universität
wien

DISSERTATION

Titel der Dissertation

"A Framework for SLA-Centric Service-Based Utility Computing"

Verfasser
Irfan Ul Haq

angestrebter akademischer Grad
Doktor der Technischen Wissenschaften (Dr. techn.)

Wien, im December 2010

Studienkennzahl lt. Studienblatt: A 786 881
Dissertationsgebiet lt. Studienblatt: Informatik
Betreuer: Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta

To my mother Sabiha Nazli

Contents

Abstract vii

1. Introduction 1
- 1.1. Motivation . . . 1
- 1.2. Hypothesis and Open Questions . . . 4
- 1.3. Vision . . . 5
- 1.4. Thesis Structure and Contribution . . . 6
- 1.5. Disclosure and Acknowledgements . . . 10

2. Basic Concepts and State of The Art 13
- 2.1. Service Oriented Architecture (SOA) . . . 13
- 2.2. Service Based Utility Computing . . . 15
- 2.3. Service Level Agreements (SLA) . . . 16
  - 2.3.1. SLA Specification . . . 17
  - 2.3.2. SLA Negotiation and Renegotiation . . . 18
  - 2.3.3. SLA Formalization . . . 18
  - 2.3.4. SLA Aggregation . . . 19
  - 2.3.5. Rule-Based SLAs . . . 19
- 2.4. SLA Languages and Their Implementations . . . 20
  - 2.4.1. SLAng . . . 20
  - 2.4.2. WSLA . . . 21
  - 2.4.3. WS-Agreement . . . 22
  - 2.4.4. Comparative Analysis . . . 24
- 2.5. Related Work . . . 27
  - 2.5.1. Virtual Organizations and NESSI business models . . . 27
  - 2.5.2. Optimization against QoS constraints . . . 27
  - 2.5.3. Views and e-Contracts in Workflows . . . 28
  - 2.5.4. Distributed Trust and Security . . . 29
- 2.6. Projects Relating SLA Management . . . 29
  - 2.6.1. SLA@SOI . . . 31
  - 2.6.2. FOSII . . . 31
  - 2.6.3. RBSLA . . . 33
  - 2.6.4. COSMA . . . 33
  - 2.6.5. SLA4D-Grid . . . 33
  - 2.6.6. LIBRA . . . 34

2.6.7. MASCHINE . . . 34
2.6.8. mOSAIC . . . 34
2.6.9. SOA4All . . . 34
2.6.10. BREIN . . . 35
2.6.11. NEXOF-RA . . . 35
2.6.12. MASTER . . . 37
2.6.13. Romulus . . . 37
2.7. Summary . . . 38

3. A Framework for SLA Centric Service-Based Utility Computing 39
3.1. Service-Based Utility Computing . . . 39
3.2. A Framework for SLA Centric Utility Computing . . . 40
3.2.1. Architecture . . . 40
3.2.2. Phased Process Model . . . 43
3.3. Motivational Scenario . . . 44
3.4. Summary . . . 45

4. SLA-Based Selection and Negotiation of Services 47
4.1. Background . . . 47
4.2. Running Example – User-Driven Service Selection . . . 48
4.3. SLA-Based Selection of Services . . . 49
4.3.1. Formal Model for Service Selection . . . 50
4.3.2. Algorithms for Service Selection . . . 54
4.4. Running Example – Service Value Chains . . . 63
4.5. SLA Negotiation of Configurable Services . . . 64
4.5.1. Dynamic Configuration of SLA Offers . . . 65
4.5.2. Formal Model for Negotiation and Renegotiation of Configurable Services . . . 66
4.5.3. Running Example – Negotiation . . . 67
4.5.4. An SLA Negotiation/Renegotiation Protocol for Configurable Services 69
4.6. Summary . . . 71

5. Hierarchical Aggregation of SLA Choreography 73
5.1. Background . . . 73
5.2. SLA Choreography . . . 74
5.3. Formal Model of SLA Choreography and its Aggregation . . . 75
5.3.1. SLA and SLA Choreography . . . 75
5.3.2. SLA Views and SLA Choreography . . . 78
5.3.3. Aggregation of Service Terms . . . 81
5.3.4. Aggregation of Guarantee Terms . . . 82
5.4. Aggregation Patterns for SLA Choreography . . . 83
5.4.1. Composite Service Provision Patterns . . . 83
5.4.2. Enterprize Structural Patterns . . . 86

