiceXML interpreter. User input affects dialogue interpretation and is collected into requests submitted back to the Web server. The latter replies with another VoiceXML document to continue the user's session with other dialogues. From a design perspective, VoiceXML minimizes client/server interactions by specifying multiple interactions per document. It separates user interaction code (in VoiceXML) from service logic (e.g., in CGI scripts) and it promotes service portability across implementation platforms. There are however several issues with this language when used for IVR applications. For instance, when working as a call controller, VoiceXML offers limited advanced telephony functions and called parties cannot be placed in an IVR. [AS 07]

3.2.3.3 State Chart XML

State Chart XML (SCXML) is an XML-based markup language which provides a generic state-machine based execution environment based on Harel state charts [H 87]. At the point of writing it is a working draft at W3C [W3C 09].

SCXML is able to describe complex state-machines; it is possible to describe notations such as sub-states, parallel states, synchronisation, or concurrency. The objective of this standard is to generalise state diagram notations which are already used in other XML contexts as CCXML or VoiceXML. As a multi-modal control language combining VoiceXML 3.0 dialogues with dialogues in other modalities including keyboard and mouse, ink, vision, haptics, etc., it may also control combined modalities such as lipreading (combined speech recognition and vision) speech input with keyboard as fallback, and multiple keyboards for multi-user editing. The following figure 3.9 illustrates the interaction of SCXML with CCXML and VoiceXML:

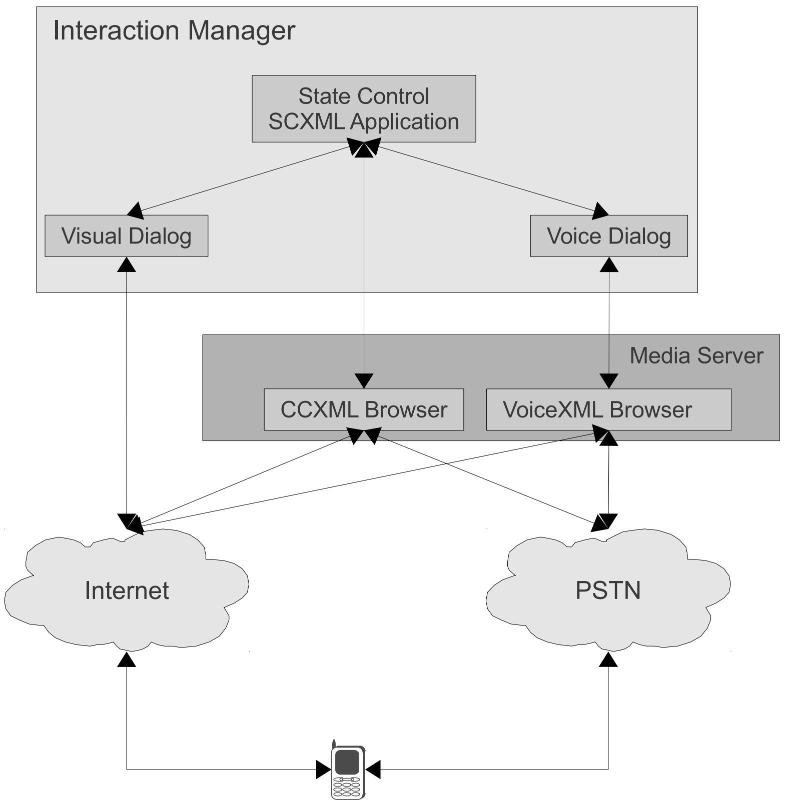


Figure 3.9: SCXML / CCXML / VoiceXML Interaction

Even if SCXML has its roots in CCXML and will be adapted into the upcoming VoiceXML standard, it offers a generic and light weight approach for expressing generic work-flows independent of protocols and programming languages.

3.2.3.4 Service Logic Graph

The OASIS Open Composite Services Architecture (Open CSA) defines an independent programming model for SOA-based systems. In this model functionality is designed as a set of services which may be tied together to create new combined services with additional value. CSA was first published as Service Component Architecture (SCA) [OSOA 09] in March 2007 by the Open Service Oriented Architecture (OSOA) industry initiative. SCA v1.0 was handed over to OASIS to become a formal industry standard. Open CSA provides a model for the composition of services, creation of service components, and re-use of existing application functions within SCA composites. It aims at the support of a wide range of technologies for service components and for the access methods which are used to connect them. According to SOA principles the offered services in Open CSA are independent from the underlying implementation technologies.

Service Logic Graphs (SLGs) are a behavioural counterpart to SCA composites and they are architecturally compliant with the coming standard for Service Composition [JMN+ 08]. SLGs provide a model for the orchestration of Service Independent Building Blocks (SIB), an analogy to the original naming of elementary telecommunication services [MBD 08]. SLGs represent the underlying model of jABC [JKN+ 06], a flexible framework that supports the whole lifecycle of a business process. It can be used by business and application experts to graphically orchestrate complex end-to-end business processes into running applications on the basis of a service library. SLGs can be canonically wrapped into (graph-) SIBs to allow for a hierarchical organization of complex process models. Moreover, process models, which follow a certain standard defined by jABC, can be directly exported into (partial or complete) stand-alone applications, a feature which turns jABC from a modelling into a development tool. Finally, there are SIBs, which serve as wrappers for outside functionality (e.g., non-Java applications such as C++, C#, SOAP/WSDL Web services, REST); this enables modeling and building heterogeneous, distributed, applications [MS 09].

3.2.3.5 Classification

The following table 3.2 provides our classification of the above depicted models and languages according to the defined criteria in section 1.5 as applicable:

	BPEL	CCXML	SCXML	SLG	VoiceXML
Configurability	++	-	++	++	-
Customisability	/	/	/	-	/
Modifiability	+	+	+	-	+
Network efficiency	-	- -	++	++	- -
Powerfulness	-	/	++	+	/
Scalability	+	- -	++	++	- -
Simplicity	/	++	-	-	++

Table 3.2: Classification of process orchestration languages and models

Configurability is related to both extensibility and re-usability in that it refers to modification of components, or configurations of components, such that they are capable of using a new service or data element type. BPEL, SLG, and SCXML provide constructs for modelling and executing abstract workflows that may either be re-used or modified for alternative usage. CCXML and VoiceXML are considered as domain- and feature-specific and in this sense limited regarding re-usability and especially extensibility.

Customisability in this context refers to the ability to temporarily manipulate the behaviour of a service either manually by changing the code or through environment variables without the need to re-deploy the service into a framework. As BPEL, CCXML, VoiceXML, and SCXML are all executable languages, manipulation of the workflow is possible through altering the orchestration script directly. Therefore, there is no distinction between those in this context. Services based on SLGs are provided as models within the jABC environment and require redeployment to the service platform when services are not executed directly within jABC.

Modifiability refers to the effort needed with which a change can be made to an existing work-flow. It has to be differentiated between the effort needed to manipulate code directly (e.g., by using a text editor) or by using a SCE that provides graphical representation and allows manipulation via such an interface. In the first case rather simple languages, especially VoiceXML and CCXML, but also SCXML and BPEL allow direct manipulation of code but graphical environments are available, too. SLG, as stated above requires the usage of graphical editors for those users that are not intimately familiar with the language constructs and the internal functionality of the SCEs. With the understanding that it is easier for non-experts to manipulate workflows through graphical interfaces, we still rate these languages higher that provide the possibility to edit code with few efforts in non-graphical editors as well.

Network efficiency comprises in this context the ability to integrate different remote service endpoints by means of variety of protocols and service technologies. VoiceXML and CCXML allow only the integration of services that are provided by the interpreter and the platform. BPEL allows the integration of distributed Web Services but only through SOAP in a homogeneous Web Services environment. SCXML provides mechanisms to include hybrid services into a work-flow when provided by the interpreter that executes the workflow script. jABC creates executables that may incorporate any kind of remote service but requires the implementation of a jABC/SLG specific service/protocol adaptor. Therefore, SCXML and SLG are rated highest.

Powerfulness refers to strength of expression of an orchestration but also to additional orchestration-specific application logic that can be included. SLG and SCXML are very powerful languages allowing the definition of complex workflows. SCXML provides furthermore the option to include script-like languages (e.g., JavaScript) into workflow definitions directly for implementing logic within the workflow definition. CCXML and VoiceXML are powerful languages but only within a limited scope of managing call and voice related services. BPEL provides no standardised mechanism for the implementation of application-specific logic but only means for defining delegations to other Web Services and receives therefore the lowest ranking.

Scalability in this context is similar to *Network efficiency* and refers to the ability to support a large number of services across various (distributed) platforms.

Generally, all composition languages that allow the inclusion of remote services to form a distributed service mash-up scale in this context better than languages that may only invoke local services. Therefore, CCXML and VoiceXML receive the lowest benchmark as services may only be locally executed. As stated above BPEL allows only the integration of Web Services and scales therefore merely in a homogeneous Web Services domain. All other languages provide mechanisms to integrate remote services with various APIs and receive high benchmarks.

Simplicity refers to the amount of information, complexity, and knowledge needed by the developer to express a specific behaviour and is unrelated to specific graphical representation forms inside an SCE. SLG and SCXML are the most complex languages, providing at the same time the widest scope in usage, they are therefore rated rather low. BPEL is considered as human-readable and -codable, but is due to its generic usage approach still a rather complex languages. CCXML and VoiceXML are considered to be the most simple languages.

Concluding, we may state that each composition language has its use case and platform specific pros and cons and there is no perfect language that combines business process related orchestrations that are rather data-driven and complex communications services that are asynchronous and event-driven. Nevertheless, SCXML has a good potential to become, once finally standardised, a candidate for a multimodal control language that offers backward compatibility to CCXML call flows and VoiceXML interactions and it may also control database access and business logic modules. BPEL and SLG offer the integration of multiple remote service endpoints to create integrated complex orchestrations. BPEL has one main drawback, for our consideration, that it only allows the integration of SOAP-based Web Services, other kind of service technologies need a special Web Services adaptor to be integrated into a BPEL work-flow.

3.3 Model-Driven Engineering

The previous two sections provide a general introduction to Service Oriented Architectures and the underlying principles of distributed services and service composition. This section focuses on a SOA-aligned software design approach for the development of software systems by the definition of models allowing a formal description of state machines and automated code production based on Model-Driven Engineering (MDE). MDE has been promoted as a solution to handle the complexity of large-scale software development by raising the abstraction level and automating labour-intensive and error-prone tasks. Related to the scope of this thesis, we consider MDE as an enabling option for Service Creation Environments for rich Telecom services (either from a 3^{rd} party developer perspective or for operator internal service development) allowing formal verification of software to reduce errors in hybrid, distributed service scenarios that are difficult to control. MDE has been hailed as the solution to handle the key problem of increasing complexity facing the software development industry by:

1. providing better abstraction techniques and
2. facilitating automation.

By switching to a MDE approach, businesses are promised to reap benefits through increased productivity and software quality [UP 07].

The following subsections depict briefly relevant standards and principles and discuss the strength and weakness of MDE. We outline several examples providing a detailed overview of industry's experiences with MDE in telecommunications.

3.3.1 Base Standards and Technologies

Software development in emerging domains and rapidly changing environments raises increasing demands to the quality and automation degree of applied development processes. Model-Driven Software Development (MDSD) is a key technology to enhance these processes significantly. Well-defined models replace programming code and text documents as primary development artefacts. MDE promotes to apply model transformations to bridge the gaps between models which contain the development information resulting from the execution of different development activities.

In [B 04], authors define MDSD as a multi-paradigm approach that embraces domain analysis, meta modelling, model-driven generation, template languages, domain-driven framework design, the principles for agile software development, and the development and use of open source infrastructure. In MDSD, aspects from popular mainstream approaches that can scale to large-scale industrialized software development are combined with less well-known techniques that are needed to prevent architectural degradation in large systems. Additonally, it provides techniques to automate the repetitive aspects of software development. To apply MDSD, we consider an existing implementation or reference implementation. It consists of unique code parts with individual structure. When analysing this unique code, it can be structured into three main parts:

1. a *generic part* that is identical for all (future) applications,
2. a *schematic part* that is not necessarily identical for each application, but uses the same taxonomy (e.g., based on the same design patterns),
3. an *application specific part* that is not generalisable.

The following figure 3.10 illustrates this separation in the context of application development using MDE:

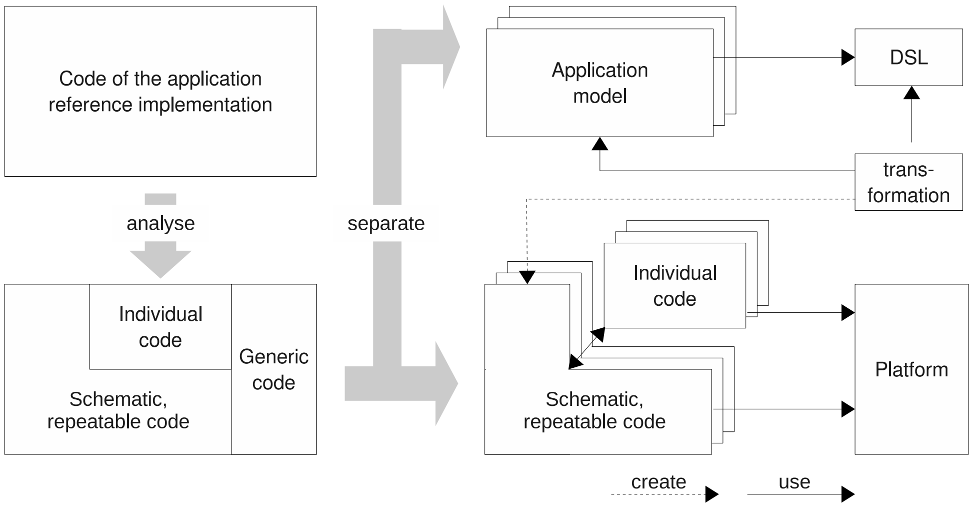


Figure 3.10: Basic Idea Model-Driven Software Development

MDSD aims at deducing the schematic part for all (sub-)applications from the same model. Intermediate steps might be possible during transformation, nevertheless the key elements are Domain-Specific Languages (DSL), transformation, and platforms. Those have to be created only once for a specific domain.

One of the important standardized approaches of MDSD based on abstraction of platform similarities is the Object Management Group's (OMG) Model Driven Architecture (MDA)[4]. It focuses on forward engineering, meaning to produce code from abstract models. The MDA approach defines through the Computation Independent Model (CIM), a colloquial system description, the transformation into a Platform-Independent Model (PIM) using an appropriate Domain-Specific Language (DSL). Then, given a Platform Definition Model (PDM) corresponding to CORBA, .NET, the Web, etc., the PIM is translated to one or more Platform-Specific Models (PSMs) that are executable. The PSM may use different DSLs, or a general purpose language like Java, C#, PHP, Python, etc. Automated tools generally perform this translation [KMW 04]. The base PIM only expresses business functionality and behaviour, it expresses business rules and functionality independent of technology. The benefit lies in its technological independence, the base PIM retains its full value over the years, requiring change only when business conditions mandate. The general division of the modelling levels can be depicted as in the following:

[4]http://www.omg.org/mda/

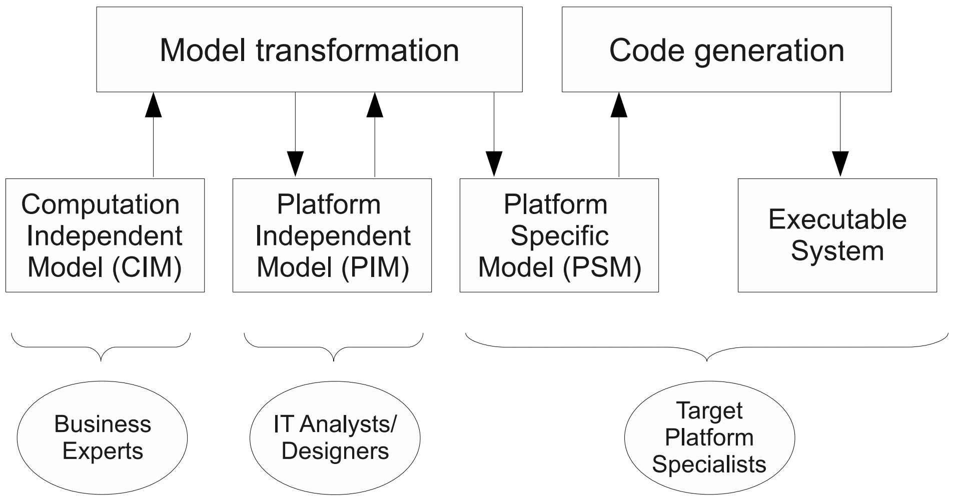

Model transformation
Code generation
Computation Independent Model (CIM)
Platform Independent Model (PIM)
Platform Specific Model (PSM)
Executable System
Business Experts
IT Analysts/ Designers
Target Platform Specialists

Figure 3.11: Service Specification Levels in OMG's MDA

Once the first iteration of the PIM is completed, it is stored in the MOF as the input to the mapping step that will produce a PSM. Specialisations and extensions to UML allow to express both PIMs and PSMs. During the mapping step, the run-time characteristics and configuration information that are designed into the application model in a general way are converted to the specific forms required by the target middle-ware platform. Guided by an OMG-standard mapping, automated tools perform as much of this conversion as possible, flagging ambiguous portions for programming staff to resolve by hand. Early versions of MDA required considerable hand adjustment.

The use of the Unified Modeling Language (UML) [OMG 97] provides the capability to describe static invariants and pre/post conditions that are particularly important features of an approach to software engineering called contract-based design. UML allows formalisation of the vocabulary otherwise left imprecise in interface specifications, as an abstract, yet precise model of the state of the object providing that interface and of any parameters exchanged. Some current OMG specifications including UML, MOF (Meta-Object Facility), and CWM (Common Warehouse Metamodel) specifications already use UML and OCL (Object Constraint Language) for specifying constraints. [OMG 08]

The life-cycle of an application may vary dramatically depending on whether it is being used to build a new application from scratch or just to add a wrapper to an existing application. MDA supports many of the commonly used steps in MDSD and deployment. A key aspect of MDA is that it addresses the complete life-cycle covering analysis and design, programming (testing, component build or component assembly) and deployment and management. [SFH+ 04]

3.3.2 Strength and Weaknesses

There is some debate among software developers about how useful code generation as such is or should be [MV 08]. This certainly depends on the specific problem domain and to which extent code generation should be applied. There are well-known areas where code generation is an established practice and not limited to the field of UML. Even though many promises are made that MDE provides a cost-efficient way to address complexity in software development and reducing the error rate of software, these are in most cases poorly, if at all, supported by evidence. This section outlines strengths and weaknesses of MDE applied in specific sectors based on a literature review. It should be noted that generally success cases are more likely to be published than failures.

A broad range of companies in various domains report their experience from investigating or applying MDE. The investigated papers cover the following sectors:

- Telecommunications domain ([BLW 05], [PB 05], [UP 07], [WW 06]),
- Business applications and financial organizations ([DLT+ 03], [SKS 07]),
- Defense / aerodynamics / avionic systems ([B 05], [JN 04]),
- Web applications ([BCF+ 05], [M 03]).

We consider three main criteria to evaluate MDE:

1. **Context and motivation**
   MDE is applied in a wide range of domains as listed above, including safety-critical systems and product lines. MDE is assumed to lead to higher productivity (by increased automation in the development process), increased standardization and formalisation, and improved communication within development teams and with external stakeholders. Labour-intensive and error-prone development tasks are automated and best-known solutions can be integrated in code generators, resulting in reducing defects and improving software quality.
2. **Maturity level of MDE**
   The current state of MDE is far from mature. There is a varying degree of automation and it is mostly applied for code generation. Tools are improved during the recent years but several papers still discuss the lack of a coherent MDE environment and tool chain. Tools should scale to large scale development and support the domain-specific approach more effectively. Software processes should also be adapted to MDE. Other challenges in adopting MDE are the complexity of modelling itself, developing PIMs that are portable to several platforms and using MDE together with legacy systems.

3. **MDE impact on productivity and software quality**
   Quantitative evidence on productivity gains in the area of telecommunications are mentioned in [BLW 05] and [WW 06]. Those are the only ones providing some quantitative data on software quality improvements. Software quality benefits are discussed in several papers but are not backed up with data.

The following table 3.3 provides an overview contrasting the arguments for strengths and weaknesses of MDE based on experiences by Motorola[5] in [BLW 05]:

Strength	Weakness
Reduction of overall cost of quality due to a decrease in inspection and testing times	Lack of common tools and interoperability between modelling tools
2-8x productivity improvement when measured in terms of equivalent source lines of code	Lack of well-defined semantics in models
2-3x reduction in effort through the use of co-simulation, automatic code generation, and model testing	Poor performances of tools and generated code
	Missing well-defined processes for team-based MDSD
	Lack of migration tools for transforming existing development processes to MDE

Table 3.3: Strengths and Weaknesses of MDE

Generally spoken, modelling should be easier and faster than code writing to be able to differentiate clearly between strengths and weaknesses of MDE. Appropriate tools and processes and increased expertise on modelling are areas for improvement in most cases. Combining MDE with domain-specific approaches and in-house developed tools has played a key role in successful adoption of the approach. One of the promises of MDE in increasing portability of solutions to multiple platforms has not often been feasible, mainly due to the fact that tools are bound to specific platforms. Interesting is the quantification of savings as a clear strength, once a development process is migrated to MDE end-to-end.

[5]http://www.motorola.com

3.3.3 Strategies for applying MDE in Telecommunications

The previous section lists a set of key sectors in IT that apply MDE for the software creation process. But MDA as a common software development architecture does neither define any methodology and guideline aiming at specific domains (such as telecommunications), nor does it provide a concrete definition for “platforms”. Taking into consideration principles and technologies described in chapter 2, a service platform for (tele-)communications services can be considered from two perspectives:

(1) A technical view on the platform providing and implementing open APIs as OSA/Parlay, OMA NGSI, SIP Servlet or JAIN SLEE.

(2) An implementation view referring to platform specific technologies as Java, J2EE and Enterprise Java Beans (EJB), Web Services, and REST.

Whereas (2) can be considered as domain independent from a service perspective, from a platform view telecommunications has similar requirements as platforms in the finance or defence sector regarding availability, resilience, and real-time behaviour.

3.3.3.1 Open API-based Meta-model Approach

The model-driven service creation approach adopts concern-separated design methods. The horizontal level involves meta-model/UML profile layer, model transformer layer, and service model layer. The meta-model layer is used to define the structure and semantics of the model layer and can be considered as the instantiation of the model layer [QLL 06]. Following this approach, the meta-model layer consists of a Telecom service domain meta-model, an open API-based meta-model, and an implementation-specific platform meta-model.

The Telecom service domain meta-model is a high-level abstraction of service concepts and classification, service characteristics, and behaviour, etc. This domain meta-model is independent of concrete open API technologies, so it can be used to construct a high-level service logic model of the model layer.

The open API-based meta-model depicts the concrete characteristics of open API technologies. By applying a concrete API-related meta-model, it is possible to describe a service model implemented by a specific API technology. These two kinds of meta-models are then related to the Telecom service domain.

Implementation platform meta-models aim at concrete implementation technologies, such as EJB, C/C+ + , Java, etc. These implementation platform meta-models provided by MDE tools are independent of specific domains generally. The following figure 3.12 illustrates these principles:

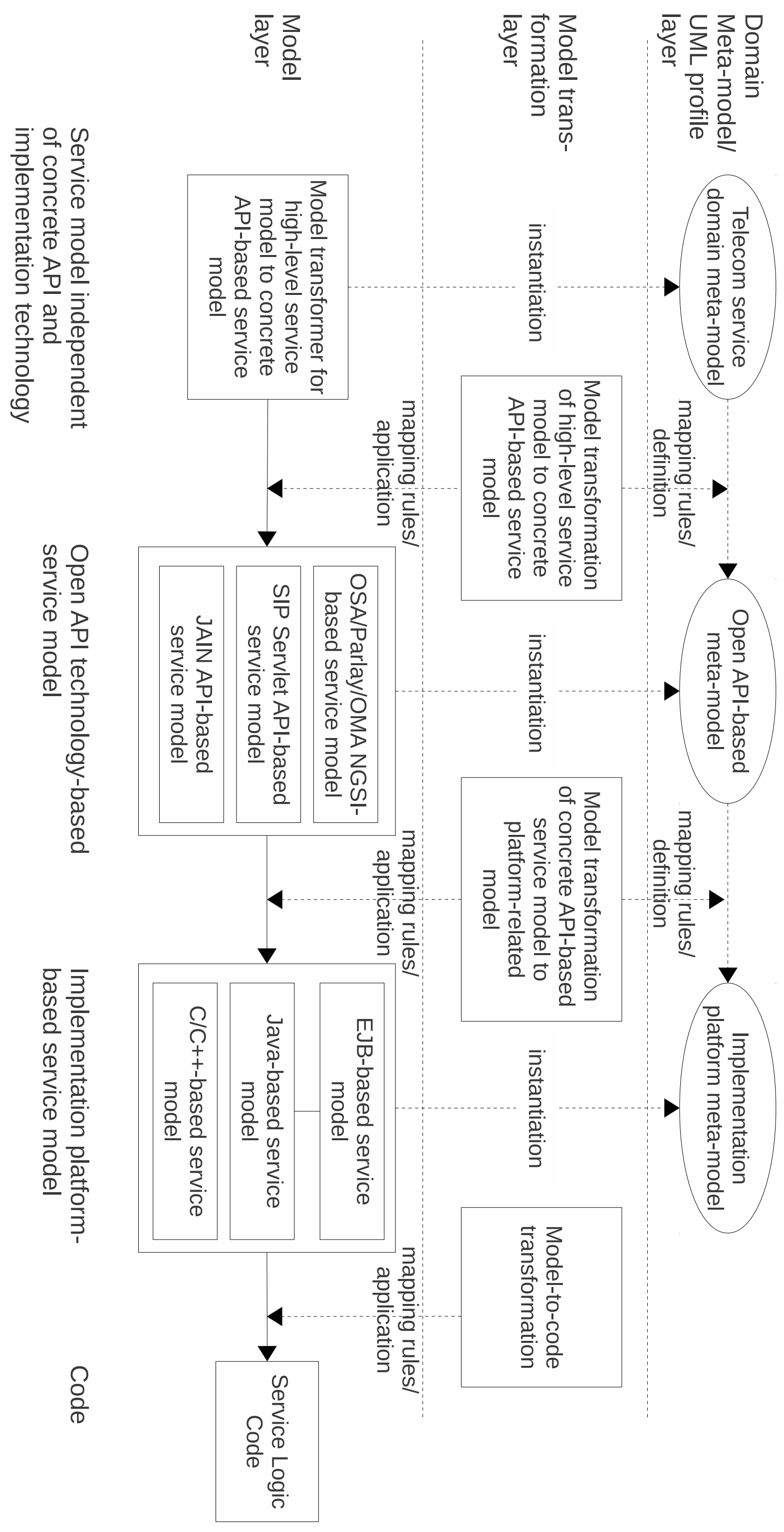

Domain Meta-model/ UML profile layer
Model trans- formation layer
Model layer
Telecom service domain meta-model
instantiation
Model transformer for high-level service model to concrete API-based service model
Service model independent of concrete API and implementation technology
mapping rules/ definition
Model transformation of high-level service model to concrete API-based service model
mapping rules/ application
Open API-based meta-model
instantiation
OSA/Parlay/OMA NGSI-based service model
SIP Servlet API-based service model
JAIN API-based service model
Open API technology-based service model
mapping rules/ definition
Model transformation of concrete API-based service model to platform-related model
mapping rules/ application
Implementation platform meta-model
instantiation
EJB-based service model
Java-based service model
C/C++-based service model
Implementation platform-based service model
Model-to-code transformation
mapping rules/ application
Service Logic Code
Code

Figure 3.12: Telecom Service Development Approach based on MDE

The model transformer layer is in charge of model-to-model and model-to-code transformation. Model transformers utilise the mapping rules to realise the transformation, they are defined according to the source meta-model and target meta-model. Using this approach, three different kind of model transformers are needed:

1. Transformation of high-level service models to concrete API-based model.
2. Transformation of API-based model to specific implementation/technology-related model.
3. Transformation of implementation/technology-related model to executable code.

With the support of related meta-models and model transformers, service designers are able to develop services applying the following steps:

1. Using constructs based on service domain meta-model to build up the high-level service logic.
2. Selecting appropriate model transformer to concrete API technologies.
3. Depending on the concrete service deployment environment, choose the corresponding model transformer to map the concrete API-based service model to a platform specific model.
4. Applying the platform specific model to the implementation platform transformer to generate code.

3.3.3.2 Domain-Specific Language Approach

DSLs offer notations and abstractions that are specific to a given domain. It captures domain knowledge (as opposed to code) and raise abstraction from the implementation world by using domain abstractions. In doing this, it applies domain concepts and rules as modelling constructs, narrowing down the design space and focusing on a single range of products. A DSL may or may not have a programming nature. Most DSLs developed by programming language researchers, although domain-specific, require programming skills. The following figure 3.13 compares DSL with other (model-driven) service creation approaches:

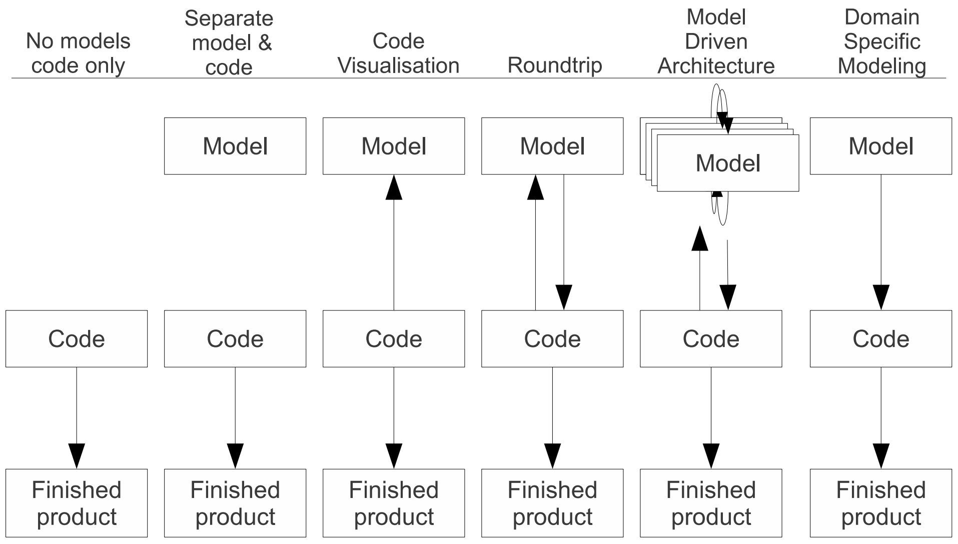


Figure 3.13: DSL in the context of other service creation approaches

Besides languages for processing data as PADS [FG 05], or MAWL, a language for interactive Web services [LR 95], DSLs have also been created for telephony service creation [BCL+ 06]. Although it is claimed that DSLs provide benefits in terms of productivity and safety [LMC 07], DSLs have mostly been targeted towards developers having some level of programming skills, excluding non-programmer domain experts.

Authors in [BCL+ 06] have designed a DSL for telephony services, named Session Procession Language (SPL). This language offers domain-specific programming constructs that permit services to be defined concisely and safely. By design, SPL is supposed to guarantee critical properties that cannot be verified in general-purpose languages. SPL serves as an interface between telephony models and telephony platforms. As the telephony domain imposes stringent safety and robustness requirements. A service should not itself incur runtime errors and should respect the underlying protocol. A number of verifications can be done at the level of the DSL layer. SPL has been designed to prevent errors that can occur when programming SIP services. In doing so, verifications are factorised because they do not depend on the target platform. As a result, it becomes easier and safer to introduce other modelling languages on top of SPL.

Another approach has been followed by IBM[6] and presented in [HKK+ 08], proposing the Telecom Service Domain Specific Language (TS-DSL) as a language for defining models of telecommunications services which are independent of specific architectures and protocols. This language is intended for service designers

[6]https://www.research.ibm.com/haifa/

who may not have Telecom industry domain knowledge. It hides the internals of the platforms focusing on their functionality and provides high level building blocks for the design of services. Model transformations are responsible for closing the abstraction gap by transforming the service model into a deployable service. TS-DSL includes both static and dynamic aspects of telecom services utilizing the power of UML2 and IBM's *UML Profile for Software Services*[7], refining and extending them for Telecom service domain.

At the point of writing, OMG is defining the Telecommunication SOA Modeling Language (TelcoML). The objective of this specification is to define a domain-specific UML Profile for designing advanced and integrated telecommunications services, meaning services that exploit the convergence of communication networks - landline, wireless and voice, and in the same time take advantage of facilities accessible from the World Wide Web. In TelcoML a service can be seen under two perspectives: a customer sees a UML-based service interface definition that lists the supported operations and parameters he should use to invoke the service. This can be considered as a black-box view. The provider defines the interface and possibly provides an explicit white-box behaviour definition for some of the operations. It should be noted that the specification is, at the point of writing, in a very early stage and not publicly available.

3.4 Process Constraint Principles

Technology continues to face challenges in coping with dynamic environments, where requirements and goals are constantly changing. The system defined within this thesis is conducive to dynamic change and the need for flexibility in execution. This section covers system governance enforcement principles for service execution through process constraints. We define classes of constraints and basic process constraints to propose a policy-based mechanism for the definition and enforcement of constraints.

3.4.1 Introduction

Workflow systems have been delivered effectively for a class of business processes, but typical workflow systems based on languages described in 3.2 are criticised due to their lack of flexibility, i.e. their limited ability to adapt to changing temporal business conditions. In the dynamic environment of e-business, it is essential that technology supports the business to adapt to changing conditions, where different process models should be derived from existing ones to tailor individual process instances.

[7]http://www.ibm.com/developerworks/rational/library/05/419_soa/

Providing a workable balance between flexibility and control is a challenge, especially regarding the performance of systems for real-time communications. Obviously, there are parts of processes which need to be strictly controlled through fully pre-defined models to reach execution performance requirements. But there can also be parts of the same process for which some level of flexibility must be offered, often because the process cannot be fully pre-defined due to lack of data at process design time.

We consider the definition and execution of constraints from the perspective of:

1. Network operators and service providers, intending to expose services for further re-use in other service domains and optimise the internal execution of service requests.
2. Service developers that want to compose those exposed services in a complex service work-flow.
3. Service users that want to protect their privacy and receive and optimised service experience depending on context information as presence, location, etc.

3.4.2 Classes of Constraints

In general, a process model needs to be capable of capturing multiple perspectives, in order to fully capture the business process [JB 96]. There are a number of proposals from academia and industry on modelling environments (languages) allowing these perspectives to be adequately described. Different proposals offer different level of expressiveness in terms of these perspectives. Basically these perspectives are intended to express the constraints under which a business process can be executed such that the targeted business goals can be effectively met. We follow [LSP+ 06] considering three essential classes of constraints:

1. selection constraints defining what activities constitute the process,
2. scheduling constraints defining when these activities are to be performed, both in terms of ordering as well as temporal dependencies, and lastly
3. resource constraints defining which resources are required to perform the activities.

These constraints are applicable at two different levels: process level and activity level. Process level constraints specify what activities must be included within the process and the flow dependencies within these activities including the control dependencies (such as sequence, alternative, parallel etc.). Activity level constraints constitute the specification of various properties of the individual activities within the process, including activity resources (applications, roles and performers, data), and time (duration and deadline constraints).

In the context of this thesis, we differentiate constraints either as permanent/ static or temporal constraints on multiple layers:

1. During service creation process, e.g., within an SCE, model-checking allows the verification of process and activity level constraints during service composition.
2. During service execution, e.g., a Service Broker providing the enforcement of temporal constraints by intercepting service requests:
    - Selection constraints: Sequences of service and features to be executed, e.g., specific logging, audio/video announcements before call setup.
    - Scheduling constraints: Temporal dependencies, e.g., enforcement of privacy rules based on location information.
    - Resource constraints: Definition of required resources, e.g., network resources (QoS), service enabler selection.

3.4.3 Policy-based Constraint Expression

According to [SL 02], we sort policies into two basic types: *authorization* and *obligation* policies. Authorization policies are used to define access rights for a subject (service provider request, user, or role). The evaluation outcome can be either positive (defining the actions subjects are permitted to perform on target objects) or negative (specifying the actions subjects are forbidden to perform on target objects). As such, authorization policies are used to define access control rules implemented by several types of mechanisms in a network security system, such as packet filters, Kerberos, and VPNs [AKG 05].

Obligation policies are, in turn, event-triggered condition-action rules used to define the activity subjects (services) that are allowed to perform on objects (e.g. service enablers) in the target domain. In the service exposure context, obligation policies can be used to specify the behaviour of services.

Many access control systems commonly provide groups of users as the access control unit. A major difference between groups and roles is that groups are typically treated as a collection of users but not as a collection of permissions. A role, serving as an intermediary, is both a collection of users and a collection of permissions. In e.g., Unix, because group membership is defined in two files (/etc/passwd and /etc/group), it is easy to determine the users belonging to a particular group. Permissions are granted to groups on the basis of permission bits associated with individual files and directories. Determining the permissions granted to a particular group generally requires a traversal of the entire file system tree. It is easier, therefore, to determine a group's membership than its permissions. Moreover, the assignment of permissions to groups is highly decentralized. Essentially, the owner of any Unix file system subtree can assign permissions for that subtree to a group. Although Unix groups are different from our concept of roles, in certain situations Unix groups can implement roles.

Although most access control and security systems do not always agree on the definition of roles [JSS 97], many to most support some form of Role-based Access Control (RBAC). Roles can be very complex entities, comprising constraints on role membership, constraints on role activation, and constraints on role use. Authors of [SCF+ 97] have defined a RBAC model as depicted in the following figure 3.14:

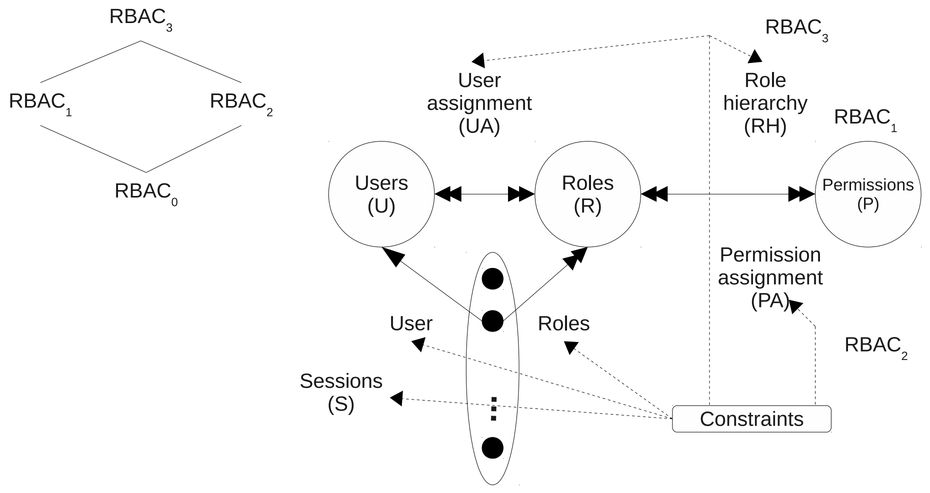


Figure 3.14: Role-based Access Control Models

RBAC$_0$, as the base model at the bottom, is the minimum requirement for an RBAC system. Advanced models RBAC$_1$ and RBAC$_2$ include RBAC$_0$, but RBAC$_1$ adds role hierarchies, whereas RBAC$_2$ adds constraints. The consolidated model, RBAC$_3$, includes RBAC$_1$ and RBAC$_2$ and, by transitivity, RBAC$_0$. Constraints are an important aspect of RBAC and are sometimes argued to be the principal motivation behind RBAC. A common example in our field is that of mutually disjoint roles, such as those of service provider and the system administrator. Generally, the same entity is not permitted to belong to both roles, because this creates a possibility for committing fraud. This well-known, time-honored principle is *separation of duties*. Constraints are a powerful mechanism for laying out higher level organizational policies. Once certain roles are declared mutually exclusive, there's less concern about assigning individual users/services to roles. If RBAC management is decentralized, constraints become a mechanism by which system administrators can restrict users'/services' ability to exercise privileges.

Applying RBAC principles on obligation policies, we can define a model providing limited inheritance of access and usage rights for users and services as depicted in the following figure 3.15 for a service provider that has access to the services S_1, S_2, S_3, and S_3’, whereas S_3, and S_3’ offer specific subservices labelled as tasks:

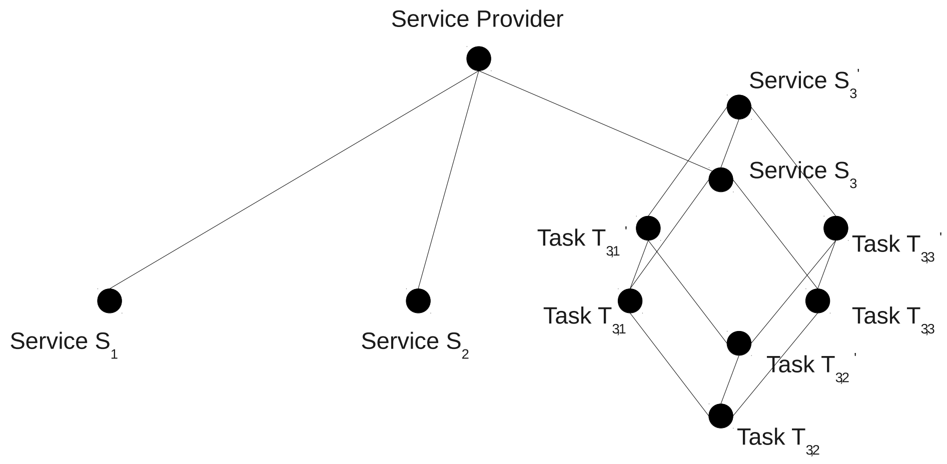


Figure 3.15: Limited Inheritance based on RBAC

3.4.4 Classification of Policy-based Constraint Definition

In the following, we provide a classification of popular policy languages, namely Business Process Execution Language (BPEL), Common Policies [STM+ 07], Web Services Policy Framework (WS-Policy) [W3C 07c], and eXtensible Access Control Markup Language (XACML) [OASIS 05] related to service exposure in the context of OMA Service Environment depicted in section 2.2.3.2.

3.4.4.1 Business Process Execution Language

In a homogeneous (Web) service environment where services that do not expose their functionality directly to a 3^{rd} party domain (rather their functionality is shielded by a BPEL engine exposing the functionality of the underlying services augmented with the necessary policies), it is possible to create a ’wrapper’ BPEL processes to expose the functionality of the underlying services. As BPEL supports control structures such as *if-then-elseif-else* and *while* as well as variable manipulation, service specific constraints can be expressed within a BPEL script. This approach follows OMA’s specification of PEEM as described in section 2.2.3.2 using a BPEL-engine as PEEM enabler. It can be applied to complex orchestrated services as depicted in the following figure 3.16:

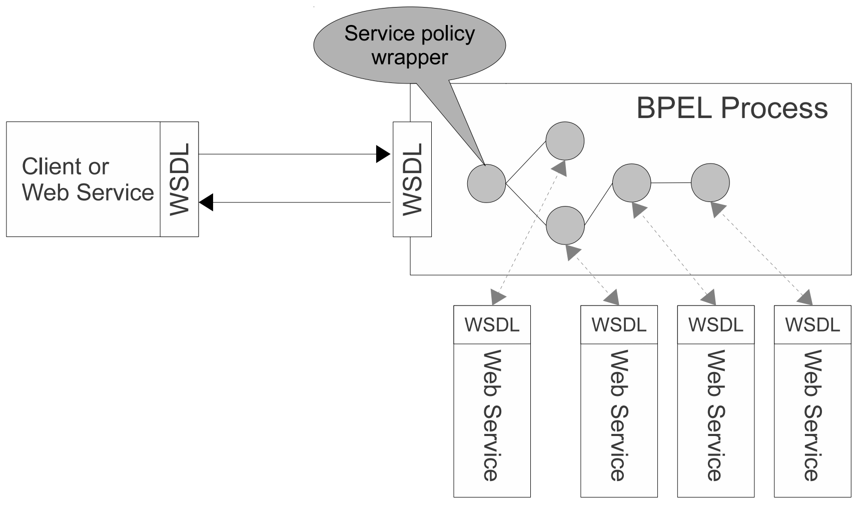


Figure 3.16: BPEL-based service policy wrapper for composed services

3.4.4.2 Common Policy

During the work in the IETF GEOPRIV (GEOlocation PRIVacy) working group[8], the need for authorisation policies that provide end-users with the possibility to disclose location information to 3rd parties, became obvious. Work was started on a common policy framework and document format [STM+ 07] following the general policy framework of [YPG 00]. This common policy framework and document format is then further extended for the specific needs with respect to location information [STM+ 09] and presence information [R 07], as depicted in the following figure 3.17:

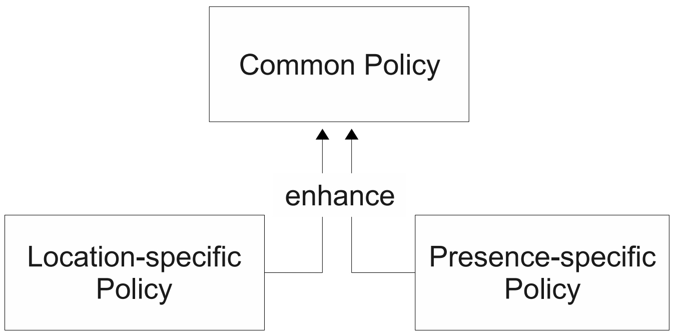


Figure 3.17: Common Policy Enhancements

[8]http://www.ietf.org/dyn/wg/charter/geopriv-charter.html

[STM+ 07] defines a policy as a rule set containing an unordered list of *rules*. A rule itself has *conditions*, *actions* and a *transformation* part. The term *permission* indicates the action and transformation components of a rule. A rule in a rule set can have a number of conditions that need to be met before executing the remaining parts of a rule. Transformations are operations that a policy server must execute and that modify the result which is returned to the policy requester. This functionality is particularly helpful in reducing the granularity of information provided to the requester, e.g., for presence and location information. Transformation are defined by application specific usage of this common framework.

3.4.4.3 WS-Policy

WS-Policy is specified by W3C [W3C 07c] allowing Web Services to use XML to advertise their policies (on security, Quality of Service, etc.) and for Web Service consumers to specify their policy requirements. It represents a set of specifications that describe the capabilities and constraints of security (and other business) policies on intermediaries and end points and how to associate policies with services and end points.

Figure 3.18 illustrates the WS-Policy data model. This figure indicates that a policy consists of a collection of policy alternatives. Policy alternatives consist of a collection of policy assertions. A policy assertion itself represents a requirement, capability or other property. It identifies a domain specific behaviour or requirement or condition.

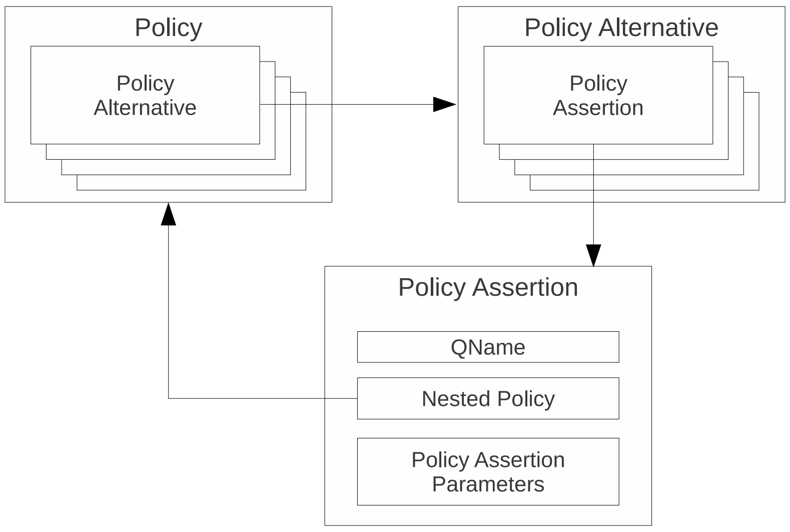


Figure 3.18: WS-Policy Data Model

A policy-aware client may use a policy to determine whether one of these policy alternatives can be met in order to interact with an associated Web Service. The WS-Policy model requires the awareness and explicit handling of policies by a requester.

3.4.4.4 eXtensible Access Control Markup Language

The eXtensible Access Control Markup Language (XACML) [OASIS 05] is a declarative access control policy language expressed in XML. It includes a processing model describing how to interpret policies. XACML allows defining fine-grained control of authorised activities, the effect of characteristics of the access requester, the protocol over which the request is made, authorisation based on classes of activities, and content introspection (i.e. authorization based on both the requester and potentially attribute values within the target where the values of the attributes may not be known to the policy writer). Figure 3.19 depicts the XACML language model:

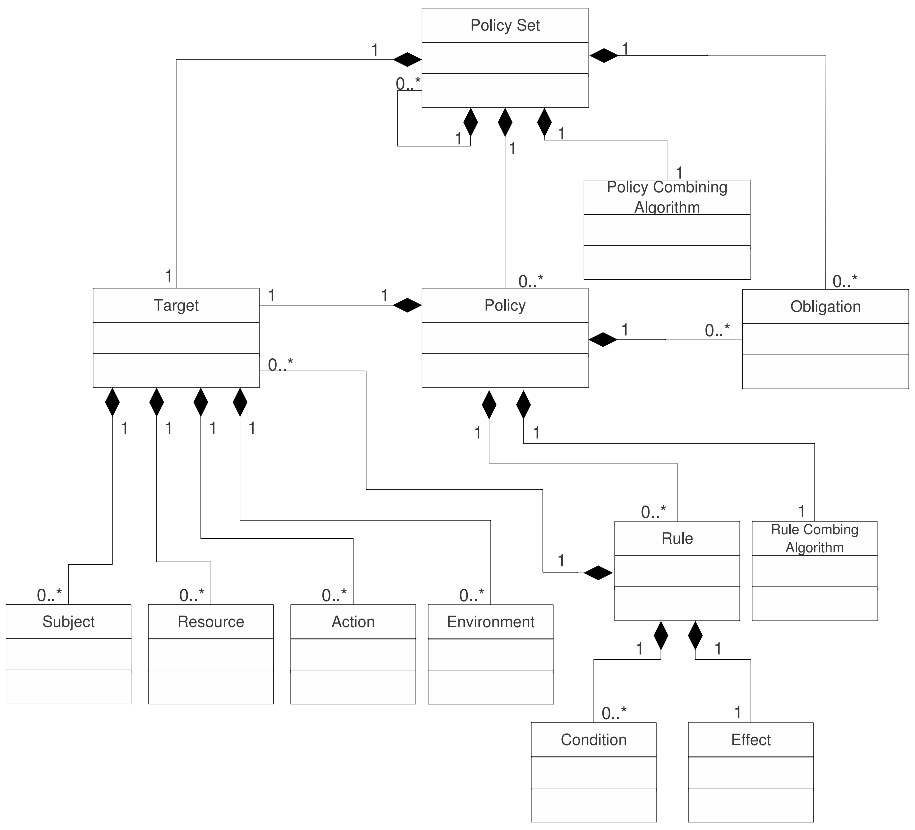


Figure 3.19: XACML Language Model

The main components of the model are *rules*, *policies* and *policy sets* as the most elementary unit of a policy. A rule can be evaluated on the basis of its contents. The main components of a rule are *targets*, that define the set of resources, subjects, actions, and the environment to which the rule is intended to apply, *effects* of a rule that indicate the rule-writer's intended consequence of a "true" evaluation for a rule, and *conditions* that represent a boolean expression of the applicability.

3.4.4.5 Classification

The following table 3.4 provides a classification of the above depicted policy languages according to the defined criteria in section 1.5:

	BPEL	Common Policy	WS-Policy	XACML
Configurability	-	+	++	++
Customizability	-	+	+	+
Modifiability	-	+	+	+
Network efficiency	n.a.	n.a.	n.a.	n.a.
Powerfulness	-	+	-	++
Scalability	- -	+	+	+
Simplicity	++	++	+	- -

Table 3.4: Classification of service policy languages

Configurability refers to the ability that a service might be re-configured through run-time without influencing clients and users. Considering BPEL as a policy expression language with a unified policy decision and enforcement, a new policy results in a new service deployment at the BPEL engine. WS-Policy and XACML provide the definition of multiple assertions or rules to a policy to allow enhanced configurability. The Common Policy framework provides the possibility to add multiple actions and transformations to a condition within one policy.

Customisability refers to the ability to temporarily manipulate the behaviour of a service through policy manipulation. As the provisioning of a policy within BPEL requires a new deployed BPEL service (wrapper service) to replace the old service, we rate customisability as low. Adding or changing an action or condition within a policy using Common Policy is trivial but requires also to change the existing policy, similar to adding a new policy or policy assertion to a WS-Policy and XACML. Therefore they all receive the same benchmark.

Modifiability is closely related to customisability but is not related to temporary changes. Nevertheless, efforts and risks are comparable with customising an existing policy to meet a temporary constraint. The rating is therefore similar, Common

Policy is benchmarked equal to WS-Policy and XACML as all require to change existing policies in the same way.

Network efficiency is not applicable to this classification, as network communication is usually not involved and can be considered as an implementation-specific issue of a policy framework.

Powerfulness refers to the strength of expression of service constraints. As BPEL and WS-Policy provide only capabilities to express policies for Web Services, they are rated lowest. XACML as the most complex of the depicted frameworks provides the most sophisticated options to express service-specific constraints.

Scalability refers to the ability to support a large number of services, or interactions among services within an active configuration. BPEL in contrast to the other options requires as depicted above a dedicated service for each exposed service and receives therefore the lowest benchmark. Common Policy, WS-Policy, and XACML allow the association of one or more policies to multiple services. Therefore, they all are rated equally.

Simplicity refers to the amount of information that is needed and the complexity to express and understand a service policy. Common Policy provides the easiest constructs and formats to define constraints. BPEL requires partner links to enforce constraints. XACML is the most powerful but also the most complex language. Compared to WS-Policy, it is not bound to a specific service protocol.

Concluding, it can be stated that XACML as the most mature policy framework provides the highest degree of flexibility and the greatest strength of expression at the cost of requiring the most complex language and evaluation mechanism. BPEL and WS-Policy have the major drawback that they are only suitable for Web Services. Common Policy is a relatively new standard, originally intended to express authorisation policies for access to application-specific data. Nevertheless, as the format allows service-specific extensions it can be applied beyond the primarily intended use. This light weight framework allows also to meet the requirement to express complex constraints.

3.5 Discussion

Throughout this technology analysis of service invocation technologies for distributed services, service composition languages, and policy expression languages, we have classified current most prominent concepts and languages. The following table 3.5 provides an overview of the results of the three classifications conducted in this chapter

	Service Invocation Technologies	Composition Languages	Policy Frameworks
Configurability	REST	BPEL, SCXML, SLG	WS-Policy, XACML
Customisability	JSON-RPC, REST	BPEL, CCXML, SCXML, VoiceXML	Common Policy, WS-Policy, XACML
Modifiability	REST	BPEL, CCXML, SCXML, VoiceXML	Common Policy, WS-Policy, XACML
Network efficiency	JSON-RPC	SCXML, SLG	n.a.
Powerfulness	Web Services	SCXML	XACML
Scalability	JSON-RPC, REST, Web Services	SCXML, SLG	Common Policy, WS-Policy, XACML
Simplicity	JSON-RPC	CCXML, VoiceXML	BPEL, Common Policy
Best distribution	JSON-RPC, REST	SLG, SCXML	Common Policy, XACML

Table 3.5: Classification Results

Adding onto this classification MDE as an approach allowing the formalization of services and platforms, we have depicted besides concrete technologies for service binding and execution also principles for the design of services and allowing the expression of service access and execution patterns with the help of formalized constraints.

In the following, we reflect the presented technologies in the context of the three roles of *service developers*, *operators/service provider*, and *users*.

Service developers need tools and paradigms tailored to their specific needs. We differentiate between developers in small and medium enterprises (SME) that have their main focus on innovative services with tight time-to-market business plans and large enterprises focusing on complex, sophisticated services as part of larger service frameworks. Whereas first ones need fast and easy access to services and service creation tool support might not be one of the main criteria, latter need dedicated SLAs and verified processes that do not harm existing software and customers when using the software. Those are in our focus for MDE-based service creation tools. Considering SMEs on the other hand, we believe that light weight APIs (e.g., REST, JSON-RPC) and easy access to new services are of major importance to stimulate new service development.

Operators need mechanisms to manage service access individually, defining constraints on service access during service creation and execution time. The more options and tools for fine-grained access management are available, the more possibilities an operator has to expose its network and service assets. Nevertheless, the systems have to be error-free and need to guarantee specific QoS in service execution and network connectivity. Proven constraint definition for service access and execution and verified service behaviour are considered as key system attributes. Therefore, the MDE-based approach is from our perspective of major importance. Furthermore, the system needs to be extendible to future services and platforms in a growing service market and a SOA provides such an approach. But current systems in an operator environment can be considered as service islands with limited access and interoperability. The transition to an open SOA is in this context a challenging task with many risks as verified service behaviour in complex, large scale systems is difficult to achieve.

Users want innovative services and in a growing network of services and data that is available to others, the ability to control the access to this data. Innovation in the Internet is closely coupled with openness related to technology in means of standards and platforms. To participate in the innovative field Internet and WWW from a telecommunications perspective, the technological paradigms of the web domain need to be applied to the telecommunications domain to attract users. This may be recognized on the one hand with the rise of the mobile Internet and on the other hand with the amount of services available for so called smart phones that are created and deployed to open market platforms easily. Currently missing in 2010 is the tight integration of communication features into available services. Furthermore, the user is starting to maintain relationships with service providers that are outside of the operator's domain providing services through the IP bit pipe. Therefore, the user needs tools and mechanisms to define and manage access to personal information as published location information or accessibility based on context information.

Concluding, we believe that the depicted mechanisms and technologies are applicable for all three roles in the context of this thesis, but, depending on the role and the use case to a different extend. MDE-based SCEs offer the ability to create services that are verified by model-checking and formal system/service description, but only within the scope and boundaries of the framework that is used for development. This might be a limitation when it comes to allowing innovative new approaches that have not been considered beforehand. The definition of constraints on various levels play an elementary role in our opinion when it comes to service access from an operator perspective as well as data access from a user point of view. In the following chapter, we present the blue print of a system incorporating these characteristics.

Chapter 4

A Service Broker for converged Internet and Telecom Services

4.1 Introduction

The overarching goal of adopting a Service Oriented Architecture is to allocate an organization's computing resources ensuring the direct alignment with core business processes. When implemented correctly, Service Oriented Architectures provide a framework that reuses existing elements of an IT infrastructure while reducing total cost of ownership and providing a more flexible and robust environment for the integration of IT and business processes. Services in a SOA are coarse-grained, discoverable software entities that exist as single instances and interact with consumers, applications, and other services via a loosely coupled, message-based communication model. These properties enable the flexibility of SOA because they remove dependencies on implementation specifics by relying on interactions between services through standardized interfaces.

The use of standardized interfaces also supports service virtualization, which allows entities to provide alternate interfaces to the same service instance. This furthermore allows value-added functionality to be inserted into the flow of a service invocation in a manner transparent to the consumer; similar concepts are being adopted in next-generation IP Multimedia Subsystem (IMS) and telecommunication networks. Loose coupling and service virtualization enable a dynamic and flexible integration infrastructure where different service providers, each of which serves as a perfect substitute for another, can be chosen at runtime to fulfil service requests.

In this chapter, we propose a Service Broker allowing the definition of the relationship between service developers, users and service providers to operator-owned (tele-)communication service enabler based on policy evaluation and enforcement. This Service Broker selects the appropriate service provider for a request, based on business and/or operations policies to optimally route service requests. Our approach is novel in its goal to efficiently maximize the value derived from the underlying IT resources through the application of service discovery and service execution-based rules. An instantiation of such a Service Broker platform delivers the promises of SOAs by enabling a dynamic and robust integration infrastructure that we believe is applicable to both middleware of next-generation telecommunications services as well as legacy services.

The remainder of the chapter is structured as follows: in the following section, we depict the main requirements from the perspective of network operators, service providers, service developers, and users. In section 4.3, an information and data model is defined for the interactions between these parties. Additionally, a policy taxonomy is derived from the model in section 4.4. We define a policy formalism in section 4.5. Section 4.6 provides the reference architecture of the Service Broker and the integrated methodologies. We also introduce the description of a Javascript/JSON-RPC high-level network abstraction API, providing easy integration of communications services into Web-based services. In section 4.7, we discuss trade-offs in the design of the platform regarding performance versus flexibility. Section 4.8 describes two selected use cases for the Service Broker from the perspective of the roles: service provider, network operator, service developer, and user. This section provides also a performance analysis of the prototype implementation and a comparison of the execution of different compositions.

4.2 Requirements

This section defines main requirements for the general architecture and service concept. We contemplate general requirements from the perspective of a network operator, service provider, service developer, and user.

4.2.1 Operator Perspective

Generally spoken, the Service Broker should provide a network operator an environment that allows the definition of its relationship with services, service providers, and service developers on multiple layers in multi-faceted way. The following table 4.1 provides an overview of these requirements:

Requirement	Description
Security	It is important for the network operator to secure its service and network infrastructure against malicious requests from other services that may harm its environment.

Operations	By dynamically defining service end points applicable for requests, the operator is capable of virtualizing service enablers for service providers. This allows to delegate requests to alternative platform, e.g., to optimise internal traffic or to route requests to alternative services in case of downtime of specific platforms or other operational requirements.
Openness for developers	The network operator should allow maximum openness to developers regarding access to developer APIs encouraging service developers to create new services.
Control for service execution	Requests from 3^{rd} party services should be interceptable on multiple levels (e.g., protocol layer, requested functions for execution, transmitted parameters) to allow differentiated levels of authorization of service requests.
Role-based Access	The Service Broker should provide the differentiation of roles of users/services that want to access the environment.
Adaptation of Service Logic	The Service Broker should provide means to insert a specific service logic into service requests (depending on e.g., business concepts) to apply or to compensate/hide internal changes to already existing services.
Performance	The Service Broker should not significantly reduce the performance of service execution.
Transparency	The Service Broker should be transparent to service consumers and providers.

Table 4.1: Service Broker requirements of a network operator

4.2.2 Service Developer Perspective

The Service Broker should provide service developers access to service enablers and service provider APIs. These APIs should not be subject to change from a developer perspective and should always be backward compatible to existing services. Furthermore, the Service Broker should allow the integration of APIs into SCEs to allow the development of services with multiple tools (e.g., IDEs, model-based SCEs). The following table 4.2 lists requirements from a service developer perspective:

Requirement	Description
Service Access	The Service Broker should provide a single point of access to service enablers, services, and service APIs to allow the creation of new services.

API integration	The Service Broker should provide means to allow the integration of its available service APIs into SCEs for convenient service development
API consistency	The Service Broker should provide APIs consistently, meaning e.g that even if operator-side service end points change, the Service Broker should hide such changes to ensure sustainable service execution at the service provider side.
Service Manipulation	The Service broker should allow the manipulation of service request to alter the behaviour of generic services without changing the API (e.g., the integration of announcements at the beginning of calls should be independent of the ThirdPartyCall API).

Table 4.2: Service Broker requirements of a service developer

4.2.3 Service Provider Perspective

The Service Broker should support service providers offering value-added services with access to operator assets to create and run new services independently of the operator infrastructure. It should be able to apply different business concepts for different user roles. The following table 4.3 lists requirements from a service provider perspective:

Requirement	Description
Service Profiles	The Service Broker should allow to define service profiles for different user classes/profiles.
Service Differentiation	The Service Broker should allow to provide mechanisms for easy service differentiation (e.g., advertisement support, different QoS).
Operator Virtualization	A service provider should be able to connect to many network operators to use their APIs. The Service Broker should provide means to abstract from specific operator APIs to offer a single API to service provider services.

Table 4.3: Service Broker requirements of a service provider

4.2.4 User Perspective

The Service Broker should provide the user with tools to protect his/her privacy and to define a specific user experience and preferences. The following table 4.4 lists requirements from a user perspective:

Requirement	Description
Privacy Protection	The Service Broker should allow users to define what data (e.g., presence, location) to expose in a fine-granular manner to other services.
User Experience	The user should be able to define its user experience of services (e.g., selection of services) on service exposure level.

Table 4.4: Service Broker requirements of a user

4.3 Definition of an Information & Data Model

The Information & Data Model is defined as an underlying abstract model for the main entities that interact in a converged Telecom and Internet service environment providing service/service and service/user interactions. The following figure 4.1 depicts the proposed model.

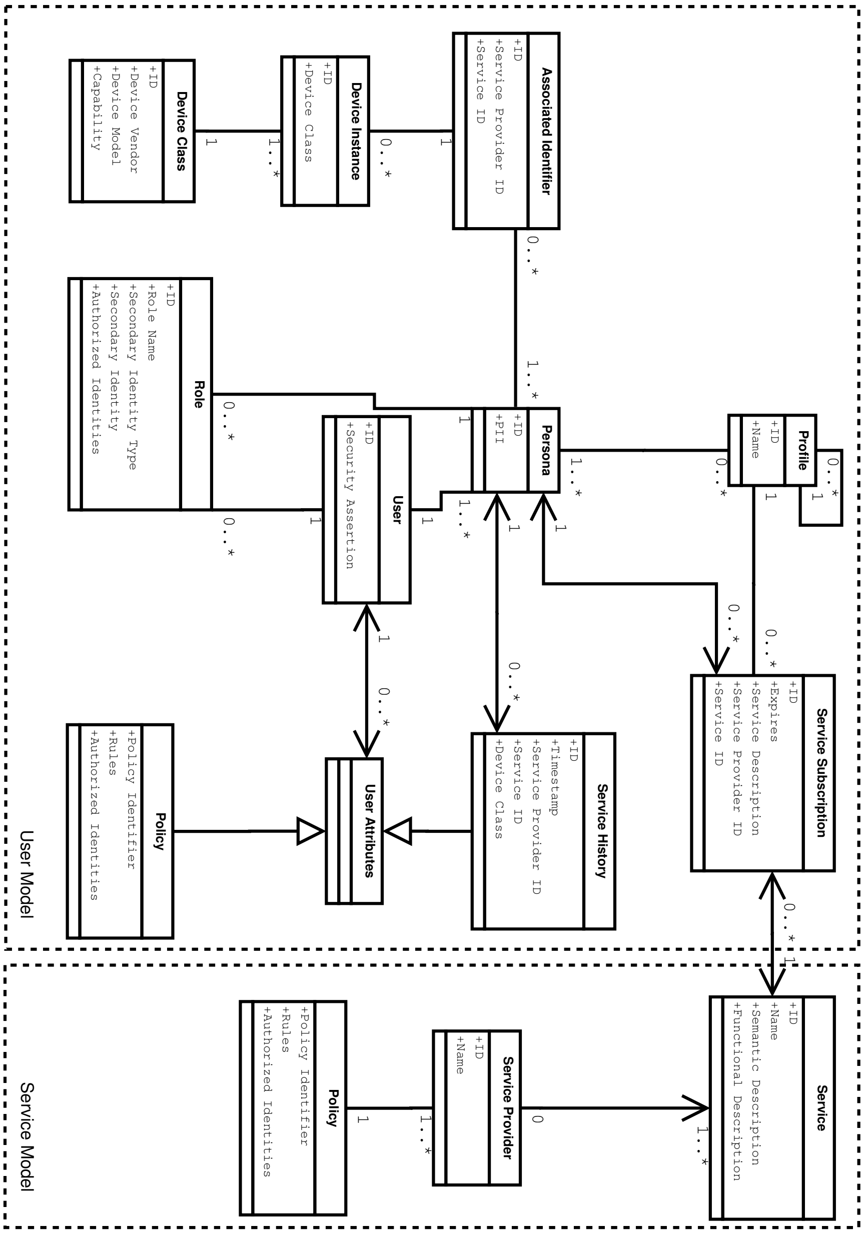


Figure 4.1: Service Broker Information & Data Model

In the context of this model, services may act on behalf of users (in the name of a user) and users are reachable through service front-ends on multiple devices and software environments (browser, mobile phone, IMS UA, etc.). From this model, we derive a general policy taxonomy in the following section 4.4. The model is separated into a user model and a service model. The following subsections depict each entity and the relation between them.

4.3.1 User & Persona Model

A user is represented within the system through the unique *User* entity, having a unique ID and associated security assertions, as e.g., credentials or Security Assertion Markup Language (SAML) tokens for Web Services security. A user may define several views on the system through Personas. A *Persona* is a concept for aggregating several user-related attributes which will be used in specific situations as configured by the user. Examples of attributes are birth date, address, etc. (so called Personally Identifiable Information (PII)). The following table 4.5 depicts the Persona model:

Name	Occurs	Description
ID	1	Persona ID
PII	0..1	Personally Identifiable Information

Table 4.5: Persona Model

4.3.2 Associated Identifier Model

One special attribute of the Persona is the *Associated Identifier*. The Associated Identifier represents an identifier of the user in a specific service context (e.g., user name, Email address, SIP URI). The Associated Identifier is defined in relation with a service and/or service provider. Given the implementation of the services and service providers for the Associated Identifier, credentials may be provided. This model offers support for services and service providers which need to authenticate on behalf of the user to execute specific services. The Associated Identifier is identified by the attributes in the following table 4.6:

Name	Occurs	Description
ID	1	Associated Identifier ID
Service ID	0..1	The Service ID to whom the identifier is associated.
Service Provider ID	0..1	The associated service provider.