5.5. Running Example – Aggregation of SLAs . . . 89
5.6. Summary . . . 92

6. Hierarchical SLA Validation and Distributed Trust Management 93
6.1. Background and Challenges . . . 93
6.2. Enabling Requirements . . . 94
6.2.1. Multi Agent System . . . 94
6.2.2. SLA-Views . . . 95
6.2.3. Rule Based SLAs . . . 96
6.2.4. Distributed Trust Model . . . 96
6.3. A Validation Framework for Hierarchical SLA Choreographies . . . 96
6.3.1. Rule Responder Architecture . . . 97
6.3.2. Rule Based Service Level Agreements (RBSLA) . . . 98
6.3.3. Distributed Trust Model . . . 100
6.3.4. Rule based Validation of SLA Choreographies . . . 103
6.3.5. Delegation of Validation . . . 106
6.4. Running Example – Hierarchical SLA Validation . . . 107
6.5. Role of Validation and Trust Model During Service Selection . . . 111
6.6. Summary . . . 111

7. Implementation 113
7.1. Optimization of SLA-Based Service Selection . . . 113
7.1.1. Use Case Scenario . . . 113
7.1.2. Simulation Setup and Tools . . . 114
7.1.3. Performance Analysis . . . 115
7.2. Validation of SLA Choreographies . . . 121
7.2.1. Use Case Scenario . . . 121
7.2.2. Assumptions . . . 121
7.2.3. Simulation Setup and Tools . . . 122
7.2.4. Architecture . . . 124
7.2.5. The Validation Process . . . 126
7.2.6. Conclusion . . . 132
7.3. Summary . . . 133

8. Extensions and Applications of the SLA-centric Framework 135
8.1. Extension of the Validation Framework . . . 135
8.2. Extension of the Negotiation Model . . . 137
8.2.1. Formal Model . . . 137
8.2.2. Negotiation Protocol with Payment . . . 138
8.3. SLA Oriented Service Selection for Reverse Auction based Systems . . . 139
8.4. SLA Aggregation Patterns in Business Processes . . . 140
8.5. Applications in Enterprize 2.0 . . . 140
8.6. Summary . . . 141

9. Conclusion **143**

A. Installation Guide for the Simulation Environment for Optimized Service Selection **145**
A.1. Compiling the Single-Machine Optimization Cores 145
A.2. Compiling and Installing the CORBA Optimization Cores 145
A.3. Unsupported Variants of the Optimization Core 146
A.4. Executing the Optimizer through Kepler 146
A.5. Executing the Optimizer Directly 147
A.6. Testcases 147

B. Installation Guide for the Simulation Environment for Hierarchical SLA Validation **149**
B.1. Setting Up Eclipse Environment 149
B.1.1. Prerequisites 149
B.1.2. Obtaining Source Code 149
B.1.3. Configuring Mule 149
B.1.4. Ports Required by SLAValidator 150
B.2. Setting Up Apache HTTP Server 150
B.3. Setting Up TomCat Server 150
B.3.1. Configuring the first level PA Source 150
B.3.2. Configuring the second level PA Source 152
B.4. Starting Up 152

Bibliography **153**

List of Figures

1.1. Gartner Hype Cycle(2009) for Emerging Technologies 2
1.2. Hierarchical Service Level Agreements and its Choreography 4
1.3. Organization of the Thesis 7

2.1. Overview of WSLA structure 21
2.2. Overview of WS-Agreement structure 23
2.3. Comparison of SLA Languages 25
2.4. SLA@SOI Project 30
2.5. LoM2HiS model as part of FOSII Infrastructure 32
2.6. moSAIC Architecture 35
2.7. SOA4All Architecture 36
2.8. BREIN Objectives 36
2.9. NEXOF Architecture 37

3.1. An SLA-Centric Framework for Service-Based Utility Computing 42
3.2. Phased Process Model for an SLA-Centric Framework for Service-Based Utility Computing 44
3.3. Motivational Scenario 45

4.1. Running Example-User Driven Service Selection 49
4.2. An example of the Knapsack problem with K representing total cost, T total time, and h as degree of happiness 50
4.3. The two phases of the optimization algorithm 55
4.4. Flow Chart for the algorithm based on updating heuristics 62
4.5. Service Value Chains 64
4.6. Geometric interpretation of the negotiation function g and the distance d_w . 67
4.7. (a) Client's Preferences, (b) Service Provider's Options 68
4.8. The storage service is available from two service providers: by an independent provider with low reputation and through "rendering workflow", which has higher reputation 69
4.9. Negotiation Protocol for SLA configuration 70

5.1. structure of an agreement in accordance with WS-Agreement specification . 75
5.2. Hierarchical Aggregation of SLAs 77
5.3. Different Views in SLA Choreography 78
5.4. SLA Aggregation Patterns for Service Composition 83

5.5. SLA Aggregation Pattern for Virtual Enterprize Organization (VEO) 87
5.6. SLA Aggregation Pattern for P2P relationships in a Value Network 87
5.7. SLA Aggregation Pattern for Cycles 88
5.8. SLA Aggregation Pattern for Nesting 89
5.9. Running Example - Hierarchical Aggregation of SLAs 90

6.1. Validation of SLA Choreographies as a cross-section of enabling technologies 95
6.2. Validation Framework for SLA Choreographies, where red boundaries indicate the directly contributing notions 97
6.3. Rule Responder Services for SLA Validation 99
6.4. The correspondence between the PKI and reputation based systems and to the Rule Responder architecture 102
6.5. Query of PA-a about reputation of PA-c to OA-A and then redirected to TRC 103
6.6. Every SLA-View corresponds to a Personal Agent 104
6.7. Role Activity Diagram for a simple Query-Answer Conversation 105
6.8. Running Example - Hierarchical SLA Validation 106
6.9. Validation through distributed query decomposition 107

7.1. Speedup analysis with 25 service classes 116
7.2. Speedup analysis with 28 service classes 116
7.3. Speedup analysis with 30 service classes 116
7.4. Speedup analysis with 32 service classes 116
7.5. Performance comparison branch and bound vs. heuristic update 118
7.6. Happiness ratio between heuristic solution and optimum 119
7.7. Effects of Service Failures 121
7.8. Prototype Architecture) 125
7.9. External Agent with RuleML-Based Query) 127

8.1. Architecture of LAYSI 136
8.2. A holistic SLA validation framework 137
8.3. gSET Architecture 139
8.4. A blackboard architecture for a reverse auction based service selection system 140

Abstract

Service oriented Utility Computing paves the way towards realization of service markets, which promise metered services through negotiable Service Level Agreements (SLA). A market does not necessarily imply a simple buyer-seller relationship, rather it is the culmination point of a complex chain of stake-holders with a hierarchical integration of value along each link in the chain. In service value chains, services corresponding to different partners are aggregated in a producer-consumer manner resulting in hierarchical structures of added value. SLAs are contracts between service providers and service consumers, which ensure the expected Quality of Service (QoS) to different stakeholders at various levels in this hierarchy. *This thesis addresses the challenge of realizing SLA-centric infrastructure to enable service markets for Utility Computing.*

Service Level Agreements play a pivotal role throughout the life cycle of service aggregation. The activities of service selection and service negotiation followed by the hierarchical aggregation and validation of services in service value chain, require SLA as an enabling technology. *This research aims at a SLA-centric framework where the requirement-driven selection of services, flexible SLA negotiation, hierarchical SLA aggregation and validation, and related issues such as privacy, trust and security have been formalized and the prototypes of the service selection model and the validation model have been implemented.* The formal model for User-driven service selection utilizes Branch and Bound and Heuristic algorithms for its implementation. The formal model is then extended for SLA negotiation of configurable services of varying granularity in order to tweak the interests of the service consumers and service providers.

The possibility of service aggregation opens new business opportunities in the evolving landscape of IT-based Service Economy. A SLA as a unit of business relationships helps establish innovative topologies for business networks. One example is the composition of computational services to construct services of bigger granularity thus giving room to business models based on service aggregation, Composite Service Provision and Reselling. This research introduces and formalizes the notions of SLA Choreography and hierarchical SLA aggregation in connection with the underlying service choreography to realize SLA-centric service value chains and business networks. The SLA Choreography and aggregation poses new challenges regarding its description, management, maintenance, validation, trust, privacy and security. The aggregation and validation models for SLA Choreography introduce concepts such as: SLA Views to protect the privacy of stakeholders; a hybrid trust model to foster business among unknown partners; and a PKI security mechanism coupled with rule based validation system to enable distributed queries across heterogeneous boundaries. A distributed rule based hierarchical SLA validation system is designed to demonstrate the practical significance of these notions.

Zusammenfassung

Dienstleistungsorientierte Datenverarbeitung ebnet den Weg zur Realisierung von IT-Dienstleistungsmärkten, welche messbare und abrechenbare Dienstleistungen unter Berücksichtigung verhandelbarer Dienstleistungsvereinbarungen (Service Level Agreements, kurz SLAs) versprechen. Ein Markt stellt in diesem Sinne nicht nur eine einfache Käufer-Verkäufer-Beziehung dar, sondern sollte eher als Bündelungspunkt einer komplexen Kette von Interessenvertretern gesehen werden, die eine hierarchische Integration des Wertes der Dienstleistung an jedem Knotenpunkt der Kette ermöglicht. Die Dienstleistungen verschiedener Partner werden in dieser Wertschöpfungskette unter Berücksichtigung von Erzeuger-Verbraucher-Verbindungen zusammengefasst und bilden so eine hierarchische Struktur zur Erreichung eines Mehrwerts. SLAs sind in diesem Sinne Verträge zwischen Service-Dienstleistern und Service-Konsumenten, welche eine per SLA festgelegte Qualität der Dienstleistungserbringung (Quality of Service, kurz QoS) erwarten, die auf verschiedenen Hierarchieebenen der Wertschöpfungskette vereinbart und somit gewährleistet wird. *Diese Dissertation befasst sich mit der Herausforderung, eine SLA-zentrierte Infrastruktur zu realisieren, die Dienstleistungsmärkte innerhalb des Utility Computings ermöglicht.*