Table 4.6: Associated Identifier Model

4.3.3 Role Model

A role is a set of connected behaviours, rights and obligations as conceptualized by identities in a social situation[1]. The *Role* entity defines what an identity (e.g., user or persona) does or, at least, is expected to do in relation with other identities (also called secondary identities). A secondary identity regarding the Role may be a group identity or administrator ID. The role of a user or persona may be assigned by the administrator, the user itself or other users. In order to protect the rights on the defined roles, the attribute Authorized Identity is introduced which defines the entities allowed to perform any modifications. The Authorized Identity is defined by an identity type (User or Role) and an ID (User ID or Role ID).

An administrator role has the Secondary Identity Type "system" and no Secondary Identity. For other users, the role "user" is automatically assigned. Another role that may be frequently used in the policy model is the "group administrator" which is associated automatically to the user that creates a group. The Role entity is defined as in the following table 4.7:

Name	Occurs	Description
ID	1	The identifier of this role
Role Name	1	The role name (e.g., Administrator).
Secondary Identity Type	1	May have one of the following values: system, users, group administrator.
Secondary ID	0..1	The specific second identity.
Authorized Identity	1..*	Authorized identities which have rights to modify or delete this role or its associated identities

Table 4.7: Role Model

4.3.4 Profile Model

An administrator may define specific QoS profiles for personas based on specific criteria (e.g., loyalty, service offering). In this context the entity *Profile* (e.g., silver, gold, enterprise 'xyz') will be used to reference administrator defined profiles.

If necessary, profiles may be given a hierarchical structure. Complex profiles, roles, and rights management in hierarchical groups are not considered to be necessary for the Service Broker as this should be subject of a collaboration/group server. Profiles allow besides the definition of QoS-related service attributes, also the reference to a set of services subscriptions providing automated classification of

[1]http://en.wikipedia.org/wiki/Role

users during service discovery and execution time. The following table 4.8 depicts the Profile model:

Name	Occurs	Description
ID	1	Profile ID
Name	0..1	The name of the profile.

Table 4.8: Profile Model

4.3.5 Service Subscription Model

A persona or service provider may define subscriptions to services. This can be considered as the definition of individual service profiles of a user. A *Service Subscription* is defined by a service reference and a service provider identifier. The Service Subscription Model is depicted in the following table 4.9:

Name	Occurs	Description
ID	1	Subscription ID
Expires	0..1	Time when the subscription expires. If the field is empty the subscription will never expire.
Service Provider ID	0..1	The service provider from which services are consumed including all its services.
Service ID	0..1	A specific service from the service provider.

Table 4.9: Service Subscription Model

4.3.6 User Attributes

User Attributes represent a set of attributes describing the behavioural part of a user, its personas, and associated identifiers from the perspective of the Service Broker. User Attributes consist of *Service History* and *Policies* expressing user preferences.

4.3.6.1 Service History Model

Based on the past experiences of a persona regarding service usage, a history may be maintained (e.g., services accessed within the last days, duration and destination of outgoing calls). A persona may have a service usage history for each service. The *Service History* entry is defined by the elements in the following table 4.10:

Name	Occurs	Description
ID	1	History ID
Timestamp	1	Represents the service usage time.
Service Provider ID	1	The service provider offering the service.
Service ID	0..1	The service ID of the consumed service.
Device Class	0..1	The device on which the service was consumed.

Table 4.10: Service History Model

This table does not include the Service Subscription ID because even if the subscription may expire the history can be maintained.

4.3.6.2 Policies

A user (global administrator or system user) defines preferences through policies (e.g., for specific personas) by configuring parameters of the Service Broker and services of a user. The configurable parameters may include besides the user private information:

- Disclosure policies of user (also persona) information (e.g., access permission given to service provider/user) to other users, groups or service providers.
- Policies regarding the definition of behaviour of services as a result of services subscription (e.g., targeting advertisement).

The according policy model is described below in section 4.4.

4.3.7 Device Instance Model

An associated identifier may be mapped to many device instances. During service execution time, based on the configurations/preferences (policies), it will be decided which device(s) to be chosen in order to consume a service. The following table 4.12 describes the *Device Instance* entity:

Name	Occurs	Description
ID	1	Device instance ID
Device Class ID	1	Reference to device class.

Table 4.11: Device Instance Model

4.3.8 Device Class Model

A *Device Class* is a collection of devices that have similar characteristics and can be managed in a similar manner. A device class is defined by the following model:

Name	Occurs	Description
ID	1	Device class ID
Device Vendor	1	The device vendor.
Device Model	1	The device model.
Capability	0..1	The hardware capabilities.

Table 4.12: Device Instance Model

4.3.9 Service Provider Model

The *Service Provider* entity references to several services it may offer. In case solely a service provider is addressed, the request refers to all its services. The following table 4.13 depicts this entity:

Name	Occurs	Description
ID	1	Service provider ID
Name	1	The name of the service provider.

Table 4.13: Service Provider Model

4.3.10 Service Model

A *Service* may also be considered as an ordinary user from a Service Broker perspective; consequently, policies and access information may be defined, providing rules for disclosed information and service execution behaviour by other users or service providers. The service entity is defined by the following table 4.14:

Name	Occurs	Description
ID	1	Service ID
Name	1	The name of the service.
Semantic Description	1	Semantic description about service capabilities. This information may be used for e.g., identifying result items of a search. Structured and well-defined semantic service annotation may allow automated composition of services.

Functional Description	1	Functional description of a service (e.g., WADL, WSDL).

Table 4.14: Service Model

4.4 Definition of a Policy Taxonomy

Policies are defined for expressing user and service-related preferences regarding actions to be performed by the Service Broker of service enablers when requesting a policy evaluation process for a specific event. Through policies, personalization of services according to specific preferences does not require specific adaptations in the core system or a service, thus enabling policy-based composition of services implicitly during service execution time. The following figure 4.2 depicts the developed policy taxonomy for the Service Broker:

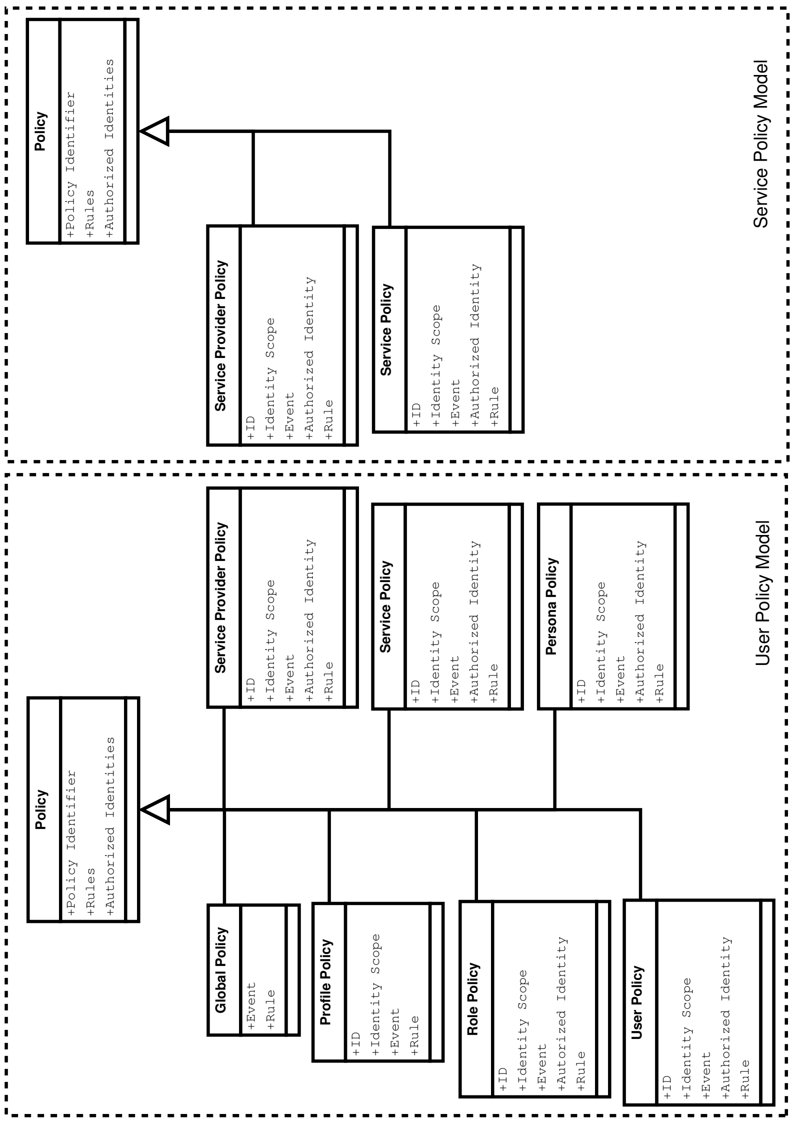


Figure 4.2: Policy Taxonomy

Policies are represented as a logical set of rules. A set of rules is selected based on the matching of a policy identifier with a set of input data. A policy is defined by the following model in table 4.16:

Name	Occurs	Description
Policy Identifier	1	Identifier of the policy.
Rules	1..*	A logical set of rules which compose the policy. When a policy is selected each of the rules will be evaluated.
Authorized Identity	1..*	Authorized identities defining the rights for operations *view*, *delete* or *modify* of the policy.

Table 4.15: Policy Model

The identifier of the policy needs to be deduced from the input data that needs to be processed against policies. The policy identifier is defined in the following table 4.16:

Name	Occurs	Description
Identifier ID	1	The ID (e.g., profile ID, role ID, user ID, persona ID, service provider ID or service ID); for a global reach, there should be no ID associated as they address all users/personas.
Identity Scope	1	Defines the relation of the identity regarding an event (e.g., request to a specific function call). It may have one of the following values: "Originator", "Target", "All".
Event Name	1	Event name represents the name of the operation inside the system (e.g., createCall) based on which a request for policy evaluation is triggered.

Table 4.16: Policy Identifier description

A rule is composed of conditions and actions. The conditions define the scope and the constraints imposed by a rule. The actions define the behaviour that should be enforced in case the evaluation of the conditions was successful.

According to the above depicted policy taxonomy several types of policies can be defined, the type of the policy is defined by the policy identifier value. The following subsections depict each type in detail.

4.4.1 Global Policy Model

A *Global Policy* is selected when a specific event is fired no matter of the identities included in the input data. It can be viewed/created/modified only by the administrator. In this case the policy identifier equals "global". The following table 4.17 depicts the Global Policy Model:

Name	Occurs	Description
Event	1	The event name (e.g., sendMessage, subscribeToLocation); can also be NULL (applies to all events)
Rule	1..*	The rule(s) of the policy.

Table 4.17: Global Policy Model

4.4.2 Profile Policy Model

An identity may be associated to a specific system profile, according to the Profile Model. Therefore a *Profile Policy* is selected based on the profile of the identity, its identity scope, and the specific event. It can be viewed/created/modified only by the administrator. The policy identifier has the identity type "Profile". The following table 4.18 depicts the Profile Policy Model:

Name	Occurs	Description
Profile ID	1	The ID of the profile.
Identity Scope	1	Defines the relation of the identity regarding the event ("originator" or "target").
Event	1	The event name (e.g., sendMessage, subscribeToLocation); can also be NULL (applies to all events)..
Rule	1..*	The rule(s) of the policy.

Table 4.18: Profile Policy Model

4.4.3 Role/Persona/User Policy Model

For each of the identities included in the input data a *Role/Persona/User Policy* may be selected based on either the role, persona, or user of the identity, its identity scope and the specific event. This policy type represents an expression of an individual preference. The policy identifier has either the identity type "Role", "Persona", or "User". The following table 4.19 depicts the Role/Persona/User Policy Model:

Name	Occurs	Description
Role/Persona/User ID	1	The ID of the role/persona/user.
Identity Scope	1	Defines the relation of the identity regarding the Event (“originator” or “target”).
Event	1	The event name (e.g., sendMessage, subscribeToLocation); can also be NULL (applies to all events).
Authorized Identity	1..*	User IDs or role IDs that are allowed to administrate the policy.
Rule	1..*	The rule(s) of the policy.

Table 4.19: Role/Persona/User Policy Model

4.4.4 Service Provider/Service Policy Model

This model includes policies selected based on the service provider ID, identity scope and event name. This class of policies may on the one hand be defined as a user-specific policy for a certain service provider and all its services or for a specific service offered by a service provider. On the other hand these policy types may as well be defined by a service provider for its service providing a general system configuration. Therefore, service provider or service policies may either belong to the User or to the Service Policy Model. The policy identifier has the identity type “ServiceProvider” resp. “Service”. The following table 4.20 depicts the *Service Provider/Service Policy Model*:

Name	Occurs	Description
Service Provider-/Service ID	1	The ID of the service provider/service.
Identity Scope	1	Defines the relation of the identity regarding the Event (“originator” or “target”).
Event	1	The event name (e.g., sendMessage, subscribeToLocation); can also be NULL (applies to all events).
Authorized Identity	1..*	User IDs or role IDs that are allowed to administrate the policy.
Rule	1..*	The rule(s) of the policy.

Table 4.20: Service Provider/Service Policy Model

4.5 Definition of a Policy Formalism

A policy is represented as a formalism developed in order to describe in a declarative manner the requirements related to the use and management of resources. It provides access control of resources, customization of access to resources, invocation of other resources, and manipulation of requests and parameters. Therefore, our approach can be considered as a combination of authorization and obligation policies expressing constraints according the definitions in section 3.4.3. The formalism should be easily extensible in order to adapt to any kind of resources and authorization scenarios. The policy model should be mapped to the proposed model in section 4.4. As defined before, a policy represents a logical set of rules. A rule represents a mechanism for defining a specific behaviour which should be enforced under specific conditions. A specific behaviour is defined as actions.

To formalize our policy-based approach, we use the language $\mathcal{L}_{active}$ presented in [BLT 97] and used in [RCC 09] to formalize the management of group obligations. $\mathcal{L}_{active}$ provides a formal characterization of active databases and allows reasoning about change in the state of dynamic systems which results from action occurrences in the system. A translation of the language to logical programs is presented in [BL 96]. The alphabet of $\mathcal{L}_{active}$ consists of four sorts:

(1) Fluents: are time-varying propositions or facts representing the system state.
(2) Actions: represent possible actions in the system.
(3) Events: are used to specify state conditions at which actions are required.
(4) Rule IDs: unique identifiers of Event Condition Action (ECA) rules.

The semantics of $\mathcal{L}_{active}$ is given by the following three propositions:

$(\mathbf{EL})\ a(\overline{X})$ **causes** $f(\overline{Y})$ **if** $p_1(\overline{X_1}), ..., p_n(\overline{X_n})$
$(\mathbf{ED})\ e(\overline{Y})$ **after** $a(\overline{X})$ **if** $p_1(\overline{X_1}), ..., p_n(\overline{X_n})$
$(\mathbf{AR})\ r(\overline{X_r}) : e(\overline{X})$ **initiates** $[\alpha]$ **if** $p_1(\overline{X_1}), ..., p_n(\overline{X_n})$

Where the symbols $f, p_1, .., p_n$ are fluent symbols, a is an action, e is an event, and r is an active rule identifier. An effect law proposition (**EL**) states that the execution of $a(\overline{X})$ in a state where the fluents $p_1(\overline{X_1}), ..., p_n(\overline{X_n})$ are true causes $f(\overline{Y})$ to be true in the next state. An event definition proposition (**ED**) states that if the conditions $p_1(\overline{X_1}), ..., p_n(\overline{X_n})$ are true in the state following the execution of the action $a(\overline{X})$, then event $e(\overline{Y})$ is produced. An active rule proposition (**AR**) states that every new detection of the event $e(\overline{X})$ initiates the execution of the sequence of actions $[\alpha]$ if the rule conditions are true.

The operational semantics of the language define a transition function which, given a state and a (potentially empty) sequence of actions, produces a new state. Actions in the input sequence are processed successively. For every action, its effect laws are evaluated and the state is updated. If after the execution of the action, conditions in some event definition are true, the event is generated. The newly generated events trigger active rules. The identifiers of the triggered rules are added to the triggered rules set. When the last action in the sequence is evaluated, if the triggered rule set is not empty, an action selection function selects the sequence of actions appearing in one of the rules to process.

The policy format, we have decided on is based on the classification results in table 3.5. It complies to [STM+ 07] and has been chosen due to its simplicity, openness for extensions, and service specific adaptability. Furthermore, it is named by OMA as an option for PEEM policies part of OSE. Appendix A provides the complete XML schema description and defined extensions. A rule example is depicted in the following listing 4.1:

```
<rule id="r1">
    <conditions>
      <originatorIdentity>
        <one id="sip:alice@open-ims.org"/>
      </originatorIdentity>
      <targetIdentity>
        <many/>
      </targetIdentity>
     <validity>
        <from>2008-05-27T14:11:00.943Z</from>
        <until>2010-05-27T14:11:00.943Z</until>
      </validity>
      <serviceOperation name="inviteParticipant">
<parameter name="participant">
   <operator name="notequal" operandsType="string" match="regx">
           sip:00[0-9]*@open- ims.org
         </operator>
  </parameter>
      </serviceOperation>
    </conditions>
    <actions>
      <invokeService url="http://www.thirdpartyprovider.com:8080/
          services/
ConferenceServiceWithAd" namespace="http://www.csapi.org/schema/
    conference/local"
name="ConferenceServiceWithAd">
        <invokeOpReq name="inviteParticipant">
          <forwardOriginal/>
      </invokeOpReq>
      </invokeService>
    </actions>
  </rule>
```

Listing 4.1: Rule listing

The above rule applies for `sip:alice@open-ims.org` message originator, for any target of the message, for the operation `inviteParticipant` and for the processing time between `2008-05-27T14:11:00.943Z` and `2010-05-27T14:11:00.943Z`. The rule is evaluated to *allow* only, if the value of participant does not match `sip:00[0-9]*@open-ims.org`. Otherwise the request will be denied. The actions define the invocation of a delegated service `ConferenceServiceWithAd` using the original message of a service exposed to a 3^{rd} party service provider which provides a conference service.

Rules are defined only for authorized access describing the constraints that the message must respect for a successful evaluation result. This approach reduces the policies conflicts that might rise up when defining contradictory rules. Conflict resolution in general is addressed below in section 4.6.4.1.

4.6 Architecture of the Service Broker

In this section, we propose our Service Broker architecture that explicitly links the value extracted from Telecom resources to other business processes and services.

4.6.1 Overview

The platform is composed of service-oriented intermediaries and service providers. The platform provides:

- A fully distributed service exposure infrastructure,
- Flexible and optimal selection of service providers that can be based on various system-level goals (e.g., user/service profile, operations optimization),
- A high-level telecommunications API for service integration into web mashups,
- Anchor points for model-based SCEs for service discovery and service execution.

The novelty of our proposal arises from the integration of several well-established theoretical and practical techniques from networking, service-oriented computing, and service creation that, together, form a fully-distributed Service Broker platform. The core component that enables the Service Broker is a policy engine. The objective of this component is the evaluation of requests on multiple layers (requestor, destination, requested action/function, parameters) and the assignment of an allow, deny or forward decision based on defined conditions and actions following the policy taxonomy introduced in 4.4 and the policy formalism introduced in section 4.5.

4.6.2 Key Assumptions

To build our Service Broker platform, we make several key assumptions:

- The network operator needs to provide a well-defined network abstraction layer incorporating a set of service enablers providing APIs that abstract of specific network technology. Service enablers and the API-based access are either distributed across platforms and networks or provide a single access point.
- The network operator intends to provide those service enabler APIs for the development and execution of services. It is interested in a fine-granulated definition of cooperative consumer/provider relationships on an individual basis.
- We assume an existing security assertion between the Service Broker operator and services/service providers that want to use exposed services.
- Service developers have their individual approaches and selection of tools and languages for service creation. We assume that enterprises have an interest to integrate communications enablers as verified and proven correct software into larger systems.
- We assume that consumers (e.g., 3^{rd} party services) only submit their service request to a single intermediary. This intermediary evaluates and delegates the service selection decision to an appropriate provider to achieve an optimal selection decision.
- Since the platform assumes global knowledge of per-service utility functions and trusted relationships between intermediaries, such that all nodes cooperate to optimally achieve common goals, it is assumed that the delivery platform exists within a single autonomous domain.

4.6.3 Reference Architecture

In this section we describe our design of the Service Broker enabling service access to domain-specific services for service developer and 3^{rd} party services using policies as a mechanism for service contract definition. The concept of a service broker is not new and a well-known entity of general Service Oriented Architecture (SOA) design patterns as depicted in the previous chapter. From an operator perspective, the following strategic aspects of such a function are most relevant for exposing services:

1. strong enforcement of flexible operator policies for service enabler usage
2. support for service discovery during the creation process
3. support of comprehensive security measures

Our design focuses especially on 1) and 2) as we consider 3) as an objective for a session border controller (SBC) or dedicated security gateways.

OMA defines its service environment (OSE) for generic NGN application enablers as described in section 2.2.3.2. We extend the concept of policy evaluation and enforcement of event-driven service requests to:

- definition of service behaviour and service selection through resource and scheduling constraints based on obligation policies,
- dynamic service capability description for service and developers using model-based SCEs.

The following figure 4.3 provides an overview of the Service Broker and its relation to service creation, service execution, and service enablers:

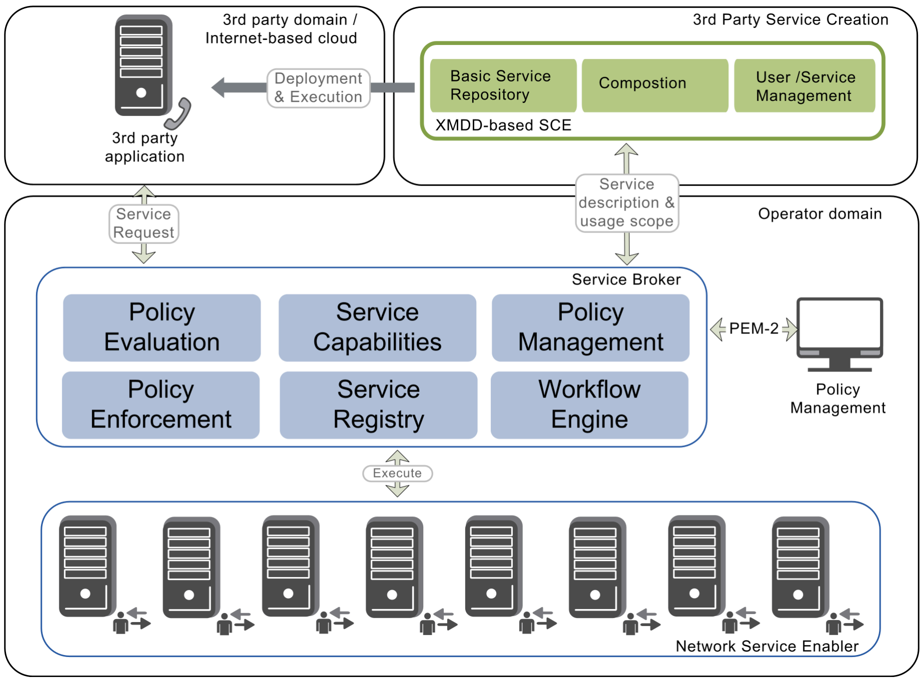


Figure 4.3: Service Broker in the context of service creation, execution, and enablers

Policy-based service access and execution within the Service Broker is designed as several independent modules that are accessible for external entities through OMA compliant interfaces, especially the PEM1 [OMA 08a] interface to trigger policy evaluation requests for incoming service requests and the PEM2 [OMA 08b] interface for policy management. The most important components of the policy-based Service Broker architecture are *Interceptor/Proxy*, *PEM1 Callable Interface*, *Policy Evaluation Engine*, *Policy Enforcement Engine*, *Service Registry* and *Service Capability Manager*, *Workflow Engine* , and *Policy Repository*.

The following figure 4.4 provides a more detailed architecture of the Service Broker and its interfaces:

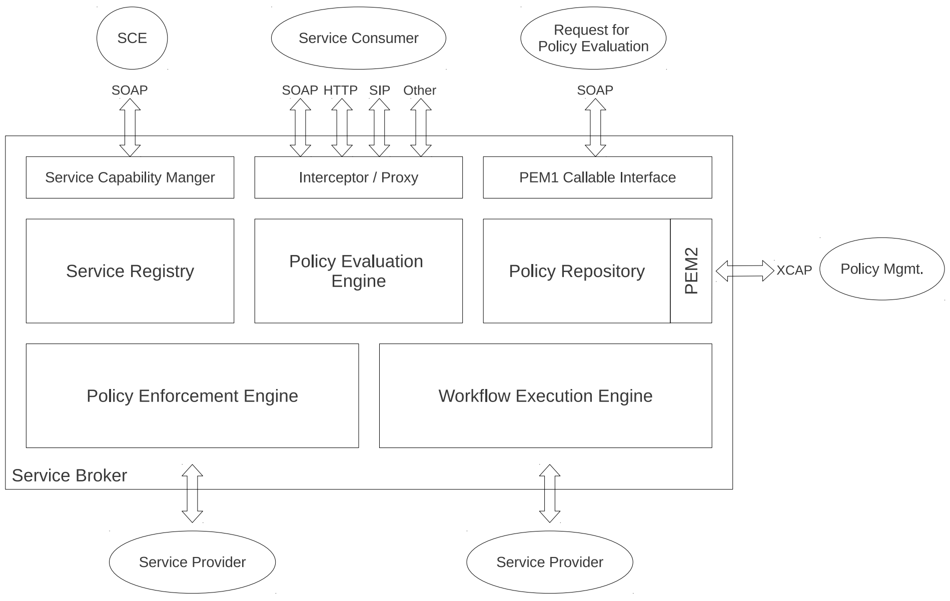


Figure 4.4: Service Broker Functional Architecture

The following subsections depict each function of the architecture in some more detail.

4.6.3.1 Interceptor/Proxy

The Interceptor/Proxy function may be applied to all critical interfaces within a service layer (e.g., as part of an ESB) or acts as the service exposure point (e.g for Web Services, RESTful APIs, etc.) within an operator's architecture and act as an intermediate system between resources. It acts as a proxy in the connection between a service consumer and a service provider and it is able to intercept every request and corresponding responses. The purpose of the Interceptor/Proxy is to enforce the evaluation resulted upon an intercepted message. An intercepted message itself does not represent its semantics for a policy evaluation process, therefore a transformation for preparing relevant data of a message needs to be applied preparing it as interpretable for an evaluation process. This mechanism involves identification of the request's relevant template and the creation of a document based on this template. The structure of this document and its semantic needs to be known by the policy evaluation component. The following figure 4.5 depicts the main subcomponents of the Interceptor/Proxy entity:

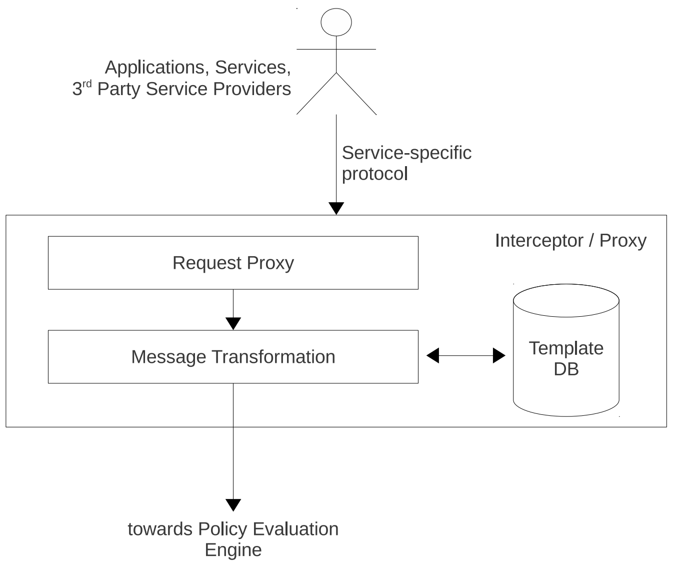


Figure 4.5: Interceptor/Proxy Architecture

A template is a transformation script that applies for a certain request and results into a data structure that respects the specifications described in the template. An example of a transformation script language is XML Stylesheet Language for Transformations (XSLT). XSLT may be used for the transformation of SOAP messages to the required policy evaluation input template.

The Interceptor/Proxy provides a decision mechanism based on the results received from the Policy Evaluation Engine as an output of the evaluation process. Thus, if the output from the evaluation process provides information that the message has been successfully authorized, the request will be forwarded to a specific target resource. This new request will be sent on behalf of the initial requester, the Interceptor/Proxy address will represent the source address of the new request. Otherwise, if the output contains information that the message has not been authorized, a rejecting message will be sent to the message initiator.

4.6.3.2 PEM1 Callable Interface

The PEM1 Callable Interface provides an explicit interface for policy evaluation that can be used by services to request for a policy evaluation result during service execution. It provides an alternative mechanism to the Interceptor/Proxy concept for services that are aware of the policy evaluation component in a service environment.

According to OMA specifications, the PEM1 interface provides a mechanism to identify and store the necessary information for policy evaluation consumption. This mechanism is called a template and may be designed generic for all services or specific for each service. PEM1 is defined to support SOAP and Diameter protocol. It provides operations for synchronous and asynchronous evaluation processes and defines a standardized way to communicate to the Policy Evaluation Engine. The payload of the PEM1 Callable Interface message is defined as XML for SOAP.

The Interceptor/Proxy does not make use of PEM1 Callable Interface to avoid unnecessary protocol overhead for internal Service Broker communication. Nevertheless, when the concept of distributed Interceptors/Proxies in a distributed service environment is applied, those (remote) Interceptors/Proxies shall use the PEM1 Callable Interface to communicate to the Policy Evaluation Engine.

4.6.3.3 Policy Evaluation Engine

The Policy Evaluation Engine's main activity is to fetch and evaluate associated policies of a message sent for evaluation. In order to achieve this, it provides a mechanism for identifying relevant policies, to evaluate policy conditions and to execute associated actions. Based on the above described behaviour, this component can be split into three main subcomponents:

- Policy Repository Requester fetches policies from a policy repository, either based on the input parameters received through PEM1 Callable Interface or the Interceptor/Proxy.
- Policy Condition Evaluation evaluates the policy conditions and decides which ones apply to a received request.
- Policy Action Execution executes matching actions between operation parameter requirements defined in the policy and parameters value received as input. It may also involve delegation to Policy Enforcement Engine.

Based on input data, the Policy Repository Requester identifies the policy identifiers that apply to the involved request, and initiates a request to fetch corresponding policies from the repository. Policy identifiers should respect the organization flow of the proposed policy repository, i.e. one example of a policy identifier is `TargetResourceName/Subscriber_Identifier.xml`. After fetching the associated policies, the Policy Condition Evaluation gets involved in order to evaluate the conditions of a set of policies. Based on the evaluation result of a policy condition, it may be decided whether the involved policy applies to the request or not. Rules that do not apply to the request are discarded from the policy set. The evaluation of conditions results either into:

- A true boolean value which translates into a decision of allow. In case a rule evaluates as true, the evaluation process will continue with the evaluation of other associated rules.

- A false boolean value which translates into a decision of deny. The evaluation of further rules is abandoned. Another case of message rejection is when no associated policies are found in the policy repository.

The next step in the evaluation process is the execution of actions associated to the evaluated condition task that is processed by the Policy Action Execution component. The involved actions represent either matching actions between the values stored in the policy and the values provided as input data or actions to delegate execution responsibility to other resources.

As one policy may contain more than one action to be executed, a combination algorithm has been defined. It respects the requirement that in case no policy is found to apply to the request, the evaluation process output will be negative for the request. It also takes into consideration that this decision should be configurable. Hence, an operator policy should be defined and describe what decision should be taken when there is no policy available. There are two algorithms to evaluate policies implemented for the Service Broker, but only one is used at a time based on a global configuration of the Policy Evaluation Engine. The optimistic algorithm allows everything, except what it is defined within a policy. In this case a request is denied if at least one policy from the associated policy set evaluates as `false` and is accepted if no condition part of a rule matches the request. The pessimistic algorithm defines a policy evaluation result as `true`, if at least one policy (and all associated rules) evaluates as `true` and is denied otherwise.

The following figure 4.6 depicts the main components of the Policy Evaluation Engine:

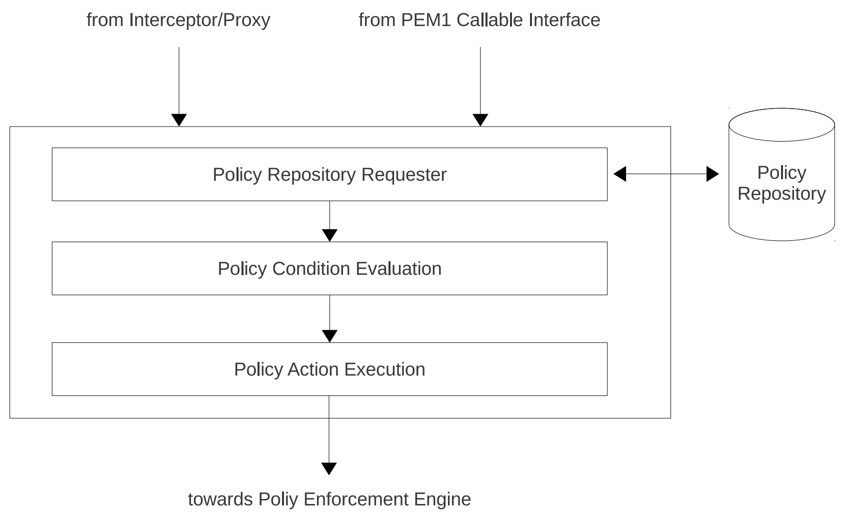

from Interceptor/Proxy
from PEM1 Callable Interface
Policy Repository Requester
Policy Repository
Policy Condition Evaluation
Policy Action Execution
towards Poliy Enforcement Engine

Figure 4.6: Policy Evaluation Engine Architecture

The next step of the Policy Evaluation Engine process is to forward the composed evaluation output to the Policy Enforcement Engine for enforcement of the request.

4.6.3.4 Policy Enforcement Engine

The Policy Enforcement Engine gets involved when the Policy Evaluation Engine has finished an evaluation process. It generates a more complex output based on the evaluation output and also on further execution output of actions and sends it to the policy evaluation requester. This enforcement decision is based on the result of the evaluation process and the execution of other actions that may be performed as consequence of an evaluation process. The following figure 4.7 depicts the main components of the Policy Enforcement Engine:

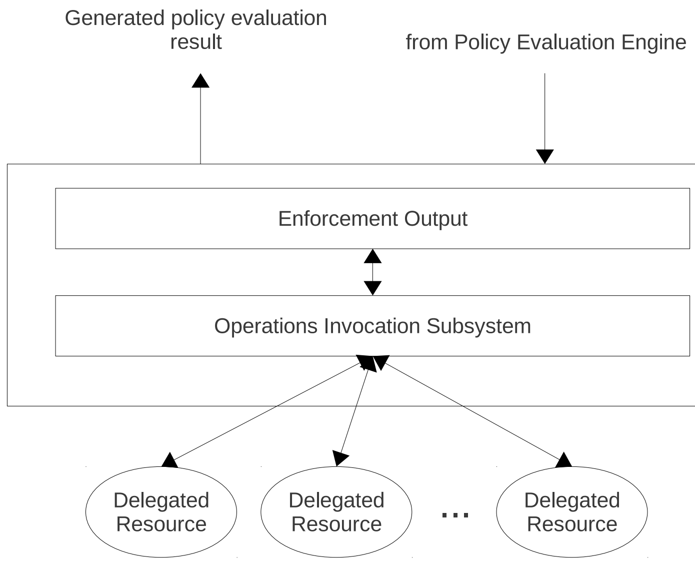

Generated policy evaluation result
from Policy Evaluation Engine
Enforcement Output
Operations Invocation Subsystem
Delegated Resource
Delegated Resource
...
Delegated Resource

Figure 4.7: Policy Enforcement Engine Architecture

For a better understanding of the functionalities within this component, it has been split into two main components:

1. Operations Invocation Subsystem invoking operations of other resources. Examples of this operations are charging, logging or requests for authorization. It involves initialization of the invocation process (further invocations to get certain parameter values) and the invocation process that executes the invocation steps for resource operations. An example may be the request for a presence status of a certain subscriber, as the Service Broker has to subscribe for the presence status of the involved user to receive its approval and in the end to receive the presence tuple set.