SLAs spielen eine zentrale Rolle im gesamten Lebenszyklus der Gewährleistung und Aggregation (Bündelung) von Dienstleistungen. Aktivitäten wie Dienstleistungsverhandlungen und die spätere Auswahl bestimmter Dienstleistungen werden oft von einer hierarchischen Bündelung und der Validierung der Gültigkeit der Dienstleistungsvereinbarungen begleitet, welche wiederum SLAs als technologische Voraussetzung, respektive Grundlage, benötigen. *Die Arbeit zielt auf die Definition eines SLA-zentrierten Rahmenwerks ab, welches die anforderungsgetriebene Auswahl von Dienstleistungen, flexible SLA-Verhandlungsmechanismen, die hierarchische SLA-Bündelung und -Validierung und verwandte Themen wie etwa Privatsphäre, Vertrauen und Sicherheit behandelt und formalisiert. Zudem werden Prototypen für die praktische Umsetzung der Dienstleistungsauswahl und deren Validierung präsentiert.* Das formale Modell zur Implementierung der benutzergesteuerten Dienstleistungsauswahl bedient sich dabei des Branch-and-Bound-Ansatzes und heuristischer Algorithmen. Eine Erweiterung dieses formalen Modells betrifft die Verhandlung von SLAs für konfigurierbare Dienste unterschiedlicher Granularität mit dem Ziel, die Interessen und die damit verbundene Wertschöpfung der Verbraucher und Service-Dienstleister zu optimieren.

Die Möglichkeit der Bündelung von Dienstleistungen eröffnet neue Geschäftsmöglichkeiten in der sich entwickelnden wirtschaftlichen Landschaft im Bereich der IT-basierten Dienstleistungen. Ein SLA als Einheit einer Geschäftsbeziehung gesehen hilft dabei innovative Topologien für Business-Netzwerke zu etablieren. Ein Beispiel ist die Zusammensetzung der rechenleistungsbasierten und ressourcenorientierten Dienstleistungen zum Ziel

eine Dienstleistung größerer Granularität zu schaffen, welche die Möglichkeit eröffnet, Business-Modelle aufbauend auf der Bündelung von Dienstleistungen, der zusammengesetzten Zurverfügungstellung von Dienstleistungen und dem Dienstleistungswiederverkauf zu etablieren. Diese Arbeit führt in den Bereich der SLA-Choreographien und der hierarchischen Aggregation von SLAs ein und behandelt diese in Verbindung mit den zugrunde liegenden Dienstleistungschoreographien um letztendlich SLA-zentrierte Dienstleistungswertschöpfungsketten und Business-Netzwerke zu ermöglichen. Die SLA-Choreographie und -Aggregation stellt neue Herausforderungen in Bezug auf die Beschreibung, Verwaltung, Wartung und Validierung, lässt aber auch Aspekte wie Vertrauen (Trust), Privatsphäre und Sicherheit nicht außer Acht. Die Aggregations- und Validierungsmodelle für SLA-Choreographien präsentieren Konzepte wie etwa SLA-Sichten, um die Privatsphäre der Betroffenen zu schützen, ein hybrides Vertrauensmodell (Trust-Modell), um das Vertrauen zwischen Unternehmen und unbekannten Partnern zu fördern, und einen PKI- (Public-Key-Infrastruktur-) basierten Sicherheitsmechanismus gekoppelt mit einem regelbasierten Validierungssystem, um verteilte Abfragen über heterogene Systemgrenzen hinweg zu ermöglichen. Ein verteiltes, regelbasiertes, hierarchisches SLA-Validierungssystem wurde konstruiert, um die praktische Signifikanz zu demonstrieren.

Acknowledgements

First of all, I am grateful to God for bestowing me with abilities and opportunities to seek and spread knowledge and to reach this prominent milestone of my life where I am in a position to extend my gratitude to my friends and family for their support and motivation, which helped me accomplish my PhD requirements.

I am especially thankful to my advisor, Prof. Erich Schikuta, for endowing me with invaluable guidance and motivation throughout my PhD studies. Erich's vast experience as an evaluator of various EU projects was a treasured asset for me as I did not experience even the slightest difficulty in selecting up one of the best research topics amongst numerous technical jargons. In addition to being an excellent researcher, Erich has some of the finest human values to inspire his students and coworkers alike. This research work could not have been completed without him and I feel lucky to find him as my research supervisor and friend.

I also want to extend gratitude to my coworkers, Peter Beran, Juergen Mangler, Helmut Wanek, Nick Tahamtan, Martin Polashek, Sonja Kabicher, Simone Kriglstein, Konrad Stark, Axel Kittenberger, Andreas Woeber and Monika Hofer for their support which was available when needed throughout the PhD cycle. I also want to thank Heinz Stockinger, Prof. Stefanie Rinderle-Ma and Prof. Renate Motschnig for their guidance and motivation. It was a great pleasure to work together and I thank you for the many inspiring discussions.

A special thanks also goes to Kevin Kofler for his skilled mathematical knowledge and careful proofreading of our publications and my thesis. Kevin contributed in building the SLA oriented service selection model as part of his Bachelor's course work and as a result of some very exciting discussions; his sound mathematical knowledge helped me formulate many complex concepts throughout my thesis. I would also like to thank Semir Rahic for his contributions in preparing the simulation environment for the rule-based validation of hierarchical SLAs as part of his Bachelor's course work. I would also like to thank my friend Adi Abdurab who through his sound grasp of the English language helped me proofread this thesis including these acknowledgments.

I am thankful to Wajeeha Khalil and Altaf Ahmad Huqqani for their continuous motivation and generous support as friends and colleagues.

I would also like to thank Ivona Brandic for the immensely valuable discussions, her motivation and her offer for research collaboration. A special thanks to Prof. Harold Boley and Prof. Adrian Paschke for their unremitting support and direction without which the thesis´ rule-based systems could not be accomplished.

I am also thankful to my eminent fellow researchers Prof. Ramin Yahyapour, Prof Wolfgang Zielgler, Prof. Umar Rana and Prof. Amin Tjoa whom I had the pleasure and honor of meeting at various conferences and their appreciation of my work and their offered

support to further my scientific career was a great motivator for me.

I am especially thankful to Mr. Asher Samuel for discovering, highlighting and trusting my business insight and rousing mine with his wonderful personality to apply myself and my technical skills for the greater good of the people.

A very special thanks to my very dear friend Dr. Kashif Aslam and to his wife Dr. Amina for facilitating us directly from California with a variant of online medical consultancy whenever any of us were sick and for encouraging me and reassuring me when I was feeling down through the appreciatively long phone calls.

I can never pay enough gratitude to Azeem bhai who had been the source of my spiritual and intellectual guidance during my studies and for inducing me with a strong will that helped me find my way whenever I was about to lose focus.

I am especially thankful to my mother, my brother, my sisters and my all other relatives in Pakistan who prayed for my success and always cared for my comfort and sent me the ever appreciated good news from home.

Last, but not least, I would like to thank my wife, Asma, and my little sunshine Arfa for their abundant patience and understanding as I spent many nights and weekends away from them buried in my research.

It was a great time in Austria and Department of Knowledge and Business Engineering with all my colleagues and friends. Vienna is a wonderful city and after these four years has become my second home. I consider it my privilege to have studied in University of Vienna under the supervision of Erich amongst all my brilliant coworkers.

Irfan
Vienna, Austria
December 2010

Chapter 1.

Introduction

Not actions but direction is premier.

(Hazrat Fazal Shah Q.A)

1.1. Motivation

Resource virtualization along with the Service Oriented Infrastructure (SOI)lays groundwork for the realization of the notoin of service based Utility Computing. Utility computing is not a new concept. As the matter of fact the term was coined by John McCarthy during his address to MIT Centennial in 1961: *"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry"*.

As Gartner Hype Cycle [1] (please see Figure 1.1) denotes Cloud Computing as the most popular emerging technology of 2009, *Service Oriented Computing (SOC)* is shown as not being far from practically realized, growing very rapidly and driven by the promise that it will dramatically change the lives of individuals, organizations and society as much as the Internet and the Web have in the past decade. The gap projected on the hype cycle between them transforms into a multi-fold relationship when Cloud Computing, which is envisioned to provide a platform for future computing utilities, turns out to be built upon the notion of SOC, where it materializes itself in form of Software as a Service (SaaS), Platform as a Service (Paas) and Infrastructure as a Service (Iaas) utilities. Services are traded under formal contracts known as Service Level Agreements (SLA). The producer and the consumer of a service, with individual views of the same service reach a shared vision in the corresponding Service Level Agreement (SLA). SLA in Service Oriented Infrastructure (SOI), is an automatically processable contract between a service and its client; the client being a person, organization or another service. In eBusiness platforms such as Cloud Computing, SLA is essentially important for the service consumer as it compensates consumer's high dependency on the service provider.