2. Enforcement Output provides a mechanism to combine the result of the policies evaluation from the upper layer (Policy Evaluation Engine) with the results of the resources invocation processes.
   The result of this combination algorithm respects the specifications of OMA related to PEM1 response messages and for each type of the result a predefined value should be used e.g., '2100' if a message has been authorized or in the case of denial '2400'. A detailed list of enforcement result codes is provided in [OMA 08a].

4.6.3.5 Workflow Engine

The Service Broker may incorporate a workflow engine to execute complex, composed services as part of a policy enforcement action of the Operations Invocation Subsystem. Based on the classification results in section 3.2.3 two options for defining abstract workflows as delegated services of a policy evaluation result are defined:

1. State Chart XML-based workflows
2. Business Process Execution Language-based workflows

Whereas the first option provides the possibility of generic workflow definition independent of a specific service invocation technology, the latter option is targeted on composed Web Services-based services that are provided again as Web Services. BPEL has been taken into consideration especially as it defines the current de-facto standard in enterprise service environment for the orchestration of services. We provide a more detailed analysis of performance-related pros and cons of these two options in section 4.8.3.

4.6.3.6 Service Registry

One main general task of a service broker is the awareness of available services to provide a brokering mechanism for. These service descriptions and meta information are stored in a service repository. Services available at the Service Broker may be described using Web Services Description Language (WSDL) or Web Application Description Language (WADL) [W3C 09a], an XML-based file format that provides a machine-readable description of HTTP-based web applications, typically REST web services.

4.6.3.7 Service Capability Manager

The Service Capability Manager (SCM) provides access to the service registry for developers and services acting in the name of a user with its persona and associated identifier. The result set of the SCM is based on a policy evaluation process. To compute the list of available services and profile information, the SCM must first have a full list of all available services retrieved from the service registry. The Policy Evaluation Engine must then be asked (using the callable PEM1 interface) for each service if it is allowed for the user. The following image depicts the functionality of the SCM within the Service Broker.

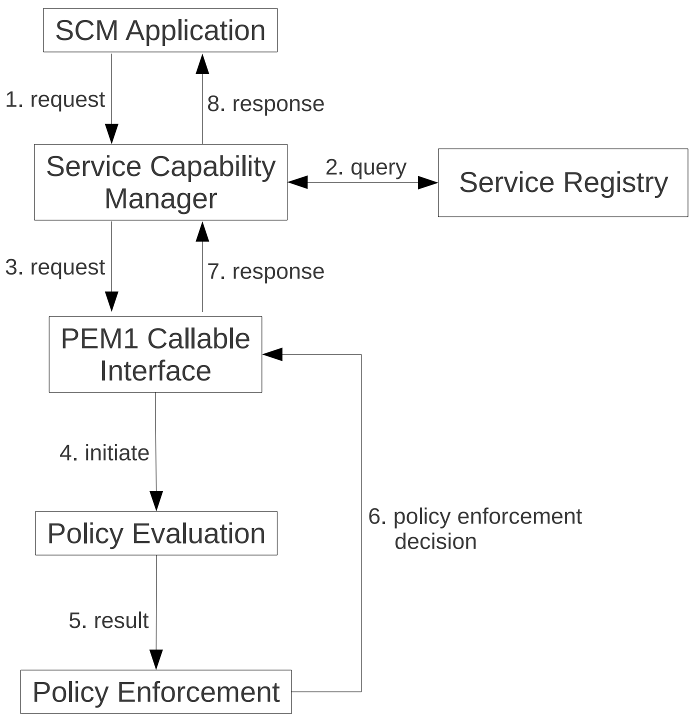


Figure 4.8: SCM functionality

SCM provides support for the concept of profiles as depicted in the Service Broker Information & Data Model. The main idea is to share policies between groups of users to classify them for different service level agreements. E.g., an operator may want to classify its subscribers to “Silver” or “Gold” profiles offering special service levels expressed in policies.

4.6.4 Methodologies Integrated in the Platform

The service broker platform is based on the integration of several key methodologies: constraint/policy evaluation/enforcement, network abstraction, and service exposure. In the subsections below, we give a brief overview of relevant issues related to each the methodologies applied to the Service Broker architecture.

4.6.4.1 Policy Evaluation/Enforcement Algorithm

System and service behaviour definition within the Service Broker is expressed through rules as part of a policy as described above. These rules are solved using a forward-chaining, data-driven algorithm that compares data in the working memory against the conditions of rules and determines which rules to fire. The applied forward chaining procedure is described in the following figure 4.9:

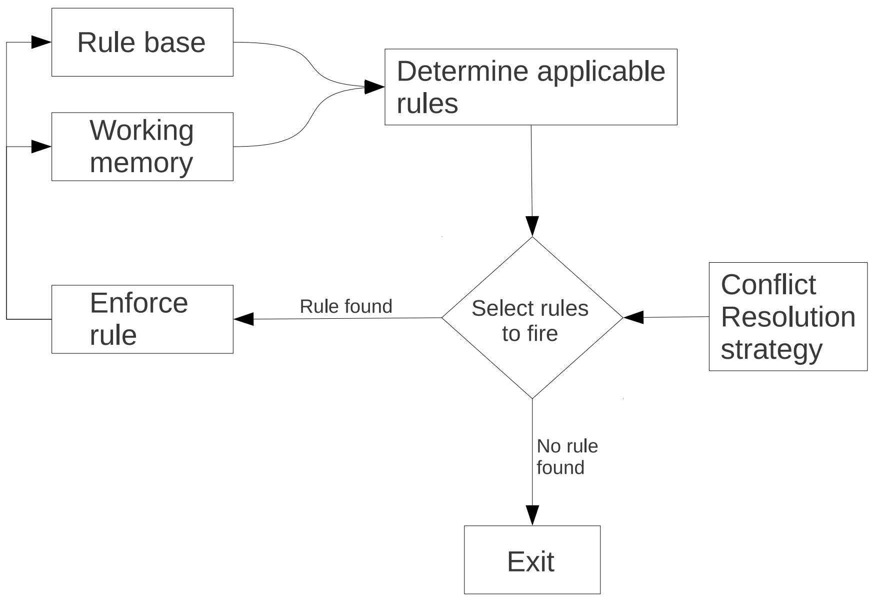


Figure 4.9: Forward-Chaining Procedure

Given the increasing number of possible applicable rules, the evaluation process may become exponential costly. Forward-chaining systems, as powerful as they may be if well designed, can become cumbersome if the problem is too large. As the rule-base and working memory grow, the brute-force method of checking every rule condition against every assertion in the working memory can become quite computationally expensive. When R is the number of rules, C is the approximate number of conditions per rule, and A is the number of assertions in working memory, such an algorithm may reach a processing requirement P, with the following big O notation of

$$P \equiv O\left(RA^C\right)$$

In order to overcome the complexity problem, the Rete algorithm [F 82], an efficient pattern matching algorithm for implementing production rule systems has been designed. Most of the current production rule engines (e.g., Microsoft Biztalk

Server[2], JBoss Drools[3], Oracle Business Rules[4]) use the Rete algorithm with improvements regarding memory consumptions. The Rete algorithm keeps up to date information associated with the nodes in a graph. When a fact is added or removed from the working memory, a token representing that fact and operation is entered at the root of the graph and propagated to its leaves modifying the information associated with the nodes. When a fact is modified, e.g., the validity of a service is changed from date 2010-05-27T14:11:00.943Z to 2010-12-27T14:11:00.943Z, this is expressed as a deletion of the old fact and the addition of a new fact. Rete is implemented as a rooted, acyclic, directed graph, providing an interconnected network of nodes.

The Rete algorithm reduces the complexity by reducing the number of comparisons between rule conditions and assertions in the working memory. To accomplish this, the algorithm stores a list of rules matched or partially matched by the current working memory. Thus, it avoids unnecessary computations in re-checking the already matched rules (they are already activated) or un-matched rules (their conditions cannot be satisfied under the existing assertions in the working memory). Only when the working memory changes, it re-checks the rules, and then only against the assertions added or removed from working memory. This method drops the complexity to a linear rather than exponential complexity:

$$P \equiv O\left(RAC\right)$$

The Rete algorithm, however, requires additional memory to store the state of the system from cycle to cycle. The additional memory can be considerable, but may be justified for an increased speed efficiency.

Concerning our strategy for conflict resolution of multiple applicable policies/rules on rule prioritisation. The priority relation of the identities is defined as:

$$User < ServiceProvider < Administrator$$

Policies are evaluated in the order of their priority starting from the one with the highest priority. Prioritisation is furthermore already applied during evaluation to determine the set of applicable rules reducing the rule set in a first iteration only to Administrator rules, in the second iteration to all service provider rules, and in a third iteration to all user-defined rules. This optimising approach allows a reduction of evaluated rules to a minimum set.u

4.6.4.2 Network Abstraction & Service Exposure

Current network abstraction concepts are based on APIs providing services for network service-specific protocols as depicted in section 2.1.5. Current state of

[2]http://www.microsoft.com/biztalk
[3]www.jboss.org/drools
[4]http://www.oracle.com/appserver/rules.html

standardization at the point of writing defines SOAP-based Web Services APIs labelled as Parlay X and REST-based APIs, named the GSMA OneAPI. Taking into consideration the evolution of APIs in the Internet, JSON-RPC and REST APIs as depicted in sections 3.1.3.3 and 3.1.3.2 are favoured by most developers. We have defined a Javascript/JSON-RPC API that can be integrated directly into HTML/-Javascript by developers. It is structured in the following functional components:

- `LoginHandler` – provides service, user authentication
- `ServiceCapabilties` – provides service discovery
- `AddressBook` – manages address book information
- `Call` – creates a call on a device
- `ConferenceCall` – creates a conference session
- `Location` – manages location information
- `Messaging` – send/receive an Instant Message
- `Presence` – manages presence information
- `ShortMessaging` – send/receive SMS
- `ThirdPartyCall` – creates a call between two parties

All functions of the components are static. User may choose between synchronous or asynchronous calls, asynchronous calls expect a callback function as input parameter. The JavaScript code continues running and the callback function is invoked when a service enabler function returns the requested result set. To invoke a synchronous call set, the callback function is simply NULL. In this case the JavaScript code waits until the service enabler returns the requested result set. Therefore, it is highly recommended to use asynchronous calls due to the fact that synchronous calls may cause an application to freeze. The full specification of the API is provided in Appendix B. The current on-going standardization efforts within the OMA NGSI program support our decision for high-level API definition to suit Web developers.

4.7 Engineering Trade-off of the Service Broker

High performance and very low delay in service execution are mandatory for service environments in the telecommunications domain. As the Telecom business environment is subject to fundamental changes through challenges by the IT and Internet industry, flexibility to adapt to new situations in service usage, creation, and deployment is key to survive from an operator's perspective. The following subsection discusses concepts applied in our design of the platform to allow a maximum of flexibility for a changing set of requirements from users, service providers, and service developers versus efficiency of service execution.

4.7.1 Flexibility versus Efficiency

We discuss the possible increase of flexibility and decrease of efficiency of our solution by evaluating on the one hand the decrease of efficiency through policy evaluation and enforcement by the Service Broker and on the other hand the increase of flexibility through policy-based orchestration of services.

The applied policy-based access control and service execution mechanism described above, even if optimised through using the Rete algorithm with enhancement through prioritisation reducing the number of rules to be evaluated imposes still an additional latency in service execution. We investigate two use cases to determine latency caused by the Service Broker during service request and execution time:

1. Authorization of a Parlay X ThirdPartyCall Web Services request resulting into an IMS SIP-based B2BUA INVITE message.
2. Manipulation of presence data provided by the Parlay X PresenceConsumer Service Web Service based on policies.

Whereas the first case requires a policy evaluation process for service request authorization, the latter one requires the execution of a composition to change presence status values according to defined values within a policy. The following table provides an overview of mean execution time of service requests for both cases above, with and without policy evaluation and enforcement process within the Service Broker[5]. It is calculated by taking the average computation of time of 250 executed requests at rate of 1 req/s.

	ThirdPartyCallService	PresenceConsumerService
Case 1: Mean time without Service Broker	7 ms	3 ms
Case 2: Mean time with Service Broker	50 ms	100 ms

Table 4.21: Mean time of Policy Evaluation

As expected, simple authorization of a request takes much less time than the execution of a composition. It can be stated that a policy-based authorization of a service request takes about $\sim \frac{1}{2}$ of the time of the execution of a composition, adding a mean time latency of `~40`ms to a request, whereas the execution of composition adds `~95ms`latency to the general service execution time. The policy evaluation process itself takes `~15`ms. It should be noted that the performance measurements depend highly on the SuT used.

[5]The System under Test (SuT) is described in section 4.8.3

4.7.2 Discussion

The increase of flexibility provided by policy-based access and usage definition can be broken down to the following statements:

- Definition of fine-grained access rules, e.g., user, service provider-specific access definition depending on time, origin, parameter values.
- Policy-based service orchestration resulting in composed complex services defining additional service behaviour without changing service logic on code level.
- Definition of optimal service routing paths based on policies, e.g., routing of service requests to service enablers based on operations rules or geographical dependencies.

We believe that there is no globally valid statement for all operator/service/service provider relationships stating that either efficiency in service execution or flexibility in access definition is more important. Generally, considering long term, well-defined service contracts, a great flexibility for service contract definition is probably of less use than efficient service execution. A static access mechanism realised through firewall rules in a hardware-based Session Border Controller (SBC) provides in this case a more efficient way to access operator-owned service enabler. But, when addressing a great variety of service providers that make use of service enabler capabilities on a changing case-by-case basis, the flexibility of our Service Broker solution is highly needed to meet service provider requirements on the one hand and to maximise control of operator assets on the other hand. This is the target environment that we have tailored our solution for.

Considering the operator/developer relationship, one may state that the greater the access for developers regarding services and service functionality is, the more services may potentially be created and a controlling entity as the Service Broker is limiting growth in services and usage. As we do agree with this statement, we still believe that operators engaging with many parties, enterprises and individuals at the same time on one logical platform should have mechanisms at hand to control access to service end points not only from a service/user but also from a developer perspective. We believe therefore that our Service Broker especially addresses this case by the definition of the above described SCM interface.

4.8 Validation

To illustrate the flexibility of the system for defining and enforcing temporal constraints (see section 3.4) to adapt to different criteria that may be of interest in particular situations, we describe two validation use cases focusing on operator/service provider relationship and on operator/developer interactions. We have integrated our Service Broker into the Open SOA Telco Playground[6] at Fraunhofer Institute FOKUS[7], providing an open environment for NGN service prototyping and validation ([BGM 07], [BMS+ 09a]). We conclude this section with a performance analysis.

4.8.1 A converged Internet / Telecommunications Service

As the main purpose of the Service Broker is to make operator-specific assets (service enablers) available to 3^{rd} parties, we have implemented a set of web-based widgets using the JavaScript/JSON-RPC API introduced in section 4.6.4.2. These widgets are integrated into the iGoogle [8] widget framework. The following image provides a screen shot of the used widgets.

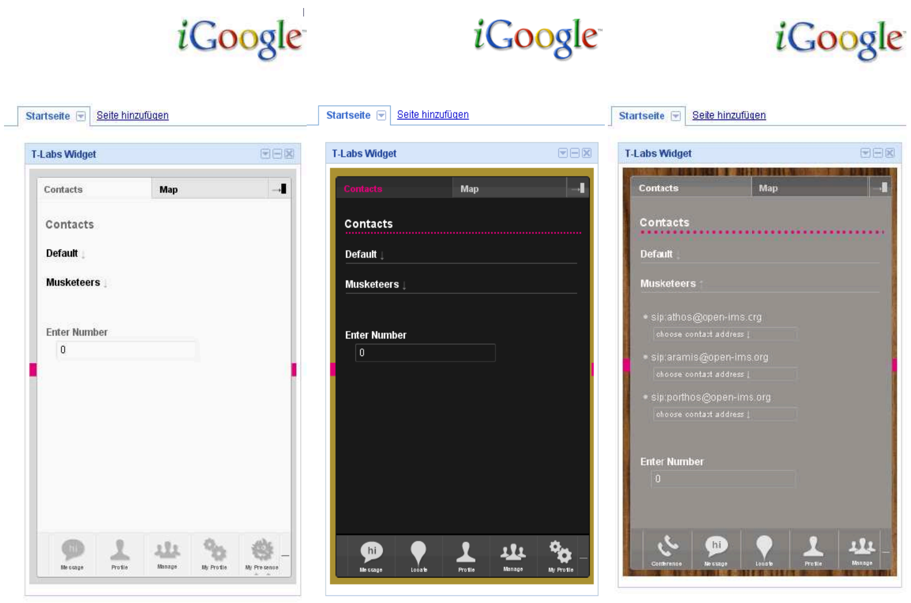


Figure 4.10: Web-based Telecom Widgets

The architecture is based on several layers of protocol/API abstraction in which the Service Broker manages the operator/service interactions by enforcing access policies and temporal constraints. The following figure 4.11 illustrates the high-level architecture:

[6]http://www.opensoaplayground.org
[7]http://www.fokus.fraunhofer.de
[8]http://www.igoogle.com

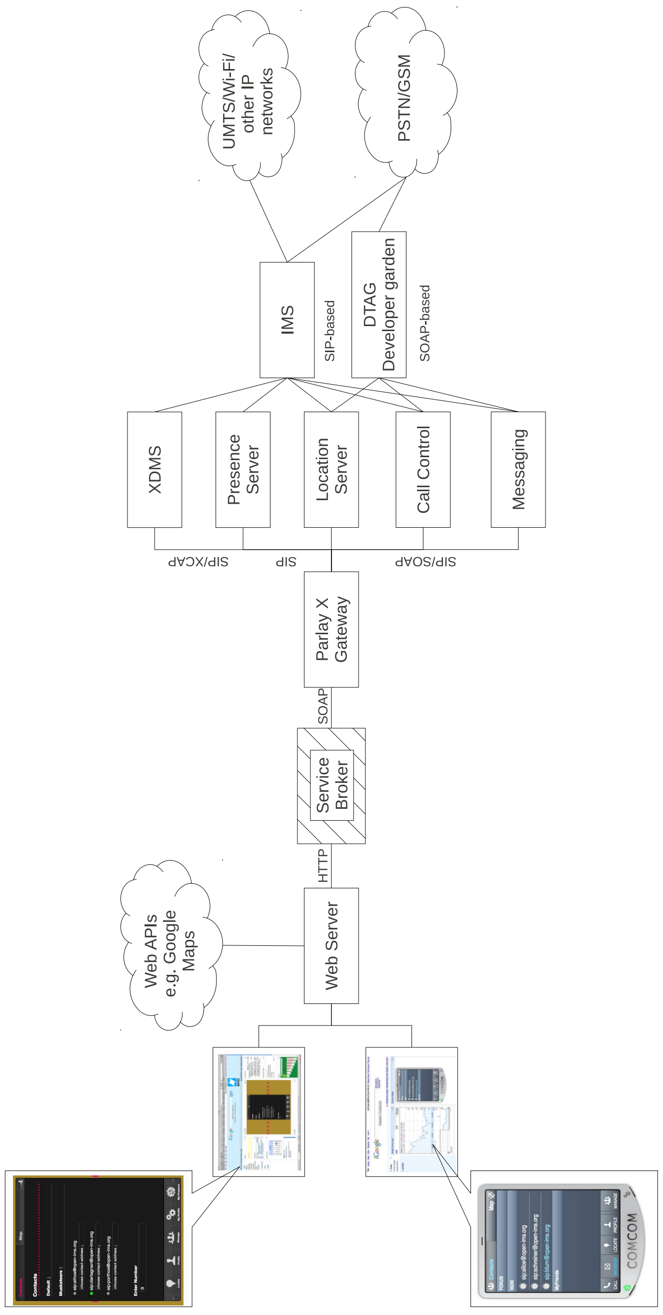


Figure 4.11: Telecom Widget Back-end Architecture

The developer uses the JavaScript/JSON-RPC API [BLK+ 08] to include into a mash-up operator provided service enablers as address book management, call control, conferencing, Instant Messaging, location, presence, and SMS. The API end points of the JavaScript API are provided by the Service Broker and translated into Parlay X-based SOAP requests. The message intercepting, policy evaluation and enforcement is applied on SOAP level by the Service Broker and forwarded to a Parlay X gateway communicating via enabler specific protocols to the service enablers. These are connected either via SIP/XCAP to an IMS core system or to SOAP-based services provided by Deutsche Telekom Developer Garden[9]. The following subsections depict selected service use cases illustrating Service Broker functionality applied on requests initiated by these web widgets.

4.8.1.1 Service Discovery via SCM

Figure 4.10 illustrates the same widget with for three different profiles Platinum, Gold, and Silver, each providing a different set of services and features. The widget application makes actively use of the SCM interface of the Service Broker for service and profile discovery. The following sequence chart 4.12 illustrates the SCM flow:

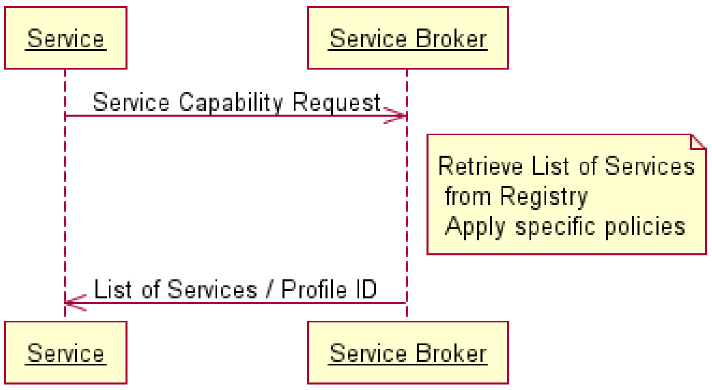

Service
Service Broker
Service Capability Request
Retrieve List of Services from Registry Apply specific policies
List of Services / Profile ID
Service
Service Broker

Figure 4.12: Sequence chart SCM

Whereas the Silver profile comprises only address book management, Instant Messaging, and presence, the Gold profile provides additionally SMS, location, and 3^{rd} party call control for VoIP-only networks. The Platinum profile contains all services without limitations. The widget application implements the SCM to explicitly display the different profiles by using adaptive Graphical User Interfaces (GUI).

4.8.1.2 Policy-based Enforcement of Resource Constraints

We have defined in section 3.4.2 resource constraints as constraints for resources that are required to perform a certain activity. This example illustrates how the Ser-

[9] http://www.developergarden.com

vice Broker and policies are used to define and enforce resource constraints during the execution of a service request. Figure 4.11 shows that more than one service enabler may provide the same service for different access networks. This might be reasonable in cases of a large geographic distribution of an operator's network/service domain or for n+1 service redundancy in operations scenarios (e.g., usage of an alternative service enabler during service maintenance). Furthermore, a service provider might also offer web-based communication services for many customers and connects to multiple operator service enablers through the Service Broker that provides a case-by-case request delegation to the corresponding service enabler for business reasons or optimal service request routing [BMS 08]. The following sequence chart illustrates a high-level flow of request and responses for policy-based enforcement of a service enabler selection constraint:

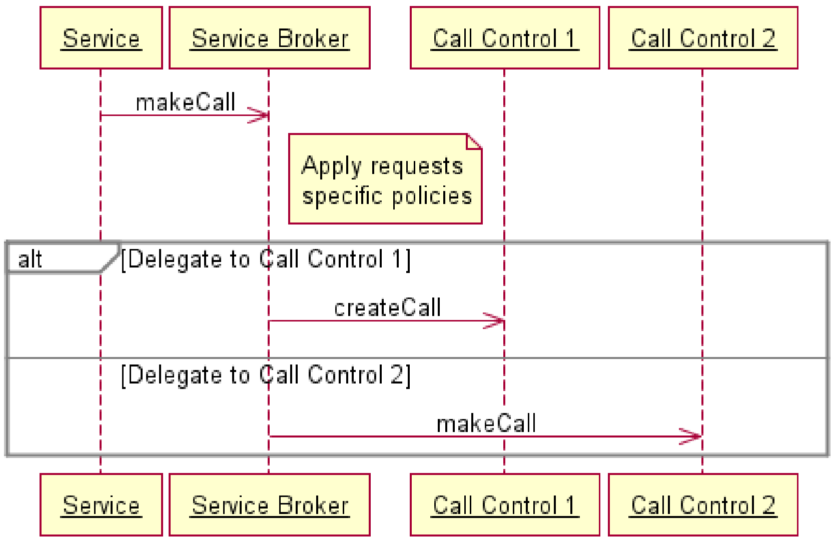


Figure 4.13: Sequence chart policy-based service delegation

The advantage from a service provider/service perspective is that a service needs to implement in this case only one call control provider interface, but the operator may still be able to use multiple call control providers with different APIs to optimise internal request routing or operations. The task enforcing the resource constraint resulting in a delegation of the service request and possible API translation to the appropriate service enabler is performed by the Service Broker. The following listing illustrates a policy for service delegation as a resource constraint:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rule id="r1" xmlns="http://urn/ietf/params/xml/ns/common-policy">
    <conditions>
        <originatorIdentity>
    <one id="alice@serviceprovider.com"/>
```

```
        </originatorIdentity>
        <targetIdentity>
    <many/>
        </targetIdentity>
        <serviceOperation name="makeCall">
    <parameter name="callee">
      <operator name="notequal" operandType="string" match="regx">
          tel:+49</operator>
    </parameter>
        </serviceOperation>
        <validity>
            <from>2008-05-27T14:11:00.943Z</from>
              <until>2010-12-27T14:11:00.943Z</until>
        </validity>
        <constraints/>
    </conditions>
    <actions>
      <invokeService id="a123" namespace="http://www.csapi.org/
          schema/parlayx/thirdpartycall/v2_1/local" name="
          ThirdPartyCallService" url=" http://IP_ADDRESS:PORT/
          opense/ThirdPartyCall/ThirdPartyCallService">
                    <invokeOpReq name="createCall"/>
      </invokeService>
     </actions>
</rule>
```

Listing 4.2: Policy expressing a resource constraint

This policy states that each `makeCall` service request from `alice@serviceprovider.com` to a PSTN number with country code `+49` will be delegated to a specific service enabler using the `createCall` method of this enabler.

4.8.1.3 Policy-based Enforcement of Scheduling Constraints

Policy-based execution of a scheduling constraint is illustrated by enforcing a privacy rule using a presence service through a policy-based composition. Presence is one of the most used services by applications that are based on user interaction and socialization where presence serves as context information. As presence is not limited to the set of know attributes of most Internet Instant Messaging clients as online, offline, away, and busy, but may also comprise personal information as moods, a user or operator may offer the possibility to provide a privacy option. The Parlay X presence service exposes various operations for receiving presence information of certain presentities (subscribers of which presence status is requested). The following list describes the two possible methods for retrieving status info of a specific subscriber:

- Subscribe to the user's presence status (`subscribePresence()`) and then fetch the presence information by calling the operation `getUserPresence()`.

- Subscribe to the user's presence status and then register by notifications of presence status change by calling the operation `startPresenceNotification()`; after the registration is successful the client that initially subscribed for presence status will become a server that is waiting for `statusChanged()` notifications.

The `getUserPresenceResponse()` and `statusChanged()` calls carry inside of the SOAP message the presence status requested. The Parlay X presence service is exposed via the JavaScript/JSON-RPC Presence service as described in detail in section B.8. Users or operators may set up preferences regarding the granularity of information included in these messages as a scheduling constraint.

One example of such as granularity is limitation of the presence information only to basic values, meaning Available and NotAvailable. Mood values as Angry, Bored, Depressed, etc. are transformed to a NotAvailable value. The Service Broker offers this functionality by evaluating and enforcing a policy that requires the transformations of the request attributes, thus representing a service composition enforcing a scheduling constraint. The following sequence chart illustrates a high-level flow of request and responses for policy-based execution of service compositions:

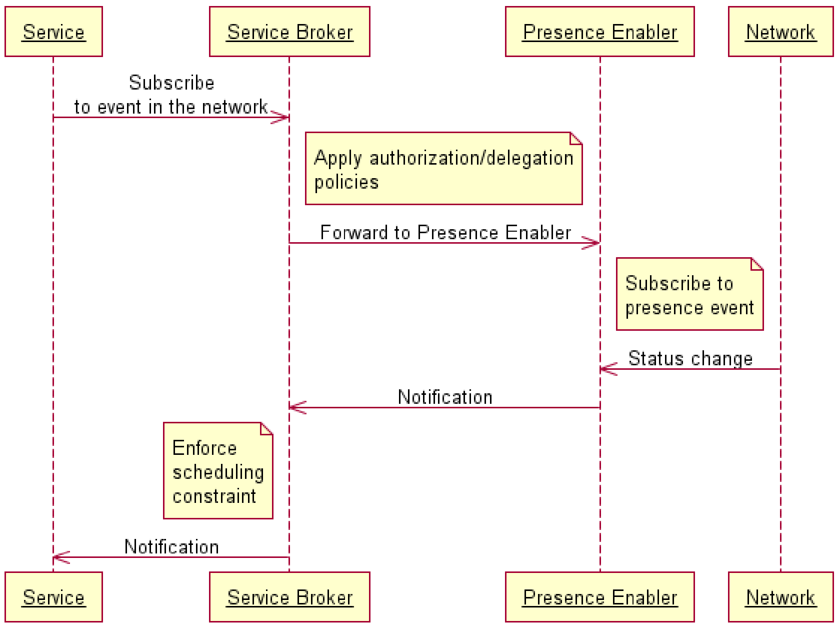

Service
Service Broker
Presence Enabler
Network
Subscribe to event in the network
Apply authorization/delegation policies
Forward to Presence Enabler
Subscribe to presence event
Status change
Notification
Enforce scheduling constraint
Notification
Service
Service Broker
Presence Enabler
Network

Figure 4.14: Sequence chart policy-based execution of a scheduling constraint

The following rule of a policy provides the granularity requested for the presence information. In this case the rule causes all Parlay X activity values to be deleted and the status to be set to ’Off’, resulting that only online/available presence status will be preserved as part of the presence information published.