Service Economy based on IT utilities, to prosper, requires IT-based Service Markets for stake holders to enable them to do business autonomically and autonomously helping them establish networks of business relationships. A market does not represent a simple buyer-seller relationship, rather it is the *culmination point of a value chain of stake-holders*

[1] http://www.gartner.com

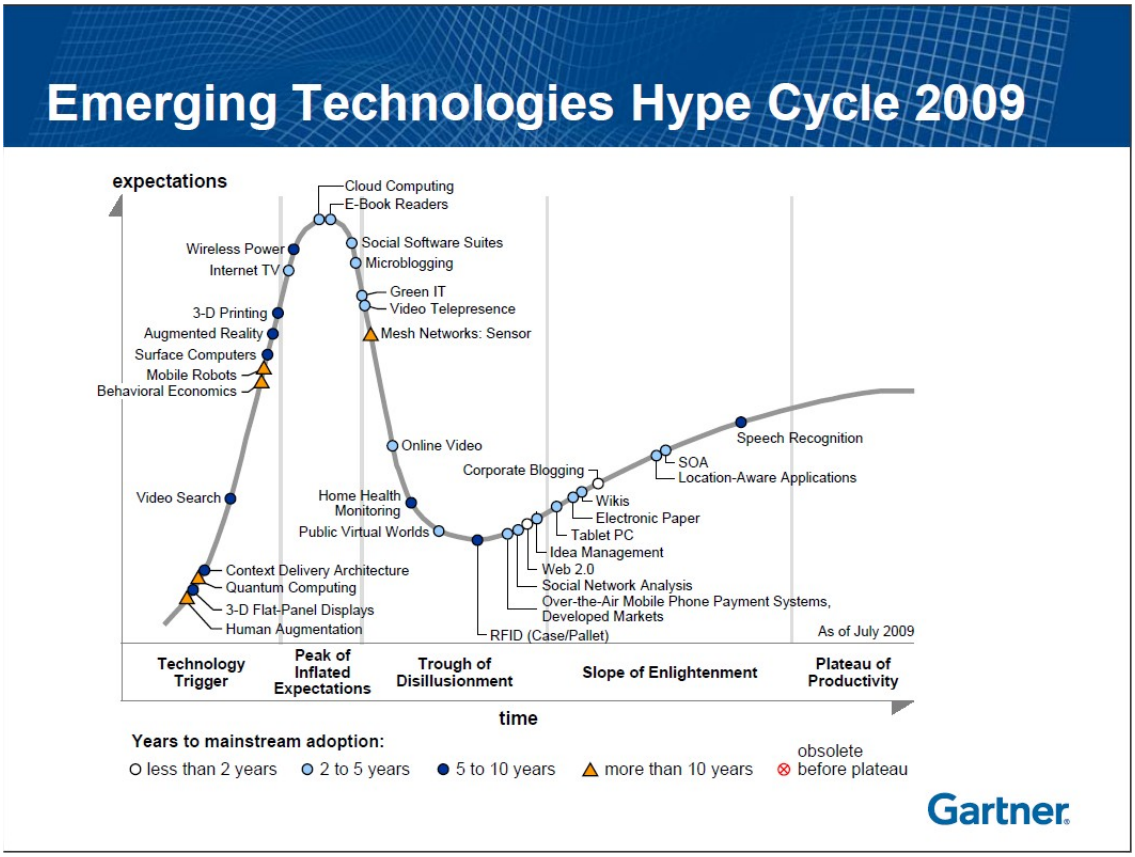


Figure 1.1.: Gartner Hype Cycle(2009) for Emerging Technologies

with a hierarchical integration of value along each point in the chain. As applicable in many evolving ICT infrastructures, there remains a gap between IT-based Service Economy as a concept and its application. The reason for which is the lack of enabling infrastructure. This gap in context with the IT-based Service Economy indicates Service Markets to be practically realized, which in turn requires an enabling infrastructure to support service value chains resulting from service composition scenarios.

In service value chains, services corresponding to different partners are aggregated in a producer-consumer manner resulting in hierarchical structures of added value. Service Level Agreements (SLAs) guarantee the expected quality of service (QoS) to different stakeholders at various levels in this hierarchy. In turn, this leads to a hierarchical structure of SLAs that may span across several Virtual Organizations (VOs) with no centralized authority. In this thesis, it is termed as *Hierarchical SLA Choreography* or simply SLA Choreography in accordance with the underlying Service Choreography. With the advent of Cloud computing and Internet of Services, there is a high potential for third party solution providers such as Composite Service Providers (CSP), aggregators or resellers to tie together services from different Clouds or external service providers to fulfill customer pay-per-use demand. *A cumulative contribution of such Composite Service Providers will*

emerge as service value chains.

Ideally these service aggregators in the service enriched eBusiness platforms, should facilitate a user to compose an application based on services of various granularity by gluing them together under binding contracts driven by user defined constraints. For example, the user states its requirements by sketching an abstract workflow and the best services should be selected and mapped on different activities of the workflow. Even in the ideal scenario, one does not expect an exact match of clients requirements but rather a selection of only the closest matching services. However, this thesis discusses how these service can still be fine tuned with client's requirements by going through a two-way negotiation phase between the client and the service provider. It is quite possible that service aggregators lineup to form chains of services resulting in service value chains in order to come up with the set of final composed services deliverable to the client.

A major challenge to enable IT-based Service Markets thus is to foster these hierarchical service composition scenarios and their underpinning business networks and supply chains. From business' point of view, the most important asset is the extraction of value from every node of such business networks in a transparent and secure manner. Service Oriented Infrastructure (SOI), in order to support these complex business interconnections leading to service value chains and the resulting business models, needs to cater enabling requirements like privacy, trust, security, transparency, autonomy, fair trade, open market, equal opportunities etc. NESSI (Networked European Software and Services Initiative, http://www.nessi-europe.com/), which is a consortium of over 300 ICT industrial partners has pointed out various possibilities for inter-organizational business models; Business Value Networks, Hierarchical Enterprises, Extended Enterprises, Dynamic Outsourcing, and Mergers to name a few. The concept of SLA Choreography becomes crucial in context to the process of business value generation.

The SLA Choreography may correspond to several value chains. As shown in figure 1.2, along these value chains, SLAs are aggregated in a bottom up fashion. In the research community to date, neither SLA Choreography nor its hierarchical aggregation has been taken into consideration as an enabling requirement for service value chains.

The layers in the SLA Choreograpy are also bound to the visibility of stake-holders, for example, the client has concerns only with the services immediately connected to it and can not see beyond. Despite these privacy concerns, as depicted in figure 1.2, a SLA is at the same time dependent upon the SLAs beneath it in the chain. The effects of this dependency are "bubbled up" through the upper layers. In this thesis, the concept of *SLA Views* has been introduced within the *SLA Choreography*, which enabling each business partner to have their own view comprising of its local SLA information. The holistic effect of these views emerges as the overall SLA Choreography. This hierarchical SLA Choreography and Aggregation poses new challenges regarding its description, and management.

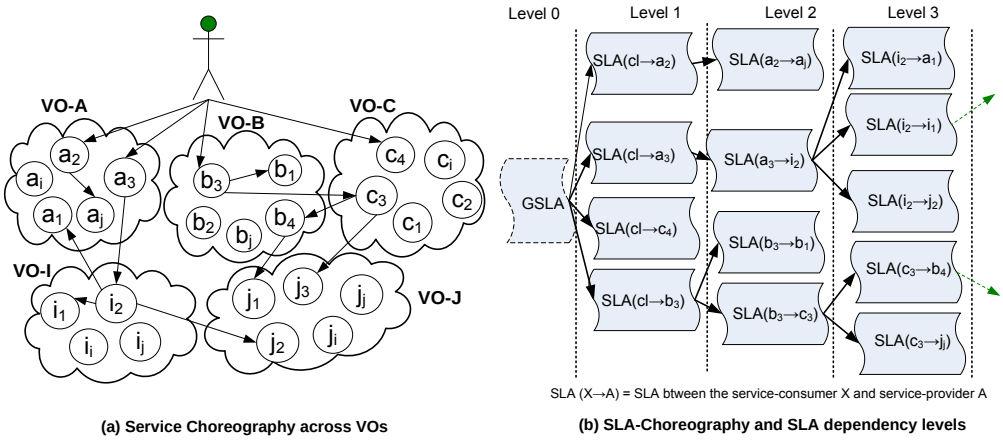


Figure 1.2.: Hierarchical Service Level Agreements and its Choreography

1.2. Hypothesis and Open Questions

The hypothesis of this research is built on the following set of reasonings:

1. IT-based Service Economy essentially depends on enabling service markets for computing utilities. A service market, which manifests itself as service value chains can be realized by identifying and addressing the issues of service value chains. SLAs, which glue services together, play a pivotal role during the entire life cycle of service value chains including the activities of selection, negotiation, hierarchical aggregation and hierarchical validation of the services.

2. Along with the service choreography and the service aggregation, notions of SLA Choreography and SLA aggregation must also be introduced and explained. On further research, related concepts such as aggregation functions and patterns can be easily traced in the spirit of generic application of these notions.

3. Business Issues and concerns such as privacy, security, trust can not be ignored while elaborating the above mentioned concepts. As a matter of fact, new concepts are highly dependent on a sensible adaptation of these business concerns.

4. The notion of hierarchical SLA aggregation demands a hierarchical validation of SLAs. This distributed validation process is highly entangled with the trust and security management across heterogeneous enterprises.