```
<ruleset>
  <rule>
    <conditions>
      <originatorIdentity>
        <many />
      </originatorIdentity>
      <validity>
        <from>2010-05-27T14:11:00.943Z</from>
        <until>2013-05-27T14:11:00.943Z</until>
      </validity>
    </conditions>
    <actions>
      <if>
        <test>
          <operator name="equal" operandsType="string" match="
              noregx">
            <operand1>fn:contains($result/typeAndValue/
                UnionElement, ’Activity’)</operand1>
            <operand2>true</operand2>
          </operator>
        </test>
        <then>
          <action type="ModifyMsg">
            <attribute name="operation">del</attribute>
            <attribute name="parameter">fn:elementAt($result,
                fn:indexOf($result/typeAndValue/UnionElement, ’
                Activity’))</attribute>
          </action>
          <action type="ModifyMsg">
            <attribute name="operation">set</attribute>
            <attribute name="parameter">$result/typeAndValue/
                Communication/means/status</attribute>
            <attribute name="to">’Off’</attribute>
          </action>
        </then>
        <else />
    </actions>
  </rule>
</ruleset>
```

Listing 4.3: Policy expressing a scheduling constraint

As the message is a SOAP message and thus based on an XML description language, the parameters of the called operation are represented using XPATH relative to the root element of the message body (the element that represents the name of the called operation). For example the element "result/typeAndValue/Activity" contains the presence status like Busy, Away, or moods. The corresponding SOAP message envelops can be found in appendix D.

4.8.1.4 Policy-based Enforcement of Selection Constraints

Policy-based execution of selection constraints is illustrated by enforcing the selection of outbound messaging channels in a messaging session. We use a converged messaging enabler [BLM 09] aiming to support personalized message delivery by taking context information into account to optimize content delivery. The enabler interacts with the Service Broker through the PEM1 interface in order to trigger policy evaluation requests for incoming Instant Messages. According to the provided policy evaluation result, the message enabler will deliver outbound messages through the appropriate messaging channel. In this case the Service Broker is only providing the Policy Decision Point (PDP), the messaging enabler acts as the Policy Enforcement Point (PEP. The following sequence chart illustrates a high-level flow of request and responses for policy-based enforcement of selection constraints:

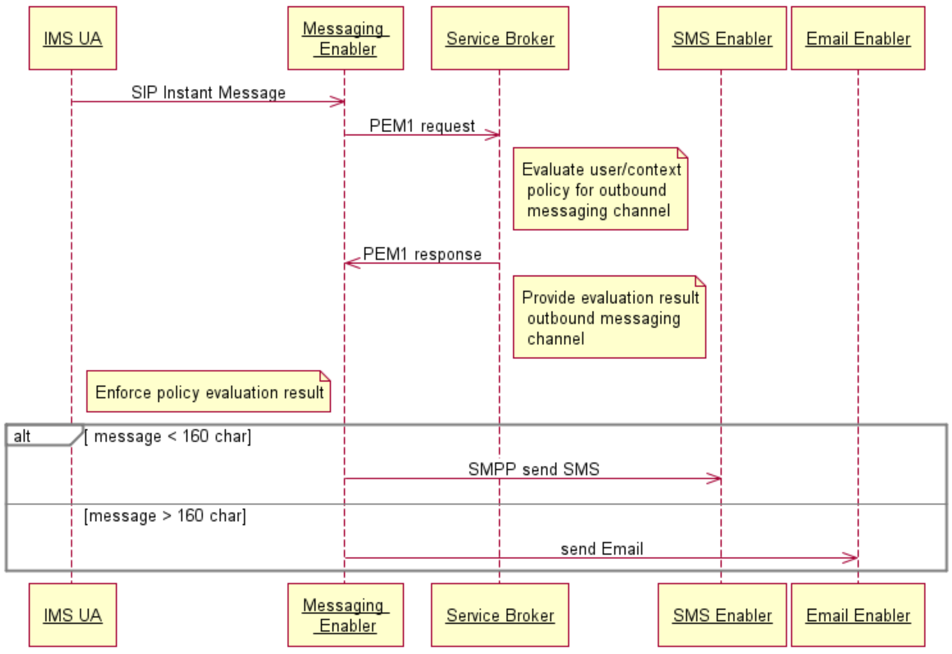


Figure 4.15: Sequence chart policy-based execution of a scheduling constraint

In this example a user Bob receives text messages while he is unregistered to the IMS core network. According to his user policies for Alice, the message will be delivered via the Messaging Enabler via a specific outbound channel determined from the Service Broker. In the first case, Alice sends a SIP Instant Message. Bob receives the message as an SMS (alt 1) because he has created a policy rule for Alice that enables the request to be routed towards the corresponding messaging channel. As soon as the content size exceeds a specified limit (160 characters), the message will be delivered via Email (alt 2) instead. Messages received from other senders as Alice will be deferred by the system (default option).

The following listing provides the necessary policy for this functionality:

```
<rule id="r1">
  <conditions>
     <originatorIdentity>
        <one id="sip:alice@open-ims.test" />
     </originatorIdentity>
  </conditions>
  <actions>
     <if>
        <test>
           <operator match="noregx" name="equal" operandsType="
               string">
              <operand1>$Message-Mode</operand1>
              <operand2>PAGER_MODE</operand2>
           </operator>
        </test>
        <then>
           <if>
              <test>
                 <operator match="noregx" name="less" operandsType
                     ="int">
                    <operand1>$Message-Size</operand1>
                    <operand2>140</operand2>
                 </operator>
              </test>
              <then>
                 <invokeService id="i1" ...  name="AccountService"
                     >
                    <invokeOpReq name="getMobileAccount">
                       <invokeOpReqParameter name="user">
                          $fn:getOriginator()
                       </invokeOpReqParameter>
                    </invokeOpReq>
                 </invokeService>
                 <selectCommunicationChannel name="sms">
                    <originatorId>$i1:mobileAccount</originatorId>
                    <targetId>tel:01243435353</targetId>
                 </selectCommunicationChannel>
              </then>
              <else>
                 <invokeService id="i2" ...  name="AccountService"
                     >
                    <invokeOpReq name="getEmailAccount">
                       <invokeOpReqParameter name="user">
                          $fn:getOriginator()
                       </invokeOpReqParameter>
                    </invokeOpReq>
                 </invokeService>
                 <selectCommunicationChannel name="email">
```

```
45                     <originatorId>$i2:emailAccount</originatorId>
46                     <targetId>email:bob@mailhub.open-ims.test</
                           targetId>
47                  </selectCommunicationChannel>
48               </else>
49            </if>
50         </then>
51         <else>
52            <selectCommunicationChannel name="defer">
53               <originatorId>$default</originatorId>
54               <targetId>$default</targetId>
55            </selectCommunicationChannel>
56         </else>
57      </if>
58   </actions>
59 </rule>
```

Listing 4.4: Policy expressing a selection constraint

This user policy will be evaluated if the originating sender of the incoming message request is `sip:alice@open-ims.test` else, the default path will be applied. Before validating the concrete messaging channel, the first action checks the communication request against the `$Message-Mode`. Otherwise, the deferred messaging functionality will be executed, as well (lines 44-50). If the message fits the configured `$Message-Size`, the Service Broker will evaluate true, enforcing the messaging enabler to use deliver the message via a Short Messaging Service (lines 17-29). The `originatorId` and `targetId` are used by the service broker to perform modifications on the original message for manipulation of data received from the target service. In the case of large messages, the policy evaluation result will cause the interworking selection for Email. In order to obtain the `originatorId`, the Service Broker invokes another service capability (AccountService) and therefore reuses functionalities exposed by other components in the network. The AccountService queries the shared user-profile of the originator on the XDMS for communication addresses related to Email. The related PEM1 message request and response payload can be found in appendix E.

4.8.2 Modelling Hybrid Communication Services

This validation use case focuses on interactions between the Service Broker and a developer making use of the MDSD-based SCE jABC[10] to declare and enforce constraints during the service creation and execution process. We have applied the open API-based meta-model approach depicted in section 3.3.3.1 by implementing Parlay X Service Independent Building Blocks (SIB) and a role model provided by the Service Broker (see section 4.3.3) for jABC.

[10]http://jabc.cs.tu-dortmund.de

jABC implements the MDSD approach and bridges IT thinking with a management and user-level perspective aiming to overcome the capability and perspective gap between the two worlds. It offers a fine granular, feature-oriented service paradigm of modelling processes. We challenge the conventional service definition practice in the telecommunications sector by applying the pluralistic and interface supporting jABC approach that contrasts with the logic of traditional tight and isolated telecommunications service definition, which paradoxically enhances organizational inertia, static routines, and inefficient path dependencies. The following figure 4.16 illustrates the structure of an MDSD-oriented SCE provided by jABC. SIBs are grounded to services, e.g., Web services, CORBA components, or any other software components via APIs, thus enabling hybrid mash-ups [BM 10]. User-level service aggregations are called features. They are useful and flexible units for reusing purpose-specific orchestrations. Applications are service orchestrations realized in terms of features and of SIBs, they are modelled as Service Logic Graphs (SLGs).

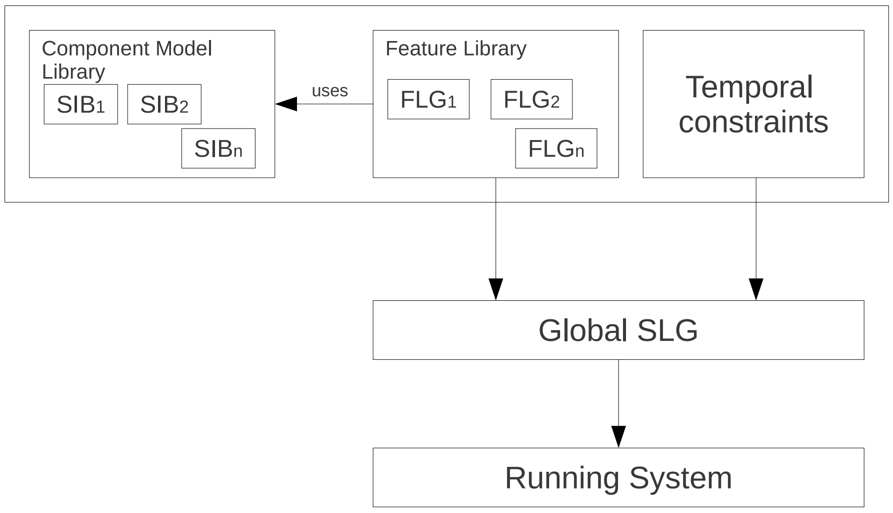


Figure 4.16: MDSD principles in jABC

Temporal constraints are declarative process-level business rules, that can be defined independently of the concrete service and serve as a background knowledge basis against which services can be validated. In jABC, this validation is automatic and happens by model checking. The SLG of an application is therefore built as a service and feature orchestration, it can be easily validated for conformance through a library of temporal logic constraints that are typically application domain specific.

Features were traditionally understood as local modifiers of the basic service: they were individually executed, i.e. a single feature was triggered by some event, executed, and it returned upon termination to the basic service. This is no longer sufficient. In order to account for complex evolutions of services, we allow in today's service description a multilevel organization of features, whereby more specialized features build upon the availability of other, more basic functionalities. In this context, features are also service components from a different domain that might be triggered from a telecommunications perspective by incoming events as calls or messages triggering a service enabler from another service provider's domain, e.g., a Web 2.0 service enabler as maps and/or information sources. The consequence is that features need to be integrated into SCEs that are not part of the service provider's domain but available via remote procedure calls on the WWW. This also means that during run-time of complex services involving non-operator enabler, application misbehaviour might occur through feature triggering by non-communication services outside of the operator domain.

4.8.2.1 Role-based Service Development

We have implemented a Parlay X library for jABC [BMK+ 09]. Each Parlay X based building block (interface) comprises operations for a specific telecommunications related functionality and consists of several operations that are available through a Web Service. Each Parlay X building block is represented by one or more SIBs providing the underlying enabler functionality in an abstract manner for service modelling inside jABC. The following table 4.22 lists the implemented Parlay X services offered by the Service Broker in jABC:

Part	Building Block	Version	Parlay X Gateway	jABC	#SIBs
1.	Commons	2.1 & 3.0	complete	complete	0
2.	Third Party Call	2.1	complete	complete	4
4.	Short Messaging	2.1	partially	partially	7
5.	Multimedia Messaging	3.0	partially	partially	9
9.	Terminal Location	3.0	complete	complete	6
11.	Audio Call	2.1 & 3.0	complete	complete	11
12.	Multimedia Conference	3.0	complete	complete	9
13.	Address List Management	2.1	complete	complete	19
14.	Presence	3.0	complete	complete	12

Table 4.22: Functional mapping Parlay X / jABC SIBs

To reduce the above described risk for the operators, we have applied the SCM of the Service Broker as discovery and role definition interface to jABC. As jABC is intended to be used by experts and non-experts for service creation, we have considered dividing jABC users in the following role categories:

- Developers – expert and non-expert users having access to Parlay X and basic jABC support SIBs according to applicable policies. Each user has his own workspace created at the first login and all his services - meaning the graphs and SIBs created in jABC - are being stored there. This category of developers has access to a user-specific workspace including SIBs for Parlay X services and supporting SIBs.
- Operator/system administrator – this user has a special role and rights, meaning that he has access to all Parlay X SIBs and services for service creation.

The following figure 4.17 depicts the role of developers in the context of the Service Broker as part of the operator domain and jABC for service creation:

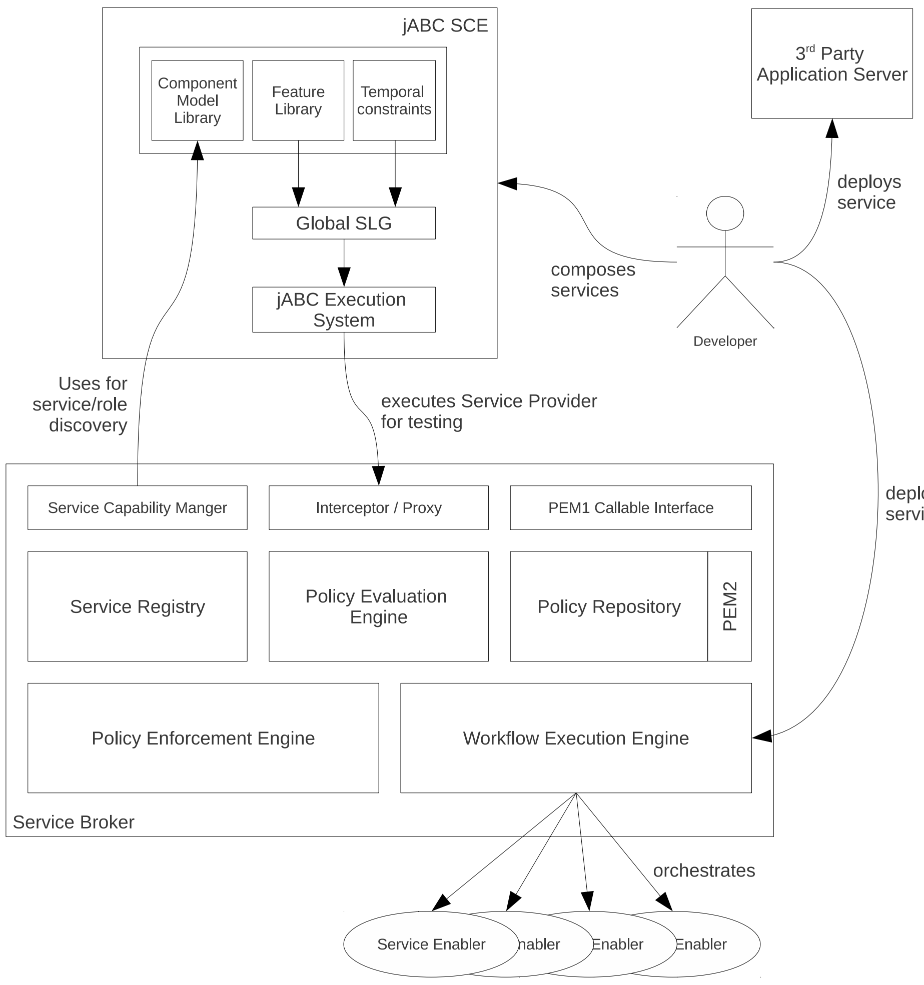


Figure 4.17: Developer / Service Broker / SCE interaction

The developer is able to use a jABC plug-in for a user/role-based service retrieval, based on policy evaluation. Different roles/profiles may be defined through a profile specific authorization policy analogous to the profile definition described in section 4.8.1.1 and [BMM 09]. Listing C.1 in appendix C shows a policy that allows only access to address book, messaging, presence, and SMS by defining constraints on requested services during service discovery at the Service Broker. The following figure 4.18 shows a screenshot of the login feature in jABC:

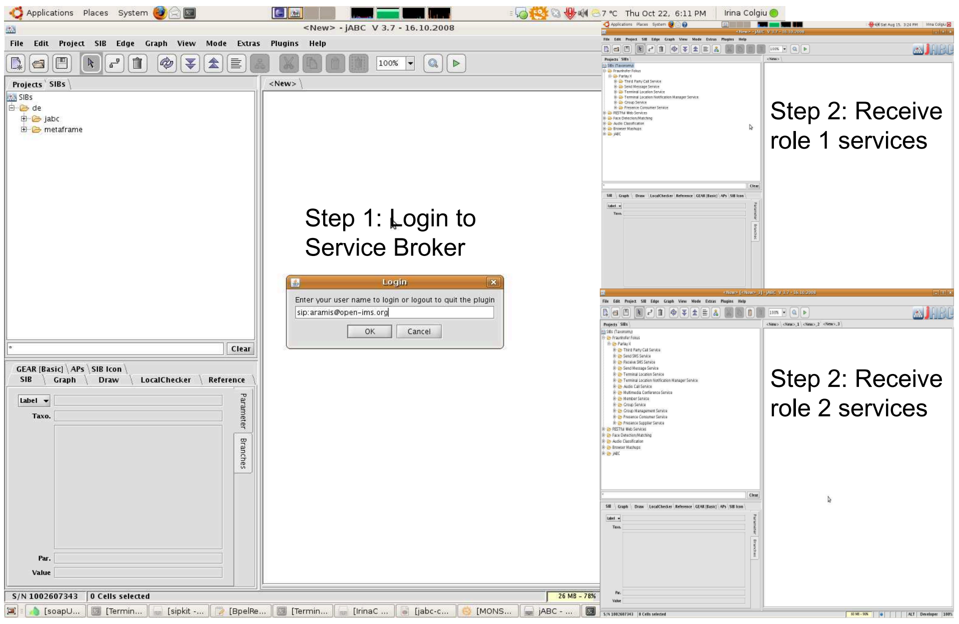


Figure 4.18: Broker-based Service Discovery in jABC

4.8.2.2 Modelling of Constraints

Events indicate a state change of the world. They are n-tuples containing an arbitrary number of data items. For instance, a *PublishPresence* event contains data related to a user, e.g., the user ID, presence information as well as a timestamp denoting the occurrence time of the publish event.

Let $\mathbb{T} = (T, \leq)$ be an ordered time domain. Then, let $\mathbb{I} := \{[t_s; t_e] \in T \times T \mid t_s \leq t_e\}$ be the set of time intervals with t_s as start and t_e as the end time-point of the interval. Let $\mathbb{D}$ be the set of atomic values, where atomic values are elementary data types, such as strings. Then, let $\mathbb{E} := \{(k_1, ..., k_n, k_n + 1, t_s, t_e) \mid k_i \in \mathbb{D}, [t_s, t_e] \in \mathbb{I}\}$ be the set of events on which constraints can be applied.

An event is characterized by the time of its occurrence, which is stored as the event's timestamp. Instantaneous events are single events which occur at a certain point in time. They have a duration of zero, $t_s = t_e$. The aforementioned

PublishPresence event is an example of an instantaneous event. Complex events describe the occurrence of a certain set of events (instantaneous or complex) having relationships defined using logical and/or temporal operators. These operators commonly include conjunction, disjunction and negation as logical operators and can include temporal operators such as *before* and *after* to define the order of events [WBG 08].

Defining events and verifying modelled process constraints for events requires the transformation of services into mathematical objects on which formal proofs are able to be carried out [MS 09]. Complex, orchestrated services are modelled in jABC using Service Logic Graph (SLG)-based models expressing the orchestration of heterogeneous service end points. SLGs are semantically interpreted as Kripke Transition Systems (KTS), a generalization of both Kripke structures and labelled transition systems that allow labels both on nodes and edges of a service graph. Nodes in the SLG represent activities (or services, or components, depending on the application domain). The edges directly correspond to SIB branches; they describe how to continue the execution depending on the result of the previous activity and express process constraints. More formally, a KTS is defined in the following.

A $KTS(V, AP, Act, \to)$ consists of:

(1) a set of nodes *V*
(2) set of atomic propositions *AP* describing basic properties for a node
(3) The interpretation function $I : V \to 2^{AP}$ specifies which propositions hold at which node
(4) A set of action labels *Act* is used to designate the edges
(5) The possible transitions between nodes are given through the relation $\to$ and is contained in $V \times Act \times V$

SLGs may be seen as KTS including atomic propositions and actions. Specifications of a model can be defined using appropriate formalisms as temporal logics, e.g., CTL (Computation Tree Logic) [CGP 00] or modal μ-calculus [K 82].

Selection constraints may be defined in jABC for enforcing verified service behaviour of communication services. An example for a selection constraint is the following:

$$\begin{array}{ll} \neg\ \text{Call U} & (\text{TerminalIsRegistered} \\ & \hat{}\,\text{GetPresence==“online/available”} \\ & \hat{}\,(\text{GetPresence==“online/busy”} \Rightarrow \text{SendMessage})) \end{array}$$

In this formula, Call, TerminalIsRegistered, etc. are atomic propositions that hold in particular nodes of the model, while U is the until operator. It states that a call can only be terminated if the callee's terminal is registered to the operator network and the presence status is “online/available”. If the presence status is “online/busy” a message should be sent.

jABC provides a graphic environment for the composition of SLGs and definition of temporal logics, depicted in the following figure 4.19:

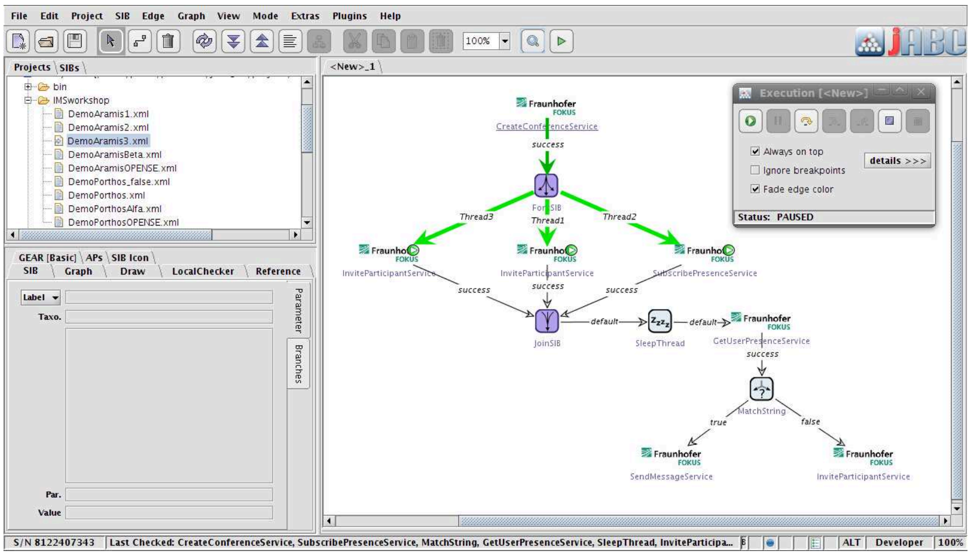


Figure 4.19: Model-checking in jABC

These models can be validated during execution time in jABC by the LocalChecker and GEAR plugin as depicted in the above image as edges of the graph visualize the successful execution of a Parlay X service. Temporal constraints can therefore be provided by the operator to service developers through pre-defined models to be included in user-created service compositions. Model-checking of the user-defined models allows then the verification of process and activity level constraints during service composition.

Composed services in jABC are generated automatically as SOAP-based Web Services. This allows the operator to model temporal constraints as jABC-composed services that are deployed to the Service Broker’s workflow engine and are applied during service execution time resulting in selection constraint enforcement.

4.8.3 Performance Analysis

The performance analysis of the developed solution is separated into two sections. The first section illustrates the performance of policy evaluation of the Service Broker, the second part focuses on the evaluation of different composition languages. As service delegation to composed services expressing scheduling, selection, or resource constraints is a central concept of the Service Broker, we have evaluated the performance of different composition languages classified in section 3.2.3.

The SuT is based on a VMWare ESX[11] virtualization system running on 3 IBM x3650, Typ 7979 B3G machines. Each system consists of 2x Intel Xeon E5420@2.5Ghz Quad-Core CPU with 48 GB RAM. The ESX virtualization share for the Service Broker has 4 CPUs, 4 GB RAM and runs Debian 5.0 OS.

4.8.3.1 Policy Evaluation

We have used SIPNuke[12] as a load generation tool to execute a set of pre-defined tests for the Service Broker. The first test illustrates the performance of a policy evaluation process initiated through the PEM1 interface for a ThirdPartyCallService to have an isolated view on the overhead caused by evaluation. The following figure 4.20 depicts the behaviour of the service broker for up to 260 requests per second (req/s):

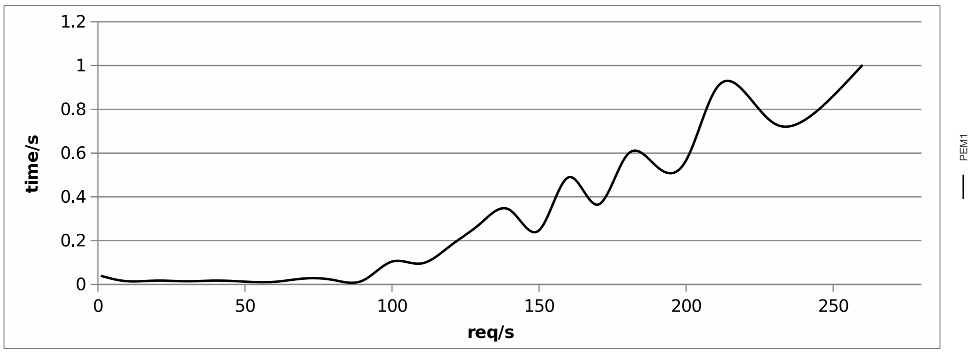


Figure 4.20: Service Broker PEM1 interface load test

The policy evaluation process needs an almost constant execution time of approx. 19 ms until a rate of 90 req/s. The processing time increases until 260 req/s linearly to 1 s.

The following two test cases illustrate the usage of the Service Broker in intercepting mode, therefore the measured execution time includes besides policy evaluation and enforcement also the time of the execution of the service enabler itself. The test cases are the same as in section 4.7, each stress tests started at a 1 req/s - 260 req/s with a 10 s sleep between each incrementation. The JavaScript/JSON-RPC API has not been used to reduce possible overhead caused by the translation of this API to Web Services. The following figure 4.21 depicts the behaviour of the

[11]https://secure.wikimedia.org/wikipedia/en/wiki/VMware_ESX
[12]http://www.sipnuke.org/

service broker for the authorization of a Parlay X ThirdParyCallService compared to the execution of the service without request interception of the Service Broker:

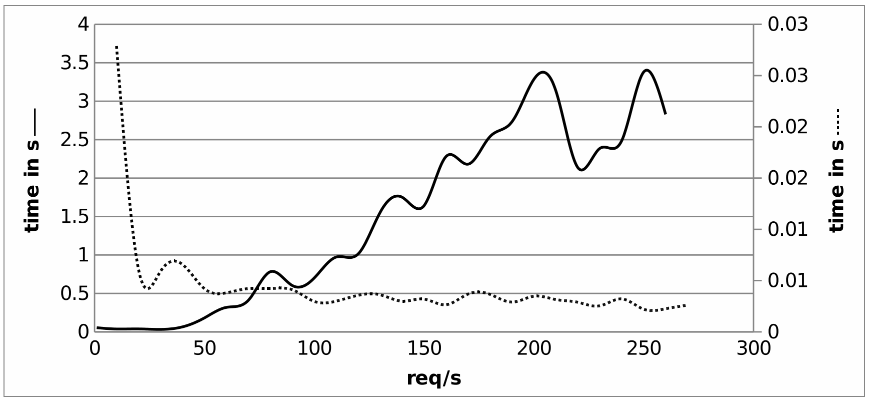


Figure 4.21: Service Broker ThirdPartyCallService load test

Until 40 req/s the Service Broker needs approx. 30 ms for the authorization process of a request. The processing time increases until 200 req/s linearly to 800 ms.

The following figure 4.22 depicts the behaviour of the service broker for the enforcement of a policy-based composition of a GetUserPresence request part of the Parlay X PresenceConsumerService compared to the execution of the service without request interception of the Service Broker:

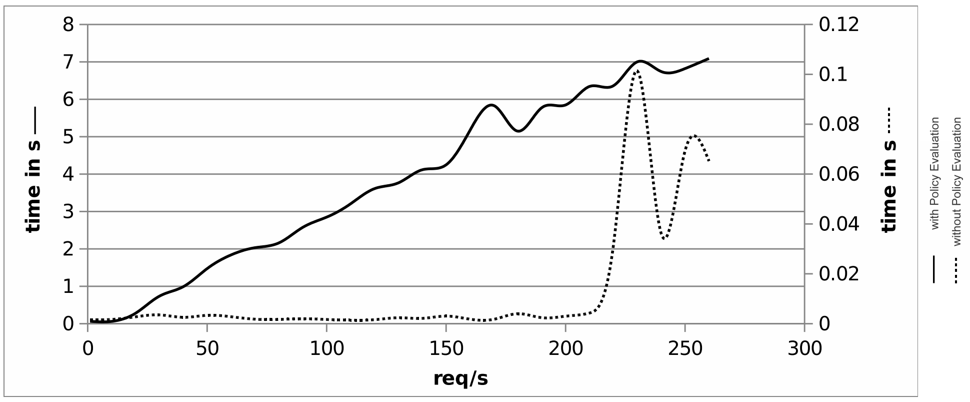


Figure 4.22: Service Broker GetUserPresence load test

The Service Broker needs until 20 req/s approx. 20 ms for the execution of a manipulation of presence parameters. The processing time increases then until 260 req/s linearly to 7 s, which is caused by the limitation of the defined virtualization.

This result shows that complex operations inside a policy may become quite expensive in means of computing power and an alternative approach should be selected in this regard. We therefore test also composition languages that allow the execution of such services on a different level than within a policy, making instead use of a delegation mechanism. The following subsection depicts the performance of selected service compositions.

4.8.3.2 Execution of Service Compositions

Following to the theoretical analysis of composition languages and models in section 3.2.3, we provide in this section a performance analysis for the following classification candidates:

- BPEL
- SCXML
- SLG

BPEL was chosen because of the widespread usage and the standard description language for the Web Service processes used. SCXML has been selected for its good adaptability, scalability and simplicity characteristics. SLG is the third candidate as it provides similar to SCXML the capability to compose heterogeneous services consisting of different APIs and the ability to generate executable Java code with the help of jABC. Two independent test scenarios are presented in this work that compares between BPEL and SCXML as well as between BPEL and SLG.

BPEL vs. Java vs. SCXML Test Scenario
For the performance analysis between SCXML and BPEL the frequent orchestration of a simple SIP Instant Messaging service is assessed. The performance measurements are related to the service invocation:

a) in Java
b) with SCXML
c) with SCXML triggering the service via a Web Service
d) with BPEL (triggering also a Web Service)

The invoked service sends a simple SIP MESSAGE request to a SIP application server that replies with a 200 OK. The implementation of the service is the same for every approach. Regarding a) the service is executed directly using the Java method implementing the service. In case of b) a local Java method has been invoked directly by the SCXML Engine. At last c) and d) need to call the Java method via Web Services inside the SCXML respectively the BPEL script. The following diagram 4.23 shows the time in milliseconds for every approach needed to execute in relation to the frequency of service invocations:

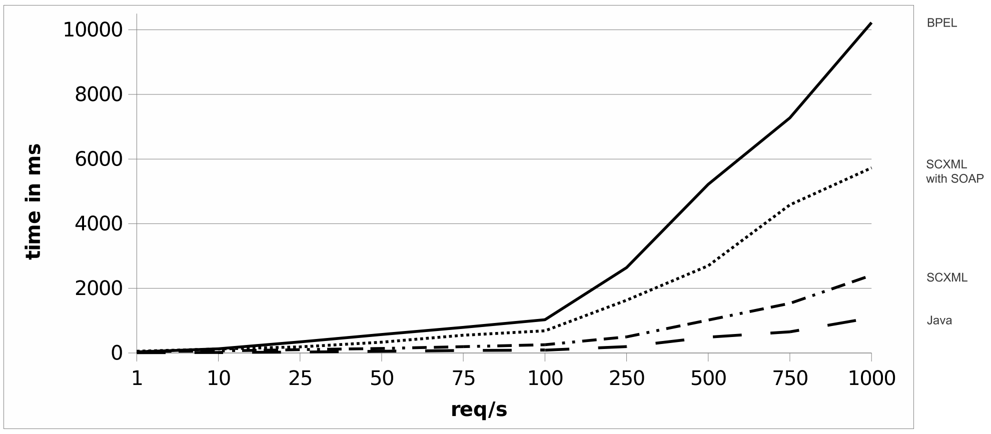


Figure 4.23: Service Composition Execution Tests: BPEL, Java, SCXML

For this experiment two machines are used. The first machine executes the service via the Apache ODE BPEL Engine[13], the Apache Commons SCXML Engine[14], the Web Service and the Java implementation. All services are deployed in a Jetty Web Server[15] in order to have comparable results. The second machine deploys the SIP Message Servlet, which gets the SIP Messages and sends back a 200 OK to avoid retransmissions disturbing the performance measurements.

The test shows, that as expected the Java service has the highest performance. We took Java into account due to two reasons:

- SLGs may be grounded to Java by code generation,
- to measure the computation overhead caused by the SCXML engine to execute the script in case b).

Applying a higher invocation frequency, the execution time of SCXML is approximately 2.5 times higher than the native Java execution. Furthermore we wanted to get a picture of the overhead caused by using Web Services instead of native Java method calls in the SCXML engine. The performance decreases rapidly in case c) in particular for higher frequencies. The same applies for the last test case using BPEL.

[13]http://ode.apache.org
[14]http://commons.apache.org/scxml
[15]http://www.mortbay.org/

It should be noted that the orchestration overhead in SCXML is lower than the overhead in BPEL, even when using Web Services. Nevertheless this overhead in combination with Web Service calls always results to the lowest performance.

BPEL vs. SLG Test Scenario
Another performance analysis compares the execution time of a more complex composition scenario using BPEL and SLG defining a selection constraint. The service receives address book information of a user from an OMA XDMS service enabler and invites users to an audio conference. Depending on the profile information stored at the XDMS, users receive a gender-dependent audio jingle at the beginning of the conference. All service end points of the composed service, namely address book access, conference management, audio call for jingles are SOAP-based Web Services. The service execution environments are also Apache ODE for the execution of BPEL scripts and Java-based services generated by the jABC framework from SLG defined service compositions. Figure 4.24 depicts the results of the performance measurements:

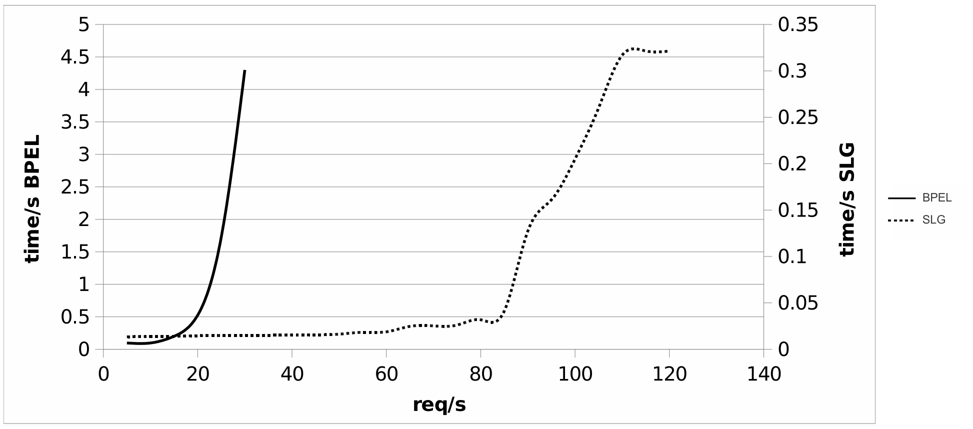


Figure 4.24: Service Composition Execution Delay Tests: Java/SLG, BPEL

It shows that services generated by jABC support a much higher rate of requests per second without returning errors - around 120 requests per second while BPEL can only manage around 30 requests per second in the same test conditions and environment.

Relevant for our analysis is also to compare the error rate obtained at the threshold value of the request rate for both jABC and BPEL composition strategies. In this case, an error means that the client receives a time-out error from the server. The following figure 4.25 depicts the results of this performance measurement.

During this test, the error rate increases directly in relation with the growth of the rate of requests per second. The errors in jABC/SLG-based composition appear later, approximately at a rate of 90 req/s and the gradient is growing slowly, regarding the BPEL-based composition strategy, the error appears quite early at a rate of 25 req/s. The error rate increases quite fast, reaching a 100% error rate at 35 req/s.

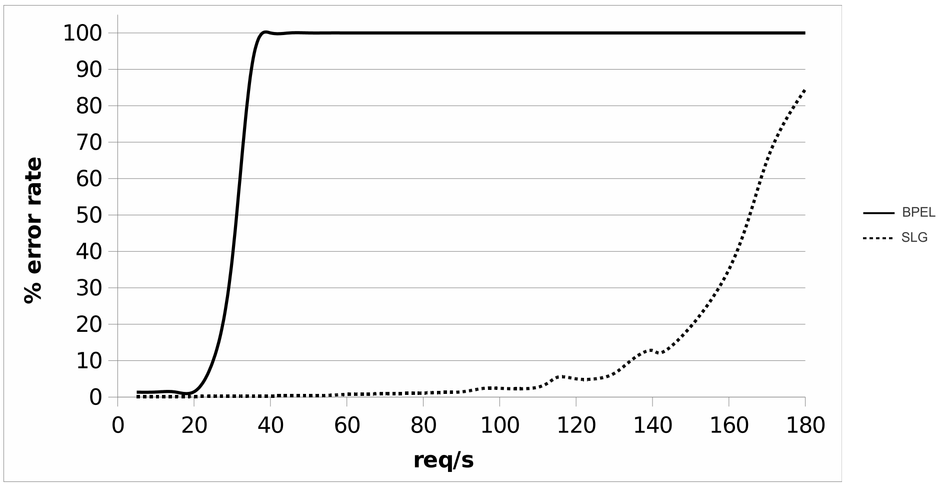


Figure 4.25: Service Error Rate Test: Java/SLG, BPEL

Concluding, it can be stated that compositions implemented in Java or grounded through code generation to executable code are much more responsive and performing than composition created and executed in workflow engines. SOAP-based Web Services create a measurable overhead caused on the one hand by network communication, but interestingly on the other hand by the SOAP stack itself (e.g., due to the required XML parsing).

4.9 Conclusions

In chapter 4, we introduced an abstract information and data model providing the underlying basis of our Service Broker reference architecture. From this model, we derived a general policy taxonomy and defined a policy formalism. Together, this builds the theoretical basis for our policy-based Service Broker reference architecture. The central part of this architecture are policy evaluation and enforcement functions. Our Service Broker serves all actors in a cooperation of operators, 3^{rd} party service providers, service developers, and users to express and enforce service related preferences as user privacy, service profiles or operator preferences for optimisations in operations and service routing. Service developer interests are addressed by the definition of a high-level Telecom API based on Javascript/JSON-RPC and the tight integration of a service discovery interface into the MDSD-based jABC environment.

As current communications service infrastructures enable in the end real-time communications as voice or video, special efforts have been conducted to assure minimum processing overhead by the policy evaluation process. We have discussed the complexity of two algorithms to motivate our selection of the Rete algorithm for policy evaluation. Furthermore, we have done a set of performance tests, illustrating the performance of the Service Broker and the policy evaluation function under different conditions. A certain trade-off in performance is in our opinion acceptable compared to the increase in flexibility gained to define the behavioural part of services during execution time. One major finding of the performance tests is that complex functionality applied to service requests should not be performed within a policy composition due to heavily increasing computation time, but should be delegated to a specific service providing this isolated functionality instead.

The definition and enforcement of temporal constraints is illustrated by three validation use cases on different levels of service execution and service creation. We motivate the application of model-driven definition of services implementing constraint enforcement due to model-based service verification, on the one hand, to assure proven service behaviour and, on the other hand, a better performance achieved by code generation in contrast to workflow execution by orchestration engines or policy-based enforcement of complex constraints. This decision is also supported by a set of performance test that we have conducted using two different workflow expression languages and corresponding engines and code generated by the MDSD-based jABC SCE.

Chapter 5

Comparison with other Approaches

Service Brokering in SOAs is a wide research field. A lot of different research work has already been published to propose a middleware function, abstracting from network resources and enabling 3^{rd} parties through APIs. In this chapter we survey different recent middleware approaches from other researchers that are in scope with our work and have been published in scientific literature. We compare these to our own solution as specified in chapter 4 defining different criteria on which we base the comparison and summarize the evaluated approaches.

5.1 Evaluation Criteria for Telecom Service Broker Systems

Multiple service infrastructures have been proposed for NGN and converging networks, either with focus on feature interaction between services on a platform or interaction between different domains and corresponding platforms. For comparative purposes we will evaluate these methods using different criteria as they target different networks and use cases, use different internal mechanisms or have performance differences. We define two main criteria groups: *architecture-related* and *functional* criteria. The former is related to the design of a solution. It covers IT and service engineering principles, including set-up and performance. The latter covers the functional perspective of the designed solution.

5.1.1 Architecture-related Evaluation Criteria

Network Agnostic
Description of network abstraction principles and technologies, e.g., what kind of API technology used, specific for a certain protocol or network technology.

System Governance
Details about the defined system governance approach, e.g., similar to the specified policy-based approach in the previous chapter, if any.

Platform-specific
Description of the architecture with dependencies to a specific platform technology, e.g., JAIN SLEE.

Implementation
Details about the actual implementation, if any. This includes the programming language and frameworks used.

Performance Test Results
If an implementation exists, we will present performance results as described by the authors. Noteworthy features include processing latency, e.g., the delay services/users encounter, how many messages the solution can process in 1 s. Note that these results cannot directly be compared between different implementations – there are too many variations between test set-ups, test configurations and testing hardware.

5.1.2 Functional Evaluation Criteria

Composition Capabilities
Described capabilities of a solution to define and execute composed services, e.g., defined as a workflow.

Feature Interaction
Description and development of certain feature interaction mechanisms allowing feature invocation between services and domains.

Integration with SCE
Details about service creation principles required to develop extensions and services for a platform, if any. Description of possible integration and interactions with a service creation environment.

Service Exposure
Description of service exposure approach to offer services to 3^{rd} parties for re-use in e.g., mash-ups.

5.2 Survey of Service Broker Systems for Telecommunications

In this section we present related work that has been proposed by various researchers. The approaches are presented in order of publication. Each proposal is summarised according to the aspects of the evaluation criteria introduced above and separated into an architecture-section and a functional-section.

We base this summary on available facts from the publication and do not judge any of the proposed ideas here. A comparison and discussion of the presented ideas follows in a later section. There are already several commercial solutions (e.g., Amdocs[1], IBM[2], Oracle[3]) targeting similar. However, information on and specifications of these systems are restricted, and thus cannot be evaluated here.

5.2.1 Khan07

Authors of [KGL 07] propose an extension to the IMS service layer to broker and integrate SIP, HTTP, and Real Time Streaming Protocol (RTSP) service delivery in the application layer.

Architecture Description
The main proposition is an application policy function including related functions of IMS, RSTP, and HTTP networks. One option for the realization of such a function named is a SCIM. The proposed application policy function makes use a protocol specific extensions to blend services. Specific policy evaluation and enforcement points are not described, neither is a specific target platform mentioned nor are implementation results described.

Functional Description
Based on the description of the application policy function, service invocation and service blending seems to be based on static behavioural descriptions inside the proposed application function, but a detailed description is missing. Feature interaction between different services is mentioned, a specific algorithm or sequence diagram of the signalling part is not provided. The exposure of services for further re-use or integration into a SCE is out of scope of this work.

[1]http://www.amdocs.com/Offerings/CES-Portfolio/Service-Delivery/Convergent-Service-Platform

[2]http://www.ibm.com/software/industry/telecommunications/

[3]http://www.oracle.com/us/industries/communications

5.2.2 Belaunde08

Authors in [BF 08] propose an approach for developing composite telecommunication services running on mobile phones which takes advantage of the use of model driven techniques as well as the loose coupling paradigm in SOA. The presented results are based on the EU FP 6 project SPICE[4].

Architecture Description
The paper describes an SCE that is connected to a repository, to import and export the definition of running services and a life-cycle manager, to deploy and activate services in a execution environment. The model-driven SCE allows the composition of back-end BPEL services and GUI front-ends on mobile devices. The implementation makes use of available tools to generate from meta-models executable Java or Python code. The SCE is connected to a repository, to import and export the definition of running services and a life-cycle manager, to deploy and activate services in a execution environment.

Functional Description
The SPICE project has defined a high-level and executable language for describing composite telecommunication services. This formalism, named SPATEL (SPICE Advanced language for Telecommunication services), can be considered as a customization of the UML language for expressing the definition of service interfaces and service composition logic that is well-suited to the Telecom domain. The model transformations and the code generators are implemented using two alternative techniques: firstly through direct usage of the meta-modelling APIs generated from the platform-specific meta-model, and secondly using an OMG MOF-based model (see section 3.3) transformation language. SPATEL may be also transformed to BPEL providing back-end services that are exposed via SOAP. Front-end services on devices are grounded to Python to be executed on a mobile device. Feature interaction is based on Web Services.

5.2.3 Kirchberg08

Authors in [XTT+ 08] propose an integrated Telecom and Internet SDP for interworking between IMS and Web Services. The authors propose a SIP-based Micro Service Orchestration and Web Services bus to integrate Web Services, IMS services, and underlying network resources.

Architecture Description
Main focus of the work is a SIP-based micro service orchestration layer. It may

[4]http://www.ist-spice.org/

integrate protocol specific modules for CAMEL or Parlay services, generic Web Services, SIP, and Diameter. A policy manager is mentioned, it contains user-defined policies and provides and enforces these policies on a per-service session basis. A more thorough explanation is missing. The architecture is mapped to a JAIN SLEE platform, details about the implementation and related performance tests are missing.

Functional Description
The basis of the proposed environment is a so called Service Provider Deliver Environment (SPDE), a functional framework for developing, deploying, and delivering service functionality. It includes a service broker acting as an intermediary between a service provider and a service requester, and a service orchestration function. A detailed description of these functions, used algorithms and standards is missing. Feature interaction between services in the SPDE is addressed, as well as service exposure.

5.2.4 Bond08

Authors of [BC 08] propose a light-weight, Java-based framework for converged Telecom services and mash-ups. The framework supports interactions between a specific feature and external services, it can be utilized to support Telecom service mash-ups as customized web services that incorporate Telecom services with other services.

Architecture Description
The proposed service environment is based on a SIP Servlet 1.1 container, allowing the integration of SIP and HTTP-based service features. Composition capabilities are limited to the container's application router. A BPEL engine provides orchestration capabilities for Web Services.

Functional Description
The core of the defined framework is based on Java-to-SIP and SIP-to-Java feature interfaces that may be encapsulated by SOAP interfaces to use Web Services orchestration. Composition of features is available by using the graphical ECharts[5] framework for SIP Servlets supporting the notion of reusable state machine fragments: reusable parameterizable state machines that perform common functions. Convergence of Telecoms and IT is achieved by defining SOAP-based Web Services for Telecom features.

[5]http://echarts.org

5.2.5 Moro09

Authors in [MBO+ 09] present the results of a study on the use of SCXML for the real-time composition of network-centric services.

Architecture Description
The focus lies on the provisioning of a SCIM function for IMS services, a specific platform technology is not mentioned, neither are implementation details or performance evaluations depicted. The proposed architecture is based on a session control abstraction layer for IMS that may be used by the SCXML engine to orchestrate SIP-based services and IN/CAMEL services.

Functional Description
Composition capabilities are addressed by the authors based on SCXML workflows to provide feature interaction between the SIP-based Telecom domain and other brokered services (e.g., charging, IN services, other SIP services). The authors describe a use case of a service provider exposing service composition rules for Mobile Virtual Network Operators (MVNO). Details of rules and related processing are missing.

5.2.6 Matsumoto09

Authors in [MHT 09] focus on a middleware platform for Fiber-To-The-Home (FTTH) and metro Ethernet networks to manage large scale network resources.

Architecture Description
The architecture is based on a SOA middleware for NGN and legacy networks. A BPEL-driven policy manager is proposed in order to enforce global QoS policies on both NGN and legacy infrastructures. This middleware belongs logically to the network control layer and serves IMS-only services. Details about implementation-specific aspects and related performance tests are missing.

Functional Description
The proposed policy management is based on workflow-based policy rules, transaction status, and resource conditions. In accordance with the policy decision workflow, the policy manager enforces policies on network resource managers through a specific network resource management interface based on SOAP. Interaction of features or accessibility of network services for foreign domains is not addressed. The creation of policies or services is out of scope of the work.

5.2.7 Bauknecht09

Authors of [BHK 09] propose a flexible decision-making mechanism for a personalized communication controller in IMS for multi-device/multi-session use cases.

Architecture Description
The main architectural focus is based on the XACML policy framework and specific extensions for obligation policies to be used for service personalization. The underlying session control protocol for services is SIP and the used target platform is based on JAIN SLEE. The policy engine acts as a SIP B2BUA towards SIP UAs and realises service feature control for ongoing SIP sessions. The overall system governance is policy-based and example XACML polices are provided. Details of the implementation and a performance evaluation are missing.

Functional Description
Composition of features through policy enforcement is explicitly addressed by the authors with limitations to a single domain. All feature compositions are expressed within policies allowing a generic personalization of services. The workflow of a composition is considered as static, features are handled as variables. The core of the work is the proposal to support variables as parameters of obligations as an extension to XACML. The creation of compositions is out of scope of the proposed work.

5.2.8 Jin09

Authors in [JPY+ 09] propose a way to shorten and simplify the service creation life-cycle by building a template-based SCE. A model is used to separate service workflow definition and service parameter configuration and to achieve rapid development for different roles.

Architecture Description
The main focus of the work is on service template toolkit. A template represents patterns of communications services with the same or similar logic. The toolkit is an Eclipse[6] IDE plug-in allowing the definition of templates. It may act as a BPEL or SCXML editor and deploy services to corresponding workflow engines as part of a SLEE environment. Specific network technologies and protocols are not addressed by the authors.

[6]http://www.eclipse.org/

Functional Description
The underlying model of the template toolkit is based on template profiles describing business parameters and a template deployment package including Telecom service logic. Business logic may be composed either using BPEL or SCXML. Authors state that service development of a template with their tool is about 2-5 times faster than manual coding of the service. A template provides the flexibility to compose 10-15 services. The description of a specific composition model and feature interaction mechanism is missing.

5.2.9 Baladron09

Authors of [BAC+ 09] are proposing a platform for graphical service creation aimed at individuals with no specific skills in computer science or programming. Services are grounded to a service-oriented execution environment capable of seamless interoperability between Web Services and Telecom applications based on an operator-owned infrastructure.

Architecture Description
The architecture is based on a distributed environment for a SCE, an execution environment, and a base service repository. The SCE can be broken down to a browser-based BPEL designer, as the execution environment provides a BPEL orchestration engine with partner links to so called base services. Base services may be hosted on different platforms within an operator domain. Implementation-related issues are not presented besides naming Web Services as the underlying communication standard.

Functional Description
The platform provides a web portal for service creation, a service life-cycle manager automating tasks as deployment, and a user information manager to manage profiles and context data. Compositions may be created through the web-based, graphical SCE that are grounded to BPEL. It is mentioned that the composition model is asynchronous, allowing interactions with communications services. Integration with synchronous applications is provided by a platform middleware.

5.2.10 Loreto09

Authors of [LMO+ 09] present the concept and implementation a location service broker providing a flexible layer for convergence of Telecom and IT services.

Architecture Description
The architecture is based on a service front-end function providing service APIs as Parlay X, an operator network connectivity layer for connecting to different network capabilities, location-specific utility function providing geographical calculations, and a business logic function. The environment is implemented as J2EE services, performance results are not provided.

Functional Description
A web-based mash-up service is described by the authors integrating location information into a map service and adding local weather information. The service broker itself does not provide composition capabilities, it only exposes user and device location information from multiple sources. Services are developed with a code-based IDE and deployed to the J2EE service environment.

5.3 Comparison of Telecom Service Brokers

The related work we present in this chapter targets different approaches towards converged Telecom and Internet service environments. This section compares the outlined work above with our approach.

	Khan07	Kirchberg08	Belaunde08	Bond08	Moro09
Network agnostic	IP only, SIP, HTTP & RTSP services	IP-based SIP services & legacy IN services	depending on available service enabler	IP-only, SIP & HTTP services	IP-based SIP services & legacy IN services
System Governance	No specific mechanism is described	Policy-based	JSR 289 application router-based & BPEL engine	BPEL-based orchestration	Rule definition in SCXML
Platform	Openwave Multimedia Proxy	JAIN SLEE	J2EE	JSR 289 Java container	JAIN SLEE
Implementation	No implementation details provided	No implementation details provided	Transformation details provided	Java method details provided	SCXML example provided
Composition capability	Possible, no details provided	SIP micro service orchestration	Based on SPATEL, transformation to BPEL, Java, and Python based on service templates	Web Services composition based on BPEL, SIP Servlet composition based on ECharts	SCXML-based orchestration of services
Integration with SCE	out of scope	not described	Integration in GMF/Eclipse framework with graphical editor	ECharts provides a graphical SCE (without model-checking)	out of scope
Service Exposure	out of scope	SOAP Web Services	SOAP Web Services	SOAP Web Services	Parlay X SOAP Web Services

Table 5.1: Comparison of Telecom Service Broker Part I

	Matsumoto09	Bauknecht09	Jin09	Baladron09	Loreto09	Blum10
Network agnostic	IP-only	IP-only, SIP-based services	abstract service templates	API-based network abstraction	API-based network abstraction	API-based network abstraction
System Governance	BPEL-based policies	XACML-based policies	BPEL / SCXML-based	BPEL-based	not described	Common Policy-based
Platform	Platform-independent	JAIN SLEE	SLEE	OPUCE execution environment	J2EE	Platform-independent
Implementation	No implementation details provided	No implementation details provided	No implementation details provided	No implementation details provided	No implementation details provided	Implementation details & performance test results provided
Composition capability	Web Services composition based on BPEL	Policy-based	not described	Web Services composition based on BPEL	out of scope	Policy-based, SCXML-based, SLG-based
Integration with SCE	out of scope	out of scope	Integration with Eclipse IDE (without model-checking)	Browser-based graphical BPEL composer (without model-checking)	out of scope	Model-based graphical SCE (jABC)
Service Exposure	not addressed	not addressed	not addressed	SOAP Web Services	Parlay X SOAP Web Services	Protocol-agnostic, SOAP & JSON-RPC Web Services described

Table 5.2: Comparison of Telecom Service Broker Part II

Chapter 6

Summary

This dissertation has presented service-oriented paradigms applied to telecommunications and IT, discussed large-scale service-oriented systems, and proposed a new policy-based access and system governance mechanism for service delivery platforms allowing optimal service behaviour and control for operators, 3^{rd} party service provides, developers, and users.

In the course of the research for this thesis, we have published one book chapter, ten journal papers, and 18 conference papers dealing with the addressed problems.

This chapter summarizes achievements of this work and describes on-going future extensions of our work in several research projects.

6.1 Conclusions and Impact

In this dissertation, we formally proposed a policy-based Service Broker as an emerging middleware function for converged Telecom and Internet service architectures. We discussed the challenges, both in building service platforms, as well as in interconnecting different service domains to form a service-oriented converged Telecom and Internet service layer based on IT principles. We continued by discussing service evolution in telecommunications from proprietary platforms towards specialized IT-based environments. We described how IT principles have influenced this evolution and presented current trends in IT for distributed services and model-based service creation.

For defining a converged service environment for Telecom and Internet services, we have identified requirements from the perspective of a network operator, service developer, service provider, and user. Based on these requirements we designed an information and data model for the interaction of users, their personas and devices, and Internet services offered by service providers. From this model we derived a general policy taxonomy for service providers and users. Finally, we presented our policy-based Service Broker as a middleware enabling feature interaction between an operator controlled communications-centric service domain, the Internet, and its service overlay WWW. The goal of our platform is to enable cross-domain service composition for the creation of hybrid services and constraint enforcement during service execution and discovery time. We provided details of the underlying policy-based formalism, as well as results from performance experiments showing the ability of the solution to address needs of operators, service providers, developers, and users.

Our Service Broker provides a direct link from business value of a service to its priority in the service enabler layer of the operator; it applies general principles designed by OMA as part of its OSE and compliance to PEM1 and PEM2 interface specification. More important, this dissertation provides the theoretical basis of a policy taxonomy and formalism that is missing as part of the OSE specification and concentrates on interactions between the service environment and service developers in a two folded approach: by defining a high-level JSON-RPC based communications API and a model-driven approach for service and constraint definition.

We have laid down three proposition in the beginning of this thesis in section 1.4 that have been discussed from several technological and usage-related perspectives throughout this thesis:

I Technologies, mechanisms, and service paradigms in the Internet and Telecom follow different principles. Cooperative services spanning across both domains require an intermediate function for mediation between these domains.

II Policy-based service composition allows users, service providers, and network operators to express service constraints in high-performance, low-latency requirements in Telecom systems.

III Software synthesis and code generation based on formal models is a well-suited software development approach for an open Telecom systems.

As part of the state of the art technology and principles analysis in the first one-third of this thesis, we have outlined that technologies and principles from IT are currently adapted to be used in Telecom service infrastructures. Proposition I is based on the assumption that the circuit-switched, IN-based technology will still co-exist for the time being. Regarding the technology shift towards all-IP that is performed with the introduction of the NGN and related service technologies (e.g. SIP Servlets), this proposition gets softened and cannot be sustained that strictly. Nevertheless, requirements on service availability and performance remain the same for legacy and NGN service infrastructures, resulting in a special need for protection and control of arbitrary and malicious service behaviour. This statement supports proposition I and the need for an intermediary control function that we propose by the definition of a Service Broker.

Inside of the defined Service Broker, we use policies extensively to control system behaviour and enforce user, service provider, and network operator defined preferences. We have shown with the help of several use cases and validation examples the feasibility of our approach. During the conducted performance tests we have found out that the execution of complex operations through policy enforcement is creating a significant overhead for the end-to-end service execution time and is in our opinion therefore not applicable in a low-latency, high-performance service environment. Only basic obligations on called services and parameters should be enforced with this mechanism. We recommend that complex operations should be transferred to specific services to which we refer to as delegated services. Therefore, we cannot keep up proposition II to the extend that policy-based composition of constraints is a well-suited approach in any case. This statement needs to be limited to the enforcement of simple obligations to ensure the execution requirements of carrier-grade Telecom systems.

In designing our Service Broker, we have put special emphasis on programmability of network services and resources by defining on the one hand a high-level JSON-RPC API and, on the other hand, the tight integration with jABC, a model-driven software development environment. With the provided capability of jABC for constraint definition based on temporal logics, developers and service designers are able to create service models based on pre-defined constraints. During the discussion of strengths and weaknesses of MDSD in section 3.3, we refer to several studies stating that MDSD has significantly increased the productivity of the software creation process. On the other hand service performance has not been rated as sufficient, e.g. by [BLW 05].

We have found out that generated services perform still much better than interpreted services that are defined using standardized service composition/orchestration languages as BPEL that are currently being proposed for commercial service execution environments for Telecom systems. We can therefore still support proposition III and by implication, based on our tests, state that we highly doubt the success of orchestration languages for near real-time services. Proposition III may only be restrained by the fact that many developers, especially for WWW-based services tend to create their services still by hand, supported with tools for high-level programming languages and script languages instead of using graphical composition tools that do not allow direct manipulation of the code base.

Similar cross-domain, cross-layer service environments are being proposed as part of the EU Future Internet Research and Experimentation (FIRE)[1] initiative towards creating a multidisciplinary research environment for investigating and experimentally validating highly innovative and revolutionary ideas for new networking and service paradigms.

[1]http://cordis.europa.eu/fp7/ict/fire/

6.2 Outlook

Our solution design and its theoretical basis will serve in 2010 and 2011/12 as the foundation for several research projects:

- Design of a beta developer platform for a large German Telecom operator.
- Design of a cross-network, cross-domain information sharing platform for a large Japanese Telecom operator.
- Design of a cross-layer service composition platform as part of the German Federal Ministry of Education and Research (BMBF) G-Lab DEEP[2] Future Internet project.

The following subsections depict each project briefly.

6.2.1 Telecom Beta Developer Platform

The goal of this project is the provisioning of an "innovation window" part of an existing Telecom developer portal. The main idea is based on exhibiting innovative functionalities bearing the potential for future Telecom services to developer communities enabling early feedback and insights for further service evolution. This should allow the operator to leverage better open development for safeguarding future business. The target platform should incorporate services from 3^{rd} parties:

- from developers themselves
- from external service providers
- from R'n'D networks (German & EU public R'n'D projects, e.g. FP 7)

Inside this "innovation window", functionalities will be offered to the developer community as enabling services as part of the platform. The platform will provide an environment for composition of services with workflow languages as SCXML and a graphical composition tool kit. Service enablers will either be provided by the Telecom operator or any 3^{rd} party. A central role has the Policy Engine enabling the enforcement of constraints during service creation and execution time. Composed services may be exposed automatically with multiple different APIs (e.g. REST, JSON-RPC, Web Services, RMI, etc.), depending on the targeted developers and use cases. The following figure 6.1 depicts the extension of our work towards a platform for developers of beta services:

[2]http://www.g-lab-deep.de/

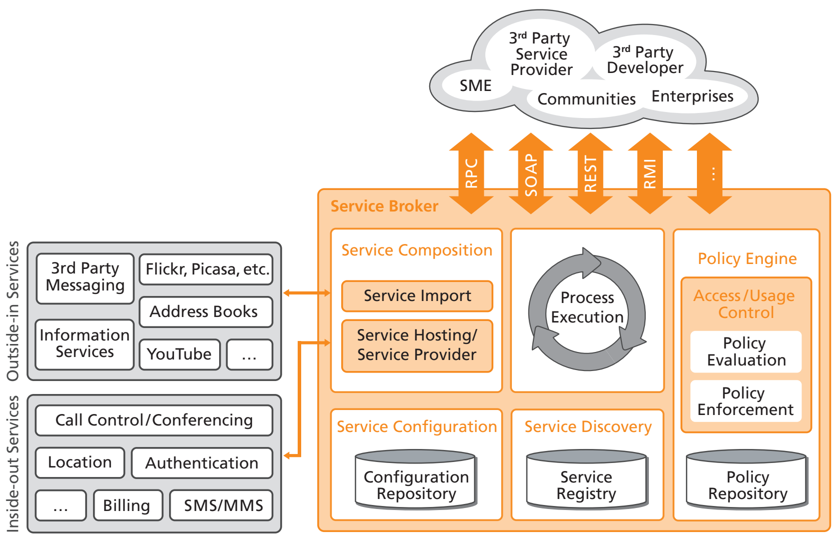


Figure 6.1: Service Broker for beta developer platform

6.2.2 Cross-network, Cross-domain Information Sharing

Integrating synchronous and asynchronous communication paradigms into a holistic service and combining services and end devices from multiple domains requires cooperative session awareness as an underlying principle. When combining different clients, connected to multiple networks, each with its own specific signalling and transport protocol, one cannot assume that all clients may become part of the same session. It would be a complicated task to handle multiple different clients including the combinations of different communication methods with their specific manners, networks, and huge variety of devices. The goal of this project is to overcome technological and administrative boundaries for information sharing by applying an information push towards clients. This push service is capable of delivering information from various sources to different devices depending on user, operator, and content provider preferences. The service can be considered as an enhancement or extension of a simple information service such as IP-based Television (IPTV), Video-on-Demand (VoD), e-Health record, P.O. box, network address book/scheduler, and file/video sharing.

The main target is to enable services to adapt to evolving service environments in three aspects:

1. multi-network,
2. multi-device, and
3. multi-service.

In a multi-network environment, users may receive services via appropriate networks and connected devices and users are able to use multiple devices without any special efforts. The term “multi-service” refers to the situation that most service providers have popular content and application services outside of a carrier network and deliver their services in an OTT manner. This spotlights the significance of inducing the content and application services to carrier networks by allowing carrier network services to mash-up outside services. Furthermore, OTT services may increase their mash-up services by utilizing additional data available from networks such as the device location, easier connection to the other users/devices on the network, etc.

6.2.3 Cross-layer Composition

The German BMBF G-Lab DEEP project focuses on research of innovative composition approaches for the cooperation of network and service layer with a special focus on security for the Future Internet. The project is scheduled as a 3.5 year project, started in September 2009 until March 2012.

The main focal point lies on the dynamic, controllable communication of requirements resulting from a concrete service request originated in the application layer towards the network layer. As a result, network and application layer dynamically provide corresponding functional building blocks and required composed services supported by the target architecture and associated tools. Central research questions and problems as description, administration, discovery, and functional combination of services to more complex (workflow-based) services are investigated on multiple layers of the network architecture. The goal is to apply concepts of SOA to all network layers and replace finally the layer-oriented architecture of the Internet through a global service-oriented (network) architecture.

In G-Lab DEEP, applicable tools and frameworks are investigated providing the flexible creation of communication workflows for secure voice and multimedia communication in the Future Internet allowing the cooperation of the network and service layer based on SOA principles. Therefore, events from multiple layers (cross-layer), different technologies (cross-technology) beyond the borders of administrative domains (cross-domain) are targeted and will be provisioned. It presents a Future Internet architecture based on the cross-layer composition concept in which the network is composed according to application specific requirements. Functional composition is used to select and compose functionalities on the network data path, suitable for the demanded application level requirements.

The project makes use of the designed policy engine to apply constraints during the functional composition process. The following figure 6.2 illustrates the application of our Service Broker for such a cross-layer service composition mechansim:

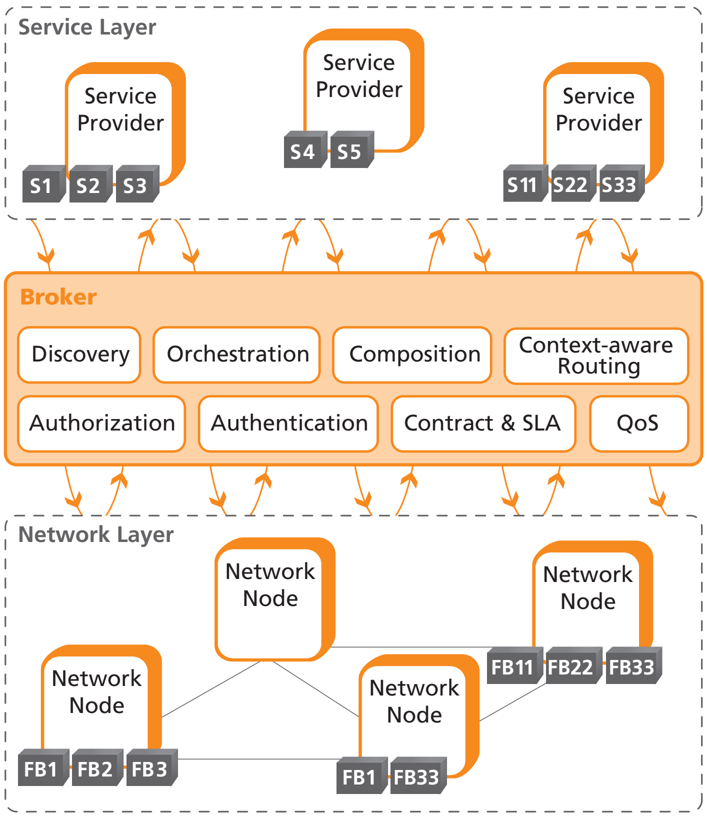


Figure 6.2: Service Broker for cross-layer service composition

The Service Broker identifies the required services, which are necessary to satisfy user requests. Such a service may consist of a single service or several services, which might be combined. Functional building blocks (FBs) of network nodes will be composed according to requirements of the service layer, where services are composed of service features from multiple providers and domains. Service layer compositions are grounded to SCXML for execution.

Acronyms

3GPP	3^{rd} Generation Partnership Project
AAA	Authorization, Authentication and Accounting
API	Application Programming Interface
AR	Application Router
AS	Application Server
BPEL	Business Process Execution Language
CAMEL	Customized Applications for Mobile Enhanced Logic
CCXML	Call Control eXtensible Markup Language
CIM	Computation Independent Model
CLASS	Customer Access Line Signalling Services
CORBA	Common Object Request Broker Architecture
CS	Capability Set
CSCF	Call Session Control Function
CTI	Computer Telephony Integration
CTL	Computation Tree Logic
CWM	Common Warehouse Metamodel
DSL	Domain-Specific Languages
EAI	Enterprise Architecture Integration
ECA	Event Condition Action
EJB	Enterprise Java Beans
ESB	Enterprise Service Bus
eTOM	enhanced Telecom Operations Map
ETSI	European Telecommunications Standards Institute
FB	Functional Building Block
FI	Future Internet
FIRE	Future Internet Research and Experimentation
FMC	Fixed Mobile Convergence
FTTH	Fiber-To-The-Home
GEOPRIV	GEOlocation PRIVacy
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GSMA	Groupe Speciale Mobile Association
GUI	Graphical User Interface

HSS	Home Subscriber System
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
iFC	initial Filter Criteria
IMS	IP Multimedia Subsystem
IN	Intelligent Network
INAP	Intelligent Network Application Part
IoS	Internet of Services
IP	Internet Protocol
IPTV	IP-based Television
ISC	IMS Service Control
ISDN	Integrated Services Digital Network
IT	Information Technologies
ITU-T	International Telecommunication Union -Telecommunication
IVR	Interactive Voice Response
JAIN	Java APIs for integrated Networks
JCP	Java Community Process
JSON	JavaScript Object Notation
JTAPI	Java Telephony Application Programming Interface
KISS	Keep It Simple, Stupid
KTS	Kripke Transitions System
LTA	Long-term Architecture
LTE	Long Term Evolution
MDA	Model Driven Architecture
MDE	Model-Driven Engineering
MDSD	Model-Driven Software Development
MOF	Meta-Object Format
MVNO	Mobile Virtual Network Operator
NGOSS	New Generation Operations Systems and Software
NGSI	Next Generation Service Interfaces
OCL	Object Constraint Language
OMA	Open Mobile Alliance
OMG	Object Management Group
OS	Operating System
OSE	OMA Service Environment
OSOA	Open Service Oriented Architecture
OSPE	OMA Service Provider Environment
OTT	Over The Top
PBX	Private Branch Exchange
PDM	Platform Definition Model
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PII	Personally Identifiable Information

PIM	Platform-independent Model
POTS	Plain old telephone service
PSM	Platform-specific Model
PSTN	Public Switched Telephone Network
QoE	Quality of Experience
QoS	Quality of Service
RBAC	Role-based Access Control
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTSP	Real Time Streaming Protocol
S-CSCF	Serving Call Session Control Function
SAML	Security Assertion Markup Language
SBC	Session Border Controller
SCA	Service Component Architecture
SCE	Service Creation Environment
SCIM	Service Capability Interaction Manager
SCM	Service Capability Manager
SCP	Service Control Points
SCXML	State Chart XML
SDF	Service Delivery Framework
SDK	Software Development Kit
SDP	Service Delivery Platform
SIB	Service Independent Building Block
SID	Shared Information/Data Model
SIP	Session Initiation Protocol
SLEE	Service Logic Execution Environment
SLG	Service Logic Graph
SME	Small and Medium Enterprise
SMI	Service Management Interface
SMS	Short Messaging Service
SOA	Service Oriented Architecture
SOA-RM	Service Oriented Architecture Reference Model
SPATEL	SPICE Advanced language for Telecommunication services
SPDE	Service Provider Deliver Environment
SPL	Session Procession Language
SuT	System under Test
TAM	Telecom Application Map
TelcoML	Telecommunication SOA Modeling Language
TINA-C	Telecommunication Information Networking Architecture Consortium
TISPAN	TIPHON (Telecommunications and Internet Protocol Harmonization over Networks) and SPAN (Services and Protocols for Advanced Networks)

TMF	TeleManagement Forum
TS-DSL	Telecom Service Domain Specific Language
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
UPSF	User Profile Server Function
URI	Uniform Resource Identifier
VAS	Value-added Service
VoD	Video-on-Demand
VoiceXML	Voice eXtensible Markup Language
VoIP	Voice over IP
VPN	Virtual Private Networks
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WS-BPEL	Web Services Business Process Execution Language
WS-Policy	Web Services Policy
WSFL	Web Services Flow Language
WWW	World Wide Web
XACML	eXtensible Access Control Markup Language
XCAP	XML Configuration Access Protocol
XML	eXtensible Markup Language
XSLT	XML Stylesheet Language for Transformations

Bibliography

[3G 99] 3GPP. 3GPP TS 23.078. Customized Applications for Mobile network Enhanced Logic (CAMEL) Phase X; Stage 2. 1999.

[3G 03] 3GPP. 3GPP TS 23.002 V5.12.0, 3rd Generation Partnership Project; Technical Specification Group Services and Systems Aspects; Network architecture (Release 5). Sep. 2003.

[3G 06] 3GPP. 3GPP TS 23.228 V7.6.0. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Stage 2 (Release 7). Dec. 2006.

[3G 08] 3GPP. 3GPP TR 23.810 V8.0.0. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Study on Architecture Impacts of Service Brokering (Release 8). Sep. 2008.

[3G 09] 3GPP. 3GPP TS 29.199-XX V8.0.0, Open Service Access (OSA); Parlay X Web Services. Sept. 2009.

[AKG 05] J. de Albuquerque, H. Krumm, P. de Geus. Policy Modeling and Refinement for Network Security Systems. Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05), pp. 24-33, 2005.

[ABM 07] A. Al-Hezmi, N. Blum, T. Magedanz. IMS basiertes Triple Pay Toolkit - Der FOKUS Open IMS Playground. in Praxis Profiline - Triple Play. pp.18-21. Vogel Industrie Medien und Triple Play Alliance. Juli 2007. ISBN: 3-8259-1948.

[AS 07] D. Amyot, R. Simoes. Combining VoiceXML with CCXML: A Comparative Study. 4th IEEE Consumer Communications and Networking Conference CCNC 2007, pp.342-346, Jan. 2007.

[BLW 05] P. Baker. P. Loh, F. Weil. Model-Driven Engineering in a Large Industrial Context - Motorola Case Study. ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2005). LNCS, vol. 3713, pp. 476-491. Springer, Heidelberg. 2005.

[BL 96] C. Baral, J. Lobo. Formal characterization of active databases. in LID '96: Proceedings of the International Workshop on Logic in Databases. London, UK. Springer-Verlag, pp.175-195. 1996.

[BLT 97] C. Baral, J. Lobo, G. Trajcevski. Formal characterizations of active databases: Part ii. in Deductive and Object-Oriented Databases, S. B. Heidelberg, Ed. 1997.

[BF 08] M. Belaunde, P. Falcarin. Realizing an MDA and SOA Marriage for the Development of Mobile Services. In. I. Schieferdecker and A. Hartman (Eds.). ECMDA-FA 2008, LNCS 5095, pp. 393-405. Springer, Heidelberg. 2008.

[BC 08] E. Bertin, N. Crespi. Describing Next Generation Communication Services: A Usage Perspective. 1st European Conference on Towards A Service-Based internet. Madrid, Spain, December 10 - 13, 2008. P. Mähönen, K. Pohl, and T. Priol, Eds. Lecture Notes In Computer Science, vol. 5377. Springer-Verlag, Berlin, Heidelberg, 86-97. 2008. ISBN 978-3-540-89896-2.

[B 04] J. Bettin. Model-Driven Software Development Activities, The Process View of an MDSD Project. 2004, http://www.softmetaware.com/mdsd-process.pdf

[B 05] T. Bloomfield. MDA, Meta-Modeling and Model Transformation: Introducing New Technology into the Defense Industry. In: A. Hartman, D. Kreische (Eds.) ECMDA-FA 2005. LNCS 3748, pp. 9-18. Springer, Heidelberg. 2005.

[BM 05] N. Blum, T. Magedanz. Push-To-Multimedia as a Platform Enabler for NGN Services. 11th European Wireless Conference 2005. Nicosia, Cyprus. April, 10-13. VDE Verlag GmbH, pp. 315-321. 2005. ISBN 3-8007-2886-9.

[BM 05a] N. Blum, T. Magedanz. PTT+IMS=PTM - Towards Community/Presence-based IMS Multimedia Services. 7th IEEE International Symposium on Multimedia, Irvine, Ca (U.S.A.). 12.-14. December 2005. ISBN 0-7695-2489-3.

[BGM 07] N. Blum, F. Carvalho de Gouveia, T. Magedanz. An Open IMS Testbed for exploring Wireless Service Evolution and Network Architecture Evolution towards SAE and LTE. 2nd IEEE Australian Conference on Wireless Broadband and Ultra Wideband Communications. Sydney, Australia. 27 - 30 August 2007. ISBN 0-7695-2842-2.

[BMS 07] N. Blum, T. Magedanz, H. Stein. Service Creation & Delivery for SME based on SOA / IMS. MNCNA ’07, Nov. 26, 2007. Port Beach - CA. 2007 ISBN 978-1-59593-932-6

[BDM 07] N. Blum, S. Dutkowski, T. Magedanz. Combining enhanced mobile IMS Presence with Web-based Location Services. 4. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste. Sept. 13./14. 2007, Hrsg. Roth, Küpper und Linnhoff-Popien. Munich, 2007. ISBN 978-3-89963-591-1.

[BLM 08] N. Blum, L. Lange, T. Magedanz. Combining Web 2.0 and NGN: Mobile geo-blogging as Service Enabler for Next Generation Networks. 5. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste. 4.-5. September 2008, Nürnberg. Jörg Roth (Ed.). Sonderdruck Schriftenreihe der Georg-Simon-Ohm-Hochschule Nürnberg Nr. 42. pp.53-58. 2008 ISSN 1867-5433.

[BLK+ 08] N. Blum, D. Linner, S. Krüssel, T. Magedanz, S. Steglich. Definition of a Web 2.0 Gateway for 3rd Party Service Access to Next Generation Networks. IFIP International Federation for Information Processing, Volume

284. Wireless and Mobile Networking. Zoubir Mammeri. (Boston: Springer). pp. 247-258. 2008 ISBN: 978-0-387-84838-9.

[BMS 08] N. Blum, T. Magedanz, F. Schreiner. The Role of Service Brokers for Composed Services in an Open Service Environment. TELEKOMMUNIKATION AKTUELL 1-2/2008. Verlag für Wissenschaft und Leben Georg Heidecker GmbH, Erlangen. 2008. ISSN 1619-2036

[BT 08] N. Blum, T. Magedanz. Requirements and Components of a SOA-based NGN Reference Architecture. e&i - elektrotechnik und informationstechnik. Österreichischer Verband für Elektrotechnik. pp.263-267. Springer-Verlag 2008. Juli/August 2008. ISSN 0932-383X.

[BMS+ 09] N. Blum, T. Magedanz, F. Schreiner, S. Wahle. From IMS Management to SOA based NGN Management. Journal of Network and Systems Management, Springer New York. Feb. 2009. ISSN 1064-7570.

[BMS+ 09a] N. Blum, T. Magedanz, F. Schreiner, S. Wahle. A Research Infrastructure for SOA-based Service Delivery Frameworks - The Open SOA Telco Playground at Fraunhofer FOKUS. 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2009), Washington DC, USA. April 2009. ISBN 978-1-4244-2847-2.

[BMK+ 09] N. Blum, T. Magedanz, J. Kleeßen, T. Margaria. Enabling eXtreme Model Driven Design of Parlay X-based Communications Services for End-to-End Multiplatform Service Orchestrations. 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS). pp.240-247. 2009. ISBN 978-0-7695-3702-3

[BLM+ 09] N. Blum, L. Lange, T. Magedanz, J. Simoes. Mediacast for Mobile Communities: When the Web and Telecommunications converge. 10th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks. Greece, June 15-19. 2009. ISBN 978-1-4244-4439-7.

[BLM 09] N. Blum, S. Lampe, T. Magedanz, "Design of a Message Interworking Function for Converged IP Messaging in Next Generation Networks", IEEE Symposium on Computers and Communications (ISCC'09), July 5 - 8, 2009, Sousse, Tunisia, www.comsoc.org/iscc/2009

[BMM 09] N. Blum, T. Magedanz, T. Margaria. Rapid Service Creation using eXtreme Model Driven Design for real-time Communications Services on top of Next Generation Networks. 13th International Conference on Intelligence in Next Generation Networks, 2009. ICIN 2009. pp.1-6. 26-29 Oct. 2009. doi: 10.1109/ICIN.2009.5357109.

[BM 10] N. Blum, T. Margaria. An Open Service Environment for Service Exposure and Orchestration of Heterogeneous NGN Services. Praxis in der Informationsverarbeitung und Kommunikation (PIK), Fachzeitschrift für den Einsatz von Informationssystemen. 33. Jahrgang, Heft 1, 2010. De Gruyter SAUR Verlag, Germany. www.degruyter.com/pik. ISSN: 0930-5157. 2010

[BMS+ 10] N. Blum, T. Magedanz, H. Stein, I. Wolf. A Platform For User Generated Multimedia Communication Services. Journal of Mobile Multimedia, Vol.6 No.3. pp.185-206. 2010. ISSN 1550-4646

[BHL+ 99] M. Born, A. Hoffmann, M. Li, I. Schieferdecker. Using Formal Methods for the Design of Telecommunication Services. Proceedings of the Conference on Formal Methods for Object Oriented Distributed Systems (FMOODS) 1999, Florence, Italy 1999. Boston: Kluwer, 1999. ISBN: 0-7923-8429-6.

[BCF+ 05] M. Brambilla, S. Ceri, P. Fraternali, R. Acerbis, A. Bongio. Model-Driven Design of Service-Enabled Web Applications. ACM SIGMOD International Conference on Management of Data, pp. 851-856. 2005.

[BMS+ 97] V. Braun, T. Margaria, B. Steffen, F.-K. Bruhns. Service Definition for Intelligent Networks: Experience in a Leading-edge Technological Project Based on Constraint Techniqueservice Definition for Intelligent Networks: Experience in a Leading-edge Technological Project Based on Constraint Techniques. Proc. PACT'97. 3rd Int. Conf. on Practical Application of Constraint Technology - April 1997. London, UK. Ed. by The Practical Application Company.

[B 07] C. Bussler. The Fractal Nature of Web Services. IEEE Comp, vol. 40, no. 3, pp. 93-95. 2007. doi:10.1109/MC.2007.106

[BCL+ 06] L. Burgy, C. Consel, F. Latry, J. Lawall, L. Réveillère, and N. Palix. Language Technology for Internet-Telephony Service Creation. IEEE International Conference on Communications (ICC'06), Istanbul, Turkey. 2006.

[BC 08] G. Bond, E. Cheung. A framework for converged telecom services and mashups. Technical Report TD-7DKHSW v2, AT&T, 2008. available at http://echarts.org

[BHK 09] J. Bauknecht, J. Haussler, D. Kraft, M. Kuhnen M. Lischka, A. Schulke A. Policy-Based Real-Time Decision-Making for Personalized Service Delivery. IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY '09). IEEE Computer Society, Washington, DC, USA, 53-59. 2009. DOI=10.1109/POLICY.2009.13.

[BAC+ 09] C. Baladron, J. Aquiar, B. Carro, L. Goix, A. Leon Martin, P. Falcarin, J. Sienel. User-Centric Future Internet and Telecommunication Services. Future of the Internet Conference, Prague (Czech Rep.) May 2009.

[CLG+ 03] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko. Diameter Base Protocol. Request for Comments: 3588, Network Working Group, <http://www.ietf.org/rfc/rfc3588.txt?number=3588>. Sep. 2003.

[CIJ+ 00] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, M. Shan. eFlow: A Platform for Developing and Managing Composite e-Services. HP Labs Technical Report, HPL-2000-36, HP Software Technology Laboratory, Palo Alto, Ca. USA. March 2000.

[CJF+ 05] D. Chakraborty, A. Joshi, T. Finin, Y. Yesha. Service Composition for Mobile Environments. Journal on Mobile Networking and Applications, Special Issue on Mobile Services, Vol. 10, No. 4, pp. 435-451. 2005.

[CGP 00] E. Clarke, O. Grumberg, D. Peled. Model Checking. MIT Press. 1999. ISBN 0-262-03270-8.

[CFB 04] I. Constantinescu, B. Faltings, W. Binder. Large Scale, Type-Compatible Service Composition. ICWS '04: Proceedings of the IEEE International Conference on Web Services, Washington, DC, USA. 2004.

[C 06] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). Request for Comments: 4627, Network Working Group, <http://www.ietf.org/rfc/rfc4627.txt?number=4627>. July 2006.

[DLT+ 03] G. Deng, T. Lu, E. Turkay, A. Gokhale, D. Schmidt, A. Nechypurenko. Model Driven Development of Inventory Tracking System. 3rd OOPSLA Workshop on Domain Specific Modeling (DSM 2003), p. 6. 2003

[DS 05] S. Dustdar, W. Schreiner. A survey on web services composition. Int. J. Web Grid Services, pp. 1-30, 1, 1 (Aug. 2005).

[ETSI 09] ETSI. ETSI ES 282 002: Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); PSTN/ISDN Emulation Sub-system (PES); Functional architecture. 2009.

[ETSI 09a] ETSI. ETSI TS 182 028: Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); IPTV Architecture; Dedicated subsystem for IPTV functions. 2009

[F 00] R.T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[FG 05] K. Fisher, R. Gruber. PADS: A Domain-Specific Language for Processing Ad-Hoc Data. ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI'05), pages 295-304, Chicago, IL, USA. 2005.

[F 82] C. Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence, 19, pp. 17-37, 1982.

[FS 05] K. Fujii, T. Suda. Semantics-based dynamic service composition. IEEE Journal on Selected Areas in Communications, Vol. 23, No. 12. 2005.

[GKM 03] R.H. Glitho, F. Khendek, A. De Marco. Creating value added services in Internet telephony: an overview and a case study on a high-level service creation environment. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Review vol.33 no.4. pp.446-457. Nov 2003. doi: 10.1109/TSMCC.2003.818499.

[GC 07] A. Gouya, N. Crespi. Service Broker for Managing Feature Interactions in IP Multimedia Subsystem. Sixth International Conference on Networking (ICN '07), pp.54-54, 22-28 April 2007

[GPZ 04] T. Gu, H. Pung, D.Zhang. A Middleware for Building Context-Aware Mobile Services. Proceedings of IEEE Vehicular Technology Conference, Los Angeles, USA. 2004.

[H 87] D. Harel. Statecharts: A visual formalism for complex systems. Sci. Comput. Program. 8, 3 , pp. 231-274. Jun. 1987. doi= http://dx.doi.org/10.1016/0167-6423(87)90035-9

[HKK+ 08] A. Hartman, M. Keren, S. Kremer-Davidson, D. Pikus. Model-based Design and Generation of Telecom Services. available at https://www.research.ibm.com/haifa/projects/services/sce/papers/Automated GenerationOfTelecomServicesFinal.pdf

[HMS+ 06] R. Högg, M. Meckel, K. Stanoevska-Slabeva, R. Martignoni. Overview of Business Models for Web 2.0 Communities. Proc. Gemeinschaften in Neuen Medien (GeNeMe 2006), Technische Universität Dresden, pp. 23-37. 2006.

[ILeM 09] N. Ibrahim, F. Le Mouel. A Survey on Service Composition Middleware in Pervasive Environments. International Journal of Computer Science Issues, IJCSI, Volume 1, pp1-12. August 2009. ISSN (Online): 1694-0784.

[ITU1200] ITU-T. Recommendation Q.1200, Q-Series Intelligent Network Recommendation Structure. Technical report, International Telecommunication Union, October 1992.

[JB 96] S. Jablonski, C. Bussler. Workflow Management - Modeling Concepts, Architecture and Implementation. International Thomson Computer Press. 1996.

[JSS 97] S. Jajodia, P. Samarati, V. Subrahmanian. A logical language for expressing authorizations. In SympSecPr, Research in Security and Privacy, Oakland, CA. May 1997.

[JCP 02] Java Community Process. JSR43 JTAPI 1.4 Specification. 2002.

[JCP 02a] Java Community Process. JSR21 JAIN JCC Specification. 2002.

[JCP 04] Java Community Process. JSR 22: JAIN SLEE API Specification. 2004.

[JCP 08] Java Community Prociess. JSR289: SIP Servlet v1.1. 2008.

[JPY+ 09] L. Jin, P. Pan, C. Ying, J. Liu, Q. Tian. Rapid service creation environment for service delivery platform based on service templates. 11th IFIP/IEEE international Conference on Symposium on integrated Network Management (New York, NY, USA, June 01 - 05, 2009). IEEE Press, Piscataway, NJ, 117-120. 2009.

[JKN+ 06] S. Jörges, C. Kubczak, R. Nagel, T. Margaria, B. Steffen. Model-driven development with the jABC. In HVC 2006 - IBM Haifa Verification Conference, Haifa, Israel. October 23-26 2006. LNCS 4383. IBM, Springer Verlag. 2007. ISBN 978-3-540-70888-9.

[JN 04] E. Jouenne, V. Normand. Tailoring IEEE 1471 for MDE Support. In: Jardim Nunes, N., Selic, B., Rodrigues da Silva, A., Toval Alvarez, A. (eds.) UML Satellite Activities 2004. LNCS, vol. 3297, pp. 163-174. Springer, Heidelberg. 2005.

[JMN+ 08] G. Jung, T. Margaria, R. Nagel, W. Schubert, B. Steffen, H. Voigt, H. SCA and jABC: Bringing a service-oriented paradigm to Web-service construction. Proc. 3rd Int. Symp. on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA08). Chalkidiki (GR), Oct. 2008. CCIS N. 017, Springer Verlag. 2009. ISBN 978-3-540-88478-1

[KKS 07] S. Kalasapur, M. Kumar, B. Shirazi. Dynamic Service Composition in Pervasive Computing. IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 7, pp. 907-918. 2007.

[KMW 04] O. Kath, T. Magedanz, R. Wechselberger. MDA-based Service Creation for OSA/Parlay within 3Gbeyond Environments. Modatel Workshop, March 17-18, 2004. University of Twente, Enschede, The Netherlands, pp. 33-42, Publisher Eurescom GmbH. 2004. ISSN 1381-3625.

[KGL 07] S. Khan, R. Gaglianello, M. Luna. Experiences with blending HTTP, RTSP, and IMS. IEEE Commun. Mag. vol. 45. Mar. 2007. pp. 122-128.

[K 82] D. Kozen. Results on the Propositional Mu-Calculus. Ninth International Colloquium on Automata, Languages, and Programming, pp. 348-359. 1982.

[LR 95] D. Ladd, J. Ramming. Programming the Web: An application-oriented language for hypermedia service programming. 4th International World Wide Web Conference, Boston, Massachusetts. Dec. 1995.

[LJD 01] S. St.Laurent, J. Johnston, E. Dumbill. Programming web services with XML-RPC. O'Reilly. 2001. ISBN 0596001193.

[LMC 07] F. Latry, J. Mercadal, C. Consel. Staging telephony service creation: a language approach. 1st international Conference on Principles, Systems and Applications of IP Telecommunications, New York City, New York, July 19 - 20, 2007. IPTComm '07, pp. 99-110. 2007.

[LMO+ 09] S. Loreto, T. Mecklin, M. Opsenica, H. Rissanen. Service broker architecture: location business case and mashups. IEEE Comm. Mag. 47, 4, pp.97-103. 2009.

[LSP+ 06] R. Lu, S. Sadiq, V. Padmanabhan, G. Governatori. Using a temporal constraint network for business process execution. 17th Australasian Database Conference - Volume 49 (Hobart, Australia, January 16 - 19, 2006). G. Dobbie and J. Bailey, Eds. ACM International Conference Proceeding Series, vol. 170. Australian Computer Society, Darlinghurst, Australia, 157-166. 2006.

[M 93] T. Magedanz. IN and TMN providing the basis for future information networking architectures. in Computer and Communications, pp. 267-276, Butterworth-Heineman, Vol.16, No. 5, May 1993.

[M 96] T. Magedanz, R. Popescu-Zeletin. Intelligent Networks: Basic Technology, Standards and Evolution. Thompson Computer Press, 1996. ISBN 1-85032-293-7.

[MBD 08] T. Magedanz, N. Blum, S. Dutkowski. Evolution of SOA Concepts in Telecommunications - A Déjà vu?. IEEE Computer Special Issue on Service Oriented Architectures. Nov. 2007. ISSN 0018-9162

[M 03] K. Mantell. From UML to BPEL. IBM developerWorks vol. 2004. 2003.

[MS 09] T. Margaria, B. Steffen. Business Process Modelling in the jABC: The One-Thing-Approach. Handbook of Research on Business Process Modeling, J. Cardoso and W. van der Aalst (eds.), IGI Global, 2009.

[MHT 09] N. Matsumoto, M. Hayashi, H. Tanaka, H. Network middleware design for bridging legacy infrastructures and NGN. 5th Euro-NGI Conference on Next Generation internet Networks (Aveiro, Portugal, July 01 - 03, 2009). IEEE Press, Piscataway, NJ, 175-183. 2009.

[M 05] A. McAfee. Will Web Services Really Transform Collaboration. MIT Sloan Management Review, vol. 46, no. 2, pp. 78-84. 2005.

[MBE 03] B. Medjahed, A. Bouguettaya, A. K. Elmagarmid. Composing Web services on the Semantic Web. The VLDB Journal, Vol. 12, No. 4, pp. 333-351. 2003.

[M 03] Middleware Company. Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach. Productivity Analysis. Report by the Middleware Company on behalf of Compuware (2003), http://www.omg.org/mda/mda_files/MDA_Comparison-TMC_final.pdf

[MV 08] P. Mohagheghi, V. Dehlen. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. Model Driven Architecture - Foundations and Applications, pp. 432 - 443, Springer Berlin / Heidelberg, 2008. ISSN 0302-9743.

[M 07] S. Ben Mokhtar. Semantic Middleware for Service-Oriented Pervasive Computing. Ph.D. thesis, University of Paris 6, Paris, France. 2007.

[MTS 03] R. Montanari, G. Tonti, C. Stefanelli. Policy-based separation of concerns for dynamic code mobility management. 27th Annual International Computer Software and Applications Conference, 2003. COMPSAC 2003. pp. 82-90. 3-6 Nov. 2003.

[MBO+ 09] S. Moro, S. Bouat, M. Odini, J. O'Connell. Service Composition with Real Time Services. 13th International Conference on Intelligence in Next Generation Networks (ICIN). Bordeaux, 26 - 29 October, 2009. ISBN 978-1-4244-4694-0.

[OASIS 05] OASIS. XACML 2.0 Core: eXtensible Access Control Markup Language (XACML). Version 2.0. Feb. 2005.

[OASIS 06] OASIS. Reference Model for Service Oriented Architecture 1.0. OASIS Standard. 12 October 2006.

[OASIS 07] OASIS. Web Services Business Process Execution Language. Version 2.0. April 2007.

[OMA 05] Open Mobile Alliance (OMA). OMA Service Provider Environment Requirements. Candidate Version 1.0 - 14. June 2005.

[OMA 07] Open Mobile Alliance (OMA). Architecture Document OMA Service Environment. Approved Version 1.0.4 - 01 Feb 2007. 2007.

[OMA 08a] Open Mobile Alliance (OMA). Policy Evaluation, Enforcement and Management Callable Interface (PEM1). 2008.

[OMA 08b] Open Mobile Alliance (OMA). Policy Evaluation, Enforcement and Management - Management Interface (PEM2). 2008.

[OMA 09] Open Mobile Alliance (OMA). Enabler Release Definition for Next Generation Service Interfaces. Candidate Version 1.0 - 18 Nov 2009.

[OMG 97] Object Management Group. UML Specification version 1.1 (OMG document ad/97-08-11). 1997. http://www.omg.org/cgi-bin/doc?ad/97-08-11

[OMG 08] Object Management Group. OMG MetaObject Facility. 2008. http://www.omg.org/mof/

[OSC 03] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. White paper. 2003.

[OSOA 09] Open Service Oriented Architecture collaboration. Service Component Architecture Specifications. http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications. April 2009.

[PacketCable 09] PacketCable 2.0. http://www.packetcable.com/specifications/specifications20.html. 2009

[PC 07] N. Parlavantzas, G. Coulson. Designing and constructing modifiable middleware using component frameworks. Software, IET , vol.1, no.4. pp.113-126. Aug. 2007.

[PF 02] S. Ponnekanti, A. Fox. SWORD: A developer toolkit for web service composition. 11th World Wide Web Conference, Honolulu, USA. 2002.

[PB 05] M. Presso, M. Belaunde. Applying MDA to Voice Applications: an Experience in Building an MDA Tool Chain. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 1-8. Springer, Heidelberg. 2005.

[QLL 06] X. Qiao, X. Li, Y. Li. MDA-Based 3G Service Creation Approach and Telecom Service Domain Meta-Model. The Journal of China Universities of Posts and Telecommunications, Volume 13, Issue 2, pp.54-60. June 2006. ISSN 1005-8885, DOI: 10.1016/S1005-8885(07)60104-6.

[RCC 09] Y. Rakaiby, F. Cuppens, N. Cuppens-Boulahia. Formalization and Management of Group Obligations. IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 158-165, 2009 IEEE International Symposium on Policies for Distributed Systems and Networks. 2009.

[RSC+ 02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. SIP: Session Initiation Protocol. Request for Comments: 3261. Network Working Group, <http://www.ietf.org/rfc/rfc3261.txt?number=3261>. June 2002.

[R 07] J. Rosenberg. Presence Authorization Rules. draft-ietf-simple-presence-rules-10, work in progress. July 2007.

[R 07a] J. Rosenberg. The Extensible Markup Language (XML) Configuration Access Protocol (XCAP). Request for Comments: 4825, Network Working Group, <http://tools.ietf.org/html/rfc4825>. May 2007.

[SCF+ 97] R. Sandhu, E. Coyne, H. Feinstein, C. Youman. Role-based access control models. IEEE Computer, 29(2):38-47. 1996.

[SJ 07] C. Schroth, T. Janner. Web 2.0 and soa: Converging concepts enabling the internet of services. IT Professional, vol. 9, no. 3, pp. 36-41. May 2007. ISSN:1520-9202

[STM+ 07] H. Schulzrinne, H. Tschofenig, J. Morris, J. Cuellar, J. Polk, J. Rosenberg. Common Policy: A Document Format for Expressing Privacy Preferences. Request for Comments: 4745. Network Working Group <http://www.ietf.org/rfc/rfc4745.txt?number=4745>. February 2007.

[STM+ 09] H. Schulzrinne, H. Tschofenig, J. Morris, J. Cuellar, J. Polk. Geolocation Policy: A Document Format for Expressing Privacy Preferences for Location Information. draft-ietf-geopriv-policy-21, work in progress. July 2009.

[RA 04] R. Shah, N. Apte. Web Services: A Business Module Packaging Strategy. Prentice Hall PTR online article. Apr. 2004.

[SKS 07] D. Shirtz, M. Kazakov, Y. Shaham-Gafni. Adopting Model Driven Development in a Large Financial Organization. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) ECMDAFA 2007. LNCS, vol. 4530, pp. 172-183. Springer, Heidelberg. 2007.

[SHP 03] E. Sirin, J. Hendler, B. Parsia. Semi-automatic composition of Web services using semantic descriptions. Proc. of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS 2003, April 2003.

[SL 02] M. Sloman, E. Lupu. Security and management policy specification. IEEE Network, Special Issue on Policy-Based Networking, 16(2):10-19. March/April 2002.

[S 06] M. Stal. Using Architectural Patterns and Blueprints for Service-Oriented Architecture. IEEE Software, vol. 23, no. 2. Mar./Apr. 2006.

[SFH+ 04] J. Strassner, J. Fleck, J. Huang, C. Faurer, T. Richardson. TMF White Paper on NGOSS and MDA. April 2004.

[TINA 97] TINA-C Deliverable. Service Architecture Version 5.0. http://www.tinac.com/specifications/documents/sa50-main.pdf. 1997

[TMF 08] TeleManagement Forum. TR139 Service Delivery Framework Overview Release 2.0. September 2008.

[UP 07] A. Ulrich, A. Petrenko. Reverse Engineering Models from Traces to Validate Distributed Systems - an Industrial Case study. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) ECMDA-FA 2007. LNCS, vol. 4530, pp. 185-193. Springer, Heidelberg. 2007.

[U 00] UPnP Forum. Understanding UPnP: A White Paper. Technical Report. 2000.

[VZM 00] I. Venieris, F. Zizza, T. Magedanz (Ed.). Object Oriented Software Technologies in Telecommunications - From Theory to Practice. Wiley Publishers UK, April 2000. ISBN: 0471-6233792.

[W3C 07] W3C. SOAP Version 1.2. 2007. http://www.w3.org/TR/soap/

[W3C 07a] W3C. Voice Extensible Markup Language (VoiceXML) 2.1. 2007. http://www.w3.org/TR/voicexml21/

[W3C 07b] W3C. Voice Browser Call Control: CCXML Version 1.0. 2007. http://www.w3.org/TR/ccxml/

[W3C 07c] W3C. Web Services Policy 1.5 - Framework. 2007. http://www.w3.org/TR/ws-policy/

[W3C 09] W3C. State Chart XML (SCXML): State Machine Notation for Control Abstraction. 2009. http://www.w3.org/TR/scxml/

[W3C 09a] W3C. Web Application Description Language. 2009. http://www.w3.org/Submission/wadl/

[WBG 08] K. Walzer, T. Breddin, M. Groch. Relative temporal constraints in the Rete algorithm for complex event detection. Second international Conference on Distributed Event-Based Systems (Rome, Italy, July 01 - 04, 2008). DEBS '08, vol. 332. ACM, New York, NY, pp. 147-155, 2008.

[WR 04] P. Weill, J.W. Ross. IT Governance: How Top Performers Manage IT Decision Rights for Superior Results. Harvard Bus. School Press, 2004. ISBN 1591392535.

[WW 06] T. Weigert, F. Weil. Practical Experiences in Using Model-Driven Engineering to Develop Trustworthy Computing Systems. In: IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC 2006), pp. 208-217. 2006.

[XTT+ 08] X. Shao, T. Chai, T. Lee, L. Ngoh, L. Zhou, M. Kirchberg. An Integrated Telecom and IT Service Delivery Platform. 2008 IEEE Asia-Pacific Services Computing Conference. APSCC. pp.391-396. 2008.

[YPG 00] R. Yavatkar, D. Pendarakis, R. Guerin. A Framework for Policy-based Admission Control. Request for Comments: 2753, Network Working Group <http://www.ietf.org/rfc/rfc2753.txt>. Jan. 2000.

Appendix A

Policy Rule Model

This section describes in more detail the proposed rule model within policies. The description is based on the Common Policy Format specifications which define a highly flexible and extensible policy model.

For a more efficient rule processing and management, rules define only what it is "permited". If several rules apply, the overall evaluation result represents the union of each rule evaluation result. By default the evaluation decision equals "deny". Also in case the constraints of one of the matching rules are not respected the request is denied. In case of a "deny" decision the evaluation and respective actions execution is abandoned and an appropriate error message is returned to policy evaluation requestor through PEM1 interface. Otherwise the actions execution will be performed.

A rule is defined by the following rule model:

Name	Occurs	Description
ID	1	The unique id of the rule inside the policy.
Conflict Resolution Extensions	0..1	Necessary extensions for conflict resolution.
Conditions	1	The "if" part of a rule. Based on the evaluation result the actions execution proceeds or not.
Actions	1	The "then" part of a rule. Contains actions that must be enforced. Possible actions can be of any type (e.g. setLocationScope, forwardToService) and will be processed by the Policy Enforcement. The actions are defined according to a predefined schema.

Table A.1: Rule Model

A condition is composed of two parts:

1. Rule scope defines the input to which the policy applies.
2. Rule constraints define under which constraints the request is allowed.

The following picture depicts the schema details of the conditions. Because of its dimensions it is split on several parts.

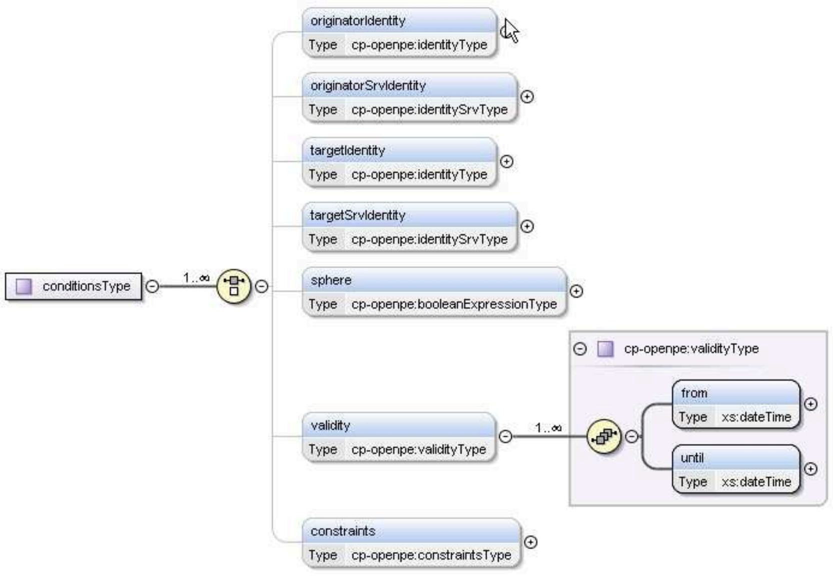

originatorIdentity
Type cp-openpe:identityType
originatorSrvIdentity
Type cp-openpe:identitySrvType
targetIdentity
Type cp-openpe:identityType
targetSrvIdentity
Type cp-openpe:identitySrvType
conditionsType
1..∞
sphere
Type cp-openpe:booleanExpressionType
cp-openpe:validityType
from
Type xs:dateTime
validity
Type cp-openpe:validityType
1..∞
until
Type xs:dateTime
constraints
Type cp-openpe:constraintsType

Figure A.1: Condition Model Part 1

The elements description of *conditionsType* is provided in the following table A.2:

[H] Name	Occurs	Description
Originator ID	0..1	Values of the accepted originator identity (e.g userID, roleID, personaID) for the rule in question in order to apply. Part of the rule scope.
OriginatorSrvIdentity	1	Accepted originator service identity.
TargetIdentity	0..1	Values of the accepted target identity (e.g userID, roleID, personaID) for the rule in question in order to apply. Part of the rule scope.
TargetSrvIdentity	0..*	Accepted target service identity.

Sphere	0..1	Part of the rule scope. Defines complex situations in which case the rule applies (it may also include validations of information type or reference). If it fails the rule evaluation will be abandoned but the overall evaluation process will continue.
Validity	1	Validity time of the rule. Part of the rule scope.
Constraints	0..1	Restrictions on the input data. It may also include constraints regarding requested information type of reference. It is the last part in a rule that is evaluated. If it fails, a deny decision is returned and the whole evaluation process dropped.

Table A.2: Condition Model Part 1

identityType is defined by the following schema:

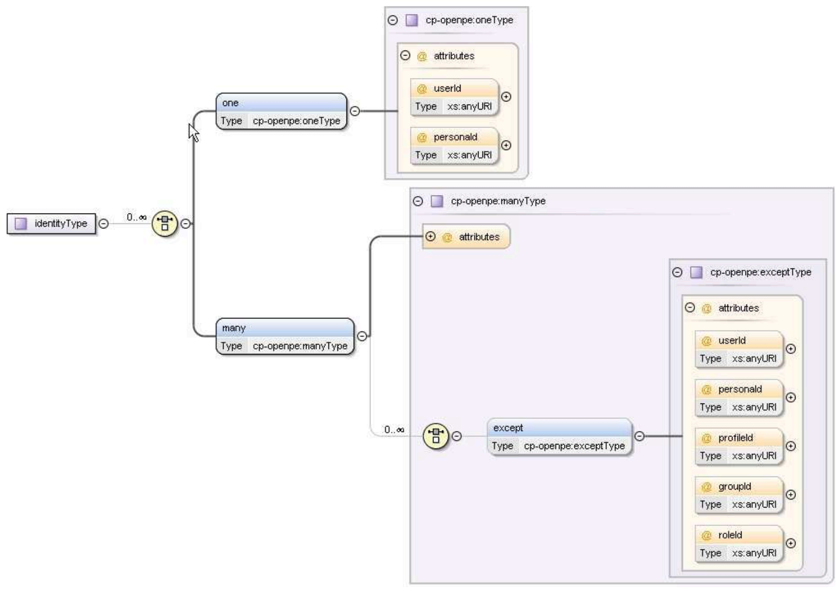


Figure A.2: Condition Model Part 2

The elements description of *identityType* is provided in the following table A.3.

Name	Occurs	Description

One	0..*	Defines for which identities the policy applies (userID, persona ID). It referes to identities that represent aggregation of identities with size one.
Many	0..*	Defines for which aggregation of identities the policy applies (profile ID, role ID, profile ID). In case the attributes are missing, it applies to all identities.
Except	0..*	Part of the many element. Describes the identities that should not be considered when deciding if "many" applies or not. ("Applies to groupX except userY")

Table A.3: Condition Model Part 2

The *identitySrvType* is similar with the one previously described but it applies to services and service providers identities. The following image depicts this element in detail:

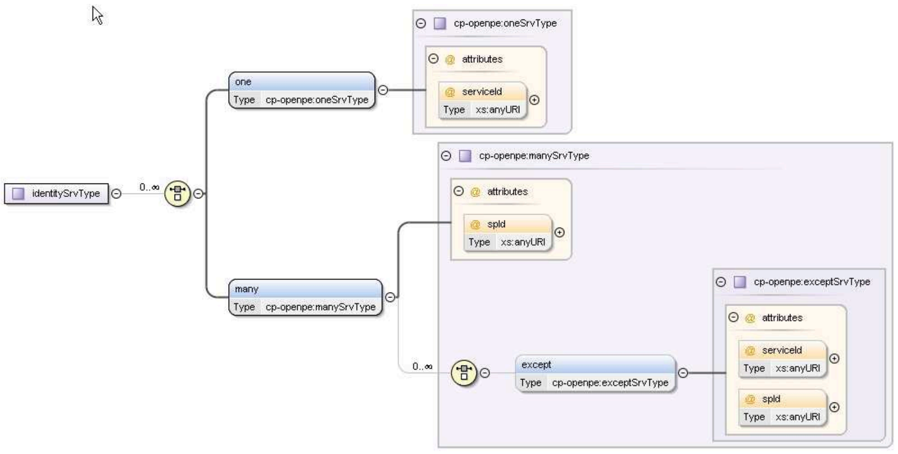


Figure A.3: Condition Model Part 3

The elements description of the *identitySrvType* element is provided in the following table A.6.

Name	Occurs	Description
One	0..*	Defines for which service identities the policy applies (serviceID). It referes to identities that represent aggregation of identities with size one.

Many	0..*	Defines for which aggregation of identities the policy applies (service provider ID). In case the attributes are missing, it applies to all identities.
Except	0..*	Part of the many element. Describes the identities that should not be considered when deciding if "many" applies or not. ("Applies to Service A except Service B")

Table A.4: Condition Model Part 3

The *constraintstype* element of a rule is defined below:

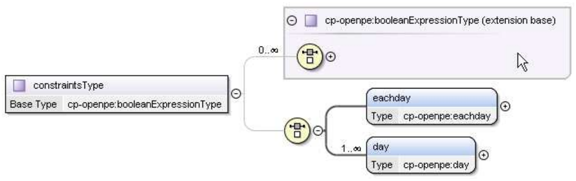

cp-openpe:booleanExpressionType (extension base)
0..∞
constraintsType
Base Type cp-openpe:booleanExpressionType
eachday
Type cp-openpe:eachday
1..∞
day
Type cp-openpe:day

Figure A.4: Condition Model Part 4

The elements description of the *constraintsType* element is provided in the following table A.5.

Name	Occurs	Description
booleanExpression Type	0..1	Defines constraints of the parameters in a boolean expression format.
eachDay	1	When the constraints are valid.
Day	0..*	Defines granular times for the contraints validity.

Table A.5: Condition Model Part 4

The *booleanExpressionType* element defines a boolean expression.It is composed of elements which may form a boolean expression when they are evaluated. The schema can be depicted in the following figure A.5.

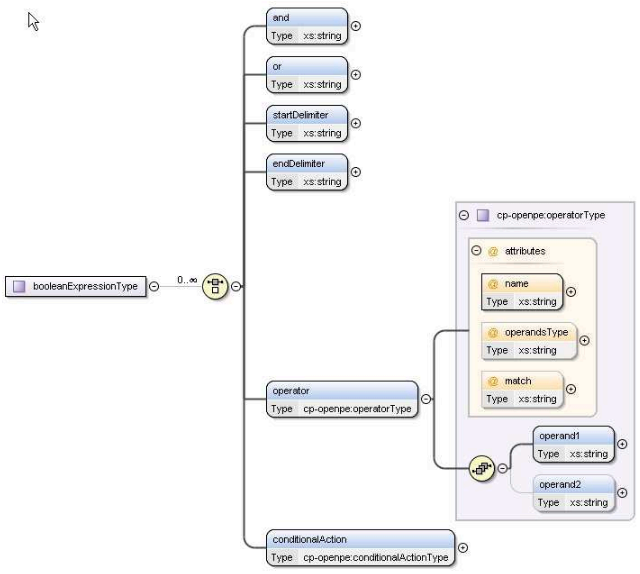


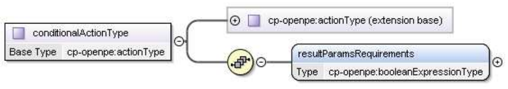
Figure A.5: Condition Model Part 5

The folowing table provides detailed information regarding the above defined schema.

Name	Occurs	Description
and	0..*	Boolean “and” operator.
or	0..*	Boolean “or” operator.
startDelimiter	0..*	Delimiter in a boolean expression (e.g. “(”)
endDelimiter	0..*	Delimiter in a boolean expression (e.g. “)”)
operator	0..*	Boolean operator defined by a comparison operation name (e.g. “equal”), operandsType (e.g. “string”) and match type (“noregx” = no regular expression, “regx” = regular expression). It has two operands (operand1 and operand2).
conditionalAction	0..*	Action with requirements on the returned values. It will be evaluated as true or false depending on the requirements resolution.

Table A.6: Condition Model Part 5

A *conditionalActionType* extends a normal action type by defining requirements on parameters resulted from the evaluation of the action. It is part of the previosly defined *booleanExpressionType*.



conditionalActionType
Base Type cp-openpe:actionType
cp-openpe:actionType (extension base)
resultParamsRequirements
Type cp-openpe:booleanExpressionType

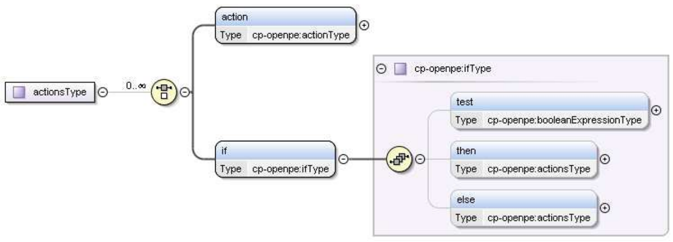
Figure A.6: Condition Model Part 6

The following table describes the main components of the *conditionalActionType* element:

Name	Occurs	Description
actionType	1	The action which is executed. The model of this type is defined in the following figure A.8.
resultParams Requirements	1	Define the requirements of the result parameters of the action execution. It evaluates as a boolean expression.

Table A.7: Condition Model Part 6

Actions represent the "then" part of a rule. They are executed either by the policy enforcer or by the policy evaluation requestor (e.g. a servic enabler). Actions, depicted in the following figure A.7 with the type "actionsType", are composed of several actions of type "actionType" and of type "ifType".



action
Type cp-openpe:actionType
actionsType
0..∞
cp-openpe:ifType
test
Type cp-openpe:booleanExpressionType
if
Type cp-openpe:ifType
then
Type cp-openpe:actionsType
else
Type cp-openpe:actionsType

Figure A.7: Action Model Part 1

The above figures describe the skeleton of the actions and provide the possibility of defining any kind of possible action. The following table A.8 describes more detailed the above schema elements. Sequentially described actions will are executed concurrently.

Name	Occurs	Description

Action	0..*	Action of type "actionType" which is described in the following figure A.8. The schema of this type is generic in order to cover most of the possible actions.
If	0..*	It is a special action used in order to minimize the number of rules which have common conditions. Integration of the "test" into the "actions" part instead to the "conditions" allows for a larger amount of actions to be set up into the same rule. If the later approach a rule should be created for each of the "then" and "else" part of "if". This granularity will be decided by the creator of the policy.

Table A.8: Action Model Part 1

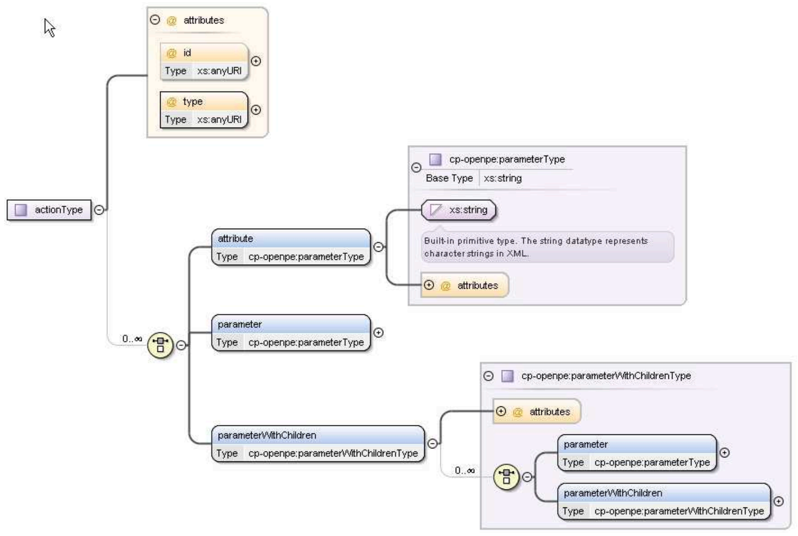

attributes
id
Type xs:anyURI
type
Type xs:anyURI
actionType
cp-openpe:parameterType
Base Type xs:string
xs:string
Built-in primitive type. The string datatype represents character strings in XML.
attributes
attribute
Type cp-openpe:parameterType
parameter
Type cp-openpe:parameterType
0..∞
cp-openpe:parameterWithChildrenType
attributes
parameter
Type cp-openpe:parameterType
parameterWithChildren
Type cp-openpe:parameterWithChildrenType
0..∞
parameterWithChildren
Type cp-openpe:parameterWithChildrenType

Figure A.8: Action Model Part 2

The following table A.9 describes in more detail the elements defined in the previous schema for type "actionType".

Name	Occurs	Description
id	1	Unique ID of the action inside a rule.
type	1	Predefined type of the action (e.g. "sendMessage").

attribute	0..*	Attributes of the action (e.g. Parlay X Location URL) necessary to prepare the action execution. Defined by attribute “name” and value.
parameter	0..*	Parameters used in the action execution (e.g. presentity, watcher). Defined by attribute “name” and value.
parameterWith Children	0..*	Representation of complex data types necessary for the action execution. Defined by attribute “name”.

Table A.9: Action Model Part 2

The above figures describe the skeleton of the actions and provide the possibility of defining any kind of possible action.

Appendix B

Specification of a Telecom JavaScript/JSON-RPC API

B.1 LoginHandler

LoginHandler does not represent a telecommunication service. It provides functions for authentication. The authentication mechanism uses browser cookies to save the session of the user. The function login checks if the user has permission to access IMS telecommunication services. If the user has permission a sessionId is stored in the cookie. All functions of IMS telecommunication services check if the stored sessionId is valid before the access to the service.

B.1.1 Method Summary

`static getSipUri()`
Returns the sipUri of the user who is logged in.

`static isAuthorized()`
Checks if there is a valid user session.

`static login (string username, string password)`
Login with username and password.

`static logout()`
Logs out the user and destroys the dedicated user session.

B.1.2 Method Details

B.1.2.1 getSipUri

`static getSipUri()`
Returns the sipUri of the user who is logged in.

Returns:
sipUri if somebody is logged in otherwise null

B.1.2.2 isAuthorized

`static isAuthorized()`
Checks if there is a valid user session.

Returns:
true or false

B.1.2.3 login

`static login(string username, string password)`
Login with username and password. The login sets a browser cookie that is needed for the user session. The cookie authenticates the user for using IMS telecommunication services. The value of the cookie is the sessionId.

Parameters:
- username
- password

Returns:
sessionId on success otherwise null

B.1.2.4 logout

`static logout()`
Logs out the user and destroys the dedicated user session.

Returns:
true on success

B.2 ServiceCapabilities

This interface is quering for service capabilities of a logged in user and returns also the service profile identifier of the user. The service capabilities are provided by a Service Broker.

B.2.1 Method Summary

`static getServices`
Returns services from the Service Capability Manager of the user who is logged in.

B.2.2 Method Details

B.2.2.1 getServices

`getServices(function userCallback)`
Get information about the last initiated, connected or disconnected VoIP call.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.
result contains:
- result.serviceCapabilities (list of services)
- result.serviceProfile (list of profile identifieres)

Returns:
including a list of service profiles definition and a list of services

B.3 AddressBook

Manage your contacts. Informations are stored on xdms resource-list

B.3.1 Method Summary

```
static addMember(function userCallback, string
groupName, string sipUri, string displayName)
```
Adds a new member to an existing group.

`static createGroup(function userCallback, string groupName, string displayName)`
Creates a new group in your resource list.

`static deleteGroup(function userCallback, string groupName)`
Removes a group from your resource list.

`static deleteMember(function userCallback, string groupName, string sipUri)`
Removes a member from a group.

`static getAddressBook(function userCallback)`
Get all contacts of your addressbook.

`getGroups(function userCallback)`
Get a list of all groups in your address book.

`static getMembers(function userCallback, string groupName)`
Get all members from a group.

`static getMyProfile(function userCallback)`
Get your own profile.

`static getProfile(function userCallback, string sipUri)`
Get the profile of a user.

`static setProfile(function userCallback, UserProfile userProfile)` Set your own profile.

B.3.2 Method Details

B.3.2.1 addMember

`static addMember(function userCallback, string groupName, string sipUri, string displayName)`
Adds a new member to an existing group.

Parameters:

userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success
groupName - user will be added to this group
sipUri - sip uri of the new member
displayName - display name of the new member

Returns:

no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.3.2.2 createGroup

```
static createGroup(function userCallback, string
groupName, string displayName)
```

Creates a new group in your resource list.

Parameters:

userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

groupName - internal name for the new group
displayName - display name for the new group

Returns:

no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.3.2.3 deleteGroup

```
static deleteGroup(function userCallback, string
groupName)
```
Removes a group from your resource list.

Parameters:

userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

groupName - internal name for the group that should be removed

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns true on success

B.3.2.4 deleteMember

```
static deleteMember(function userCallback, string
groupName, string sipUri)
```
Removes a member from a group.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

groupName - user will be removed from this group
sipUri - sip uri of the member

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.3.2.5 getAddressBook

`static getAddressBook(function userCallback)`
Get all contacts of your addressbook.

Parameters:

userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result[] contains array of
AddressBookBean: - result[].groupName
- result[].group_SipUri
- result[].MemberBean[]
MemberBean[] contains
- displayName
- sipUri

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns array of AddressBookBean.

B.3.2.6 getGroups

`static getGroups(function userCallback)`
Get a list of all groups in your address book.

Parameters:

userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result[] contains array of groups

Returns:

no return value - callback function is invoked when message returns.
If called synchronous: returns true on success

B.3.2.7 getMembers

```
static getMembers(function userCallback, string
groupName)
```
Get all members from a group.

Parameters:

userCallback - userCallback(result[])function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result[] contains:
- list of sip uris
groupName - group name

Returns:

no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.3.2.8 getMyProfile

```
static getMyProfile(function userCallback)
```
Get your own profile.

Parameters:

userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:

- result.address
- result.communicationAddresses[]
- result.profileSipUri
- result.displayName
- result.givenName
- result.familyName
- result.birthDate
- result.addressCountry

- result.addressRegion
- result.addressCity
- result.addressStreet
- result.addressStreetNumber
- result.addressPostalCode
- result.gender
- result.freetext
- result.hobbies[]
- result.favouriteLinks[]

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns user profile

B.3.2.9 getProfile

`static getProfile(function userCallback, string sipUri)`
Get the profile of a user.

Parameters:

userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains user profile:

- result.address
- result.communicationAddresses[]
- result.profileSipUri
- result.displayName
- result.givenName
- result.familyName
- result.birthDate
- result.addressCountry
- result.addressRegion
- result.addressCity
- result.addressStreet
- result.addressStreetNumber
- result.addressPostalCode
- result.gender
- result.freetext

- result.hobbies[]
- result.favouriteLinks[]

sipUri - sip uri of the user

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns user profile

B.3.2.10 setProfile

```
static setProfile(function userCallback, UserProfile
userProfile)
```
Sets a profile.

Parameters:

userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

userProfile - user profile of the user

- communicationAddresses[]
- profileSipUri
- displayName
- givenName
- familyName
- birthDate
- addressCountry
- addressRegion
- addressCity
- addressStreet
- addressStreetNumber
- addressPostalCode
- gender
- freetext
- hobbies[]

- favouriteLinks[]

Returns:

no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.4 Call

Functions for initiating or receiving a VoIP call on a locally installed client.

B.4.1 Method Summary

`static answerCall(function userCallback)`
Connects an incoming call.

`static cancelCall(function userCallback)`
Cancels an initiated VoIP call that is not yet connected.

`static endCall(function userCallback)`
Hangs up a connected VoIP call.

`static getCallInformation(function userCallback)`
Get information about the last initiated, connected or disconnected VoIP call.

```
static makeCall(function userCallback, string
calleeSipUri)
```
Initiates a VoIP call.

`static onConnectionStatusChange(function userFunction)`
Assign a function that is invoked when call connection status changes.

`static onIncomingCall(function userFunction)`
Assigns a function that is invoked when an incoming call arrives and registers this instance (browser or widget) for receiving incoming calls.

B.4.2 Method Details

B.4.2.1 answerCall

`static answerCall(function userCallback)`
Connects an incoming call.

Parameters: userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.4.2.2 cancelCall

`static cancelCall(function userCallback)`
Cancels an initiated VoIP call that is not yet connected.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

Returns: no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.4.2.3 endCall

`static endCall(function userCallback)`
Hangs up a connected VoIP call.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when

method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.4.2.4 getCallInformation

`static getCallInformation(function userCallback)`
Get information about the last initiated, connected or disconnected VoIP call.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
- result.callStatus
- result.duration
- result.startTime
- result.terminationCause

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns result parameters

B.4.2.5 makeCall

`static makeCall(function userCallback, string calleeSipUri)` Initiates a VoIP call.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

calleeSipUri - sip uri of the user that will be called

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.4.2.6 onConnectionStatusChange

`static onConnectionStatusChange(function userFunction)`
Assign a function that is invoked when call connection status changes.

Parameters:
userFunction - userFunction(status) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
status - CONNECTED / DISCONNECTED

Returns:
no return value

B.4.2.7 onIncomingCall

`static onIncomingCall(function userFunction)`
Assigns a function that is invoked when an incoming call arrives and registers this instance (browser or widget) for receiving incoming calls.

Parameters:
userFunction - userFunction(incomingCallSipUri) - function that is invoked when an incoming call arrives

Returns:
no return value

B.5 ConferenceCall

Conference Calls connect multiple users. The initiator of the conference may invite new participants and disconnect users.

B.5.1 Method Summary

`static createConference(function userCallback, string description, int maxParticipants)`
Creates a call conference with a various number of user.

`disconnectParticipant(function userCallback, string sipUri)`
Disconnects a user from a running conference call.

`static endConference(function userCallback)`
Ends a running conference call.

`static getCallInformation(function userCallback)`
Get information about the actual call conference.

`static getParticipantInfo(function userCallback, string sipUri)`
Get information about a conference participant.

`static getParticipants(function userCallback)`
Get a list of participants for the actual conference call.

`static inviteParticipant(function userCallback, string sipUri)`
Adds a new participant to a running conference call.

B.5.2 Method Details

B.5.2.1 createConference

`static createConference(function userCallback, string description, int maxParticipants)`
Creates a call conference with a various number of user. At the beginning the conference has no user. Use inviteParticipant() function to add new users.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success
description - description for the conference
maxParticipants - max. number of participants

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns true on success

B.5.3 disconnectParticipant

```
static disconnectParticipant(function userCallback,
string sipUri)
```
Disconnects a user from a running conference call.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success
sipUri - sip uri of the participant

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns true on success

B.5.3.1 endConference

static endConference(function userCallback)
Ends a running conference call.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns true on success.

B.5.3.2 getCallInformation

```
static getCallInformation(function userCallback)
```
Get information about the actual call conference.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains callInfo:
- result.duration
- result.conferenceDescription
- result.conferenceIdentifier
- result.maximumNumberOfParticipants
- result.numberOfParticipants
- result.owner
- result.startTime
- result.status

Returns: no return value - function is invoked when message returns.
If called synchronous: returns callInfo

B.5.3.3 getParticipantInfo

```
static getParticipantInfo(function userCallback, string
sipUri)
```
Get information about a conference participant.

Parameters: userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
- result.sipUri
- result.startTime

- result.status
sipUri - participants sip uri

Returns: no return value - callback function is invoked when message returns
If called synchronous: returns participant

B.5.3.4 getParticipants

`static getParticipants(function userCallback)`
Get a list of participants for the actual conference call.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
array of participants - result[].sipUri
- result[].startTime
- result[].status

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns participant array

B.5.3.5 inviteParticipant

```
static inviteParticipant(function userCallback,
string sipUri)
```
Adds a new participant to a running conference call

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success
sipUri - sip uri of the new participant

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.6 Location

Location service group provides functions for getting location information of yourself and other users.

B.6.1 Method Summary

`static getLocation(function userCallback, string sipUri)`
Get location information for a user

`static getLocationForGroup(function userCallback, string sipUris[])`
Get location information for a list of users.

`static getTerminalDistance(function userCallback, string sipUris)`
Computes the distance between yourself and another client.

`static onChangeLocation(function userFunction)`
Assigns a function that is invoked when location of one of the subscribed buddies changes.

B.6.2 Method Details

B.6.2.1 getLocation

`static getLocation(function userCallback, string sipUri)`
Get location information for a user.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains location information:
- result.timestamp
- result.accuracy
- result.altitude
- result.latitude
- result.longitude

sipUri - sip uri of the user

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns location information

B.6.2.2 getLocationForGroup

```
static getLocationForGroup(function userCallback, string
sipUris[])
```
Get location information for a list of users.

Parameters:
userCallback - userCallback(result[]) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
- result[].timestamp
- result[].accuracy
- result[].altitude
- result[].latitude
- result[].longitude

sipUris[] - sip uris of the user

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns location information

B.6.2.3 getTerminalDistance

```
static getTerminalDistance(function userCallback, string
sipUris)
```
Computes the distance between yourself and another client.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
distance in meter

sipUris - sip URI of the user

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns distance in meter

B.6.2.4 onChangeLocation

`static onChangeLocation(function userFunction)`
Assigns a function that is invoked when location of one the subscribed buddies changes.

Parameters:
userFunction - userFunction(sipUri, latitude, longitude) - function that is invoked when an incoming location change arrives.

Returns:
no return value

B.7 Messaging

Messaging allows to send and receive instant messages from and to other IMS users.

B.7.1 Method Summary

`static getDeliveryStatus(function userCallback)`
Get delivery information for a message that was just sent.

`static onIncomingMessage(function userFunction)`
Assigns a function that is invoked when an incoming message arrives and registers this client (browser or widget) for receiving messages.

```
static sendMessage(function userCallback, string
calleeSipUris[], string subject, string content)
```
Sends a sip message to one or more users.

B.7.2 Method Details

B.7.2.1 getDeliveryStatus

`static getDeliveryStatus(function userCallback)`
Get delivery information for a message that was just sent.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains delivery status arrray:
- result[].addressURI
- result[].status

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns delivery status arrray.

B.7.2.2 onIncomingMessage

`static onIncomingMessage(function userFunction)`
Assigns a function that is invoked when an incoming message arrives and registers this client (browser or widget) for receiving messages.

Parameters:
userFunction - userFunction(from,subject,content) - function that is invoked when an incoming message arrives.

Returns:
no return value

B.7.2.3 sendMessage

```
static sendMessage(function userCallback, string
calleeSipUris[], string subject, string content)
```
Sends a sip message to one or more users.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when

method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success
calleeSipUris[] - array of sip uris who will receive the message
subject - message subject
content - message content

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.8 Presence

Functions to get presence state of other users and to set your own presence state.

B.8.1 Method Summary

`static getUserPresence(function userCallback, string sipUri)`
Get the actual presence state of a user.

`static getUserPresenceList(function userCallback, string sipUri)`
Get the actual presence state for a list of users.

`static onChangePresence(function userFunction)`
Assigns a function that is invoked when presence status of a person changes that you are subscribed to.

`static publish(function userCallback, string presenceStatus)`
Set your own presence status.

`static subscribeToAddressBook(function userCallback)`
Subscribe to presence state of all user in a address book (usually OMA XDMS resource-list).

`static subscribeToMemberList(function userCallback, string userSipUris[])`

Subscribe to a list of users that you want to get noticed when their presence state changes.

B.8.2 Method Details

B.8.2.1 getUserPresence

`static getUserPresence(function userCallback, string sipUri)`

Get the current presence state of a user.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains presence status info:
- result.sipUri
- result.lastChange
- result.presenceStatus: ONLINE|OFFLINE|AWAY|BUSY

sipUri - sip URI of the user to retrieve the presence state

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns presence status info

B.8.2.2 getUserPresenceList

`static getUserPresenceList(function userCallback, string sipUri)`

Get the actual presence state for a list of users.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains presence status info:
- result.sipUri
- result.lastChange
- result.presenceStatus: ONLINE|OFFLINE|AWAY|BUSY

sipUri - sip uri of the user, from whom you want to get the actual presence status.

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns presence status info

B.8.2.3 onChangePresence

`static onChangePresence(function userFunction)`
Assigns a function that is invoked when presence status of a subscribed identity changes.

Parameters:
userFunction - userFunction(sipUri, presenceStatus) function that is invoked when any of your subscribed buddies changes its presence state.

Returns:
no return value.

B.8.2.4 publish

`static publish(function userCallback, string presenceStatus)`
Set a presence status.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success presenceStatus - ONLINE|OFFLINE|AWAY|BUSY

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.8.2.5 subscribeToAddressBook

`static subscribeToAddressBook(function userCallback)`
Subscribe to presence state of all user in a address book (usually OMA XDMS resource-list). Use onChangePresence() to set the function that should be invoked when user changes presence state.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.8.2.6 subscribeToMemberList

```
static subscribeToMemberList(function userCallback,
string userSipUris[])
```
Subscribe to a list of users that you want to get noticed when their presence state changes. Use onChangePresence() to set the function that should be invoked when user changes presence state.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success
userSipUris[] - array of sip uris of the user you want to get noticed when presence state changes

Returns:
no return value - callback function is invoked when message returns
If called synchronous: returns true on success

B.9 ShortMessaging

Use Short Messaging Services to mobile phones.

B.9.1 Method Summary

```
static getDeliveryStatus(function userCallback)
```
Get delivery status for a sms just sent

```
static sendSMS(function userCallback, string
calleeSipUris[], string subject, string content)
```
Sends a SMS to one or more mobile phones.

B.9.2 Method Details

B.9.2.1 getDeliveryStatus

```
static getDeliveryStatus(function userCallback)
```
Get delivery status for a sms just sent.

Parameters:
userCallback - userCallback(result[]) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains smsstatus array:
- result[].phoneNumber
- result[].deliveryStatus

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns smsstatus array.

B.9.2.2 sendSMS

```
static sendSMS(function userCallback, string
calleeSipUris[], string subject, string content)
```
Sends a SMS to one or more mobile phones.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success
calleeSipUris[] - receiver sip uris
subject - message subject
content - message content

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns true on success.

B.10 ThirdPartyCall

Third Party Call initiates a call between two external participants. Participants can be SIP addresses like sip:bob@yourdomain.com or classic telephone numbers.

B.10.1 Method Summary

`static cancelCall(function userCallback)`
Cancels an initiated call that is NOT yet connected.

`static endCall(function userCallback)`
Hangs up a connected third party call.

`static getCallInformation(function userCallback)`
Get information about the last initiated or connected third party call.

B.10.2 Method Details

B.10.2.1 cancelCall

`static cancelCall(function userCallback)`
Cancels an initiated call that is not yet connected.

Parameters:
userCallback . userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains: true on success

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: return value: true on success.

B.10.2.2 endCall

`static endCall(function userCallback)`
Terminates a call to a connected third party call.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when

method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: return value: true on success

B.10.2.3 getCallInformation

`static getCallInformation(function userCallback)`
Get information about the last initiated or connected third party call

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
- result.callStatus
- result.duration
- result.startTime
- result.terminationCause

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns result (like asynchronous result variable)

B.10.2.4 makeCall

```
static makeCall(function userCallback, string
participant1, string participant2)
```
Initiates a call between two participants. The phone of the second participant rings first.

Parameters:
userCallback - userCallback(result) - function that is asynchronously invoked when method returns. Set userCallback parameter to null to invoke a synchronous call.

result contains:
true on success
participant1 - SIP URI or telephone number of the first participant
participant2 - SIP URI or telephone number of the second participant

Returns:
no return value - callback function is invoked when message returns.
If called synchronous: returns true on success.

Appendix C

Profile Policy Example

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rule definedFor="request" id="r1" xmlns="http://urn/ietf/params/
    xml/ns/common-policy">
    <conditions>
        <validity>
            <from>2008-05-27T14:11:00.943Z</from>
            <until>2010-05-27T14:11:00.943Z</until>
        </validity>
        <constraints>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>PresenceConsumerService</operand2>
            </operator>
            <or></or>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>PresenceNotification</operand2>
            </operator>
            <or></or>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>SendMessageService</operand2>
            </operator>
            <or></or>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>SendSMSService</operand2>
            </operator>
            <or></or>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>GroupService</operand2>
            </operator>
            <or></or>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>MessageNotificationManagerService</
                    operand2>
            </operator>
            <or></or>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>PresenceNotificationService</operand2>
            </operator>
            <or></or>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>MessageNotification</operand2>
            </operator>
            <or></or>
            <operator match="noregx" operandsType="string" name="
                equal">
                <operand1>fn:getTargetEnabler()</operand1>
                <operand2>GroupManagementService</operand2>
            </operator>
        </constraints>
    </conditions>
    <actions/>
</rule>
```

Listing C.1: Profile Policy

Appendix D

Parlay X Presence Message Manipulation

SOAP envelop of the Parlay X Presence service request:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getUserPresenceResponse xmlns:ns2="http://www.csapi.org
          /schema/parlayx/presence/consumer/v3_2/local" xmlns:ns3=
          "http://www.csapi.org/schema/parlayx/common/v3_1">
         <ns2:result>
            <lastChange>2010-06-30T17:44:39</lastChange>
            <typeAndValue>
               <UnionElement>Communication</UnionElement>
               <Communication>
                  <means>
                     <priority>0.0</priority>
                     <contact>sip:aramis@fokus.fraunhofer.de</
                         contact>
                     <type>Chat</type>
                     <status>On</status>
                  </means>
               </Communication>
            </typeAndValue>
         </ns2:result>
         <ns2:result>
            <lastChange>2010-06-30T17:44:39</lastChange>
            <note/>
            <typeAndValue>
               <UnionElement>Activity</UnionElement>
               <Activity>Busy</Activity>
            </typeAndValue>
         </ns2:result>
      </ns2:getUserPresenceResponse>
   </S:Body>
```

```
</S:Envelope>
```

Listing D.1: Parlay X Presence Service SOAP request

SOAP envelop of the manipulated response by the Service Broker:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getUserPresenceResponse xmlns:ns2="http://www.csapi.org
          /schema/parlayx/presence/consumer/v3_2/local" xmlns:ns3=
          "http://www.csapi.org/schema/parlayx/common/v3_1">
         <ns2:result>
            <lastChange>2010-06-30T17:44:39</lastChange>
            <typeAndValue>
               <UnionElement>Communication</UnionElement>
               <Communication>
                  <means>
                     <priority>0.0</priority>
                     <contact>sip:aramis@fokus.fraunhofer.de</
                         contact>
                     <type>Chat</type>
                     <status>Off</status>
                  </means>
               </Communication>
            </typeAndValue>
         </ns2:result>
      </ns2:getUserPresenceResponse>
   </S:Body>
</S:Envelope>
```

Listing D.2: Parlay X Presence Service SOAP response with deleted Activity set

Appendix E

PEM1 Input/Output Template Examples

SOAP envelop of the PEM1 input request containing the originator's and the recipient's address as well as context information characterizing the message itself. The information are extracted from the incoming SIP request.

```
<ns2:policyInputData ... >
  <policyInputTemplate>
      <requestMessage>true</requestMessage>
      <policyIdentifiers>
         <originatorId>sip:alice@open-ims.test</originatorId>
         <targetId>sip:bob@open-ims.test</targetIdD>
      </policyIdentifiers>
      <targetService name="ConvergedMessaging">
         <targetServiceOperation namespace=""
            name="InterworkingDecision">
            <operationParameter name="Message-Mode">
               PAGER_MODE
            </operationParameter>
            <operationParameter name="Message-Size">
               729
            </operationParameter>
         </targetServiceOperation>
      </targetService>
   </policyInputTemplate>
</ns2:policyInputData>
```

Listing E.1: PEM1 input template

SOAP envelop of the PEM1 output response received from the Service Broker. The user policy has been evaluated true (lines 3-4) and the `<enforcementData>` element will be taken into account for the outbound messaging channel selection process. Line 8 and line 10 contains the URIs to be used for the execution whereas line 7 determines to route the message request to the SMS enabler.

```xml
<ns2:policyOutputData ... >
   <policyOutputTemplate>
      <StatusCode>2101</StatusCode>
      <StatusText>Successfully evaluated</StatusText>
      <enforcementData>
         <enforcementAction id="4">
            <enforcementActionOperation name="sms">
               <enforcementActionOperationParameters>
                  tel:09969955776
               </enforcementActionOperationParameters>
               <enforcementActionOperationParameters>
                  tel:01243435353
               </enforcementActionOperationParameters>
            </enforcementActionOperation>
         </enforcementAction>
      </enforcementData>
   </policyOutputTemplate>
</ns2:policyOutputData>
```

Listing E.2: PEM1 output template