The fundamental open questions addressed in this thesis are:

- How to select and compose requirement-driven services and fine tune them with client's expectations by going through a negotiation phase that should culminate at binding SLAs ?

- How to aggregate and formally describe SLAs along with the composition of services?

- How to form SLA-centric hierarchical service choreography and the corresponding SLA Choreography?
- How to describe, manage and automate this SLA Choreography in formal manner?
- How to foster trust among different partners across an SLA Choreography to nurture sustainable business value?
- How to aggregate and validate SLAs in an incremental way within the SLA choreography?
- How business related issues such as privacy, automation, trust, security, and unbiased trade are related to sustainable value extraction and how can they be enabled by elaborating on the concept of SLA choreography?
- In case of SLA violations, how will the overall trust of the business network modified?

The research is driven by business requirements such as privacy, trust, security, assurances, dependability, value production, autonomy, fair trade, open market, equal opportunities etc. as well as technical issues like automation, distributed aggregation and validation, third party trust management, security, distributed business intelligence, and cross-VO inter-operation etc.

These challenges have been met by defining, designing and evaluating the formal foundations of an enabling infrastructure in order to cultivate business value along SLA Choreography and its underlying service chains spanning across multiple heterogeneous virtual organizations. These solution integrate in form of an enabling framework for SLA centric Utility Computing, which is the overall contribution of this thesis.

1.3. Vision

This research pursues the vision of *business enabled Internet of services*, which opens doors for absolutely new business processes for *consumers and producers*. So in the near future, it will be common practice to sell IT resources as service and not as goods. *For example, "Writing a letter" can be as simple as using a telephone: Forget about purchasing software and hardware! All we need is a simple interface to the services on the Internet, both the wordprocessor functionality and the necessary physical resources (processor cycles and storage space); and everything is paid transparently via our telephone bill.*

As businesses are hard pressed to respond to market changes and rapidly adapt to evolving business strategies, new business models are becoming an increasingly critical source of their competitive advantage. This research work will contribute in enabling Service Markets for the IT-based Service Economy by helping business scenarios involving composite services to materialize and emerge with a guaranteed level of service in response to on-demand service orders. The formalized description of SLA choreography and aggregation models will help the research community to better utilize the role of SLA in IT-based

Service Economy and realize new business scenarios in this regard. It will also hold significant value for upcoming companies that will establish business models on novel SOC based infrastructures. This research will also help to fill in the gap between theoretically sound business concepts and nonexisting enabling technologies. This conflicting situation is a hinderance for the practice of new business ventures in the evolving IT landscape. The output of this research will also be valuable to IT-based Service Economy in numerous ways. This contribution will not only hold for large scale enterprises (LSE) that normally operate their own data centers but also for *small and medium enterprises* (SMEs) and startup businesses that in general cannot afford their own IT department and therefore use application provisioning and hosting services. By enabling business models involving third party stake holders such as aggregators and resellers, this research work is expected to play a key role in promoting business movements at the micro-economic level of the IT-based service industry. One of the contributions is in terms of guaranteed Return of Investment (ROI) for stake holders by guaranteeing the desired level of service hence helping them shift from the usual Capital Expenditure (CAPEX) model to the Operational Expenditure (OPEX) model. Business Process Management (BPM) will also directly benefit from this research. The pay-per-use service models will not only help in cost reduction but also in various services after being composed together with the help of orchestration technology will become available again as more capable composite services with a guaranteed level of service. This work will directly contribute towards build the foundations of eBusiness in the Service Oriented Infrastructure based on social values such as fair trade and free market. Therefore this research output has the potential to give new impetus to the IT market by boosting growth and competitiveness in many industrial and business sectors.

1.4. Thesis Structure and Contribution

Figure 1.3 provides a layout of the thesis' chapters, their internal relationships and their mapping to the research topics. The vertical arrows highlight the dependencies between the chapters: the lower being dependent on the above. Depicted beside the chapters is a high level layered architecture of SLA based SOI. Starting from the SLA based service selection and negotiation it reaches upto the SLA based fault tolerance and renegotiation. It must be noticed that it is only a high level layered architecture and can be further elaborated by adding new layers and new topics. The purpose of this abstract architecture is to highlight the contribution of thesis chapters.

Figure 1.3 depicts the thesis structure as well as the research contribution. The dependencies between different chapters as depicted by vertical arrows are transitive, which means that Chapter 6 is not only dependent on Chapters 4 and 5 but also on Chapter 2. However these dependencies do not indicate any compulsory reading instructions, but only a guide for the reader about the work distribution.

The overall contribution of the thesis is a framework for enabling SLA-centric service-based Utility Computing. On the left side of Figure 1.3, various activities of the framework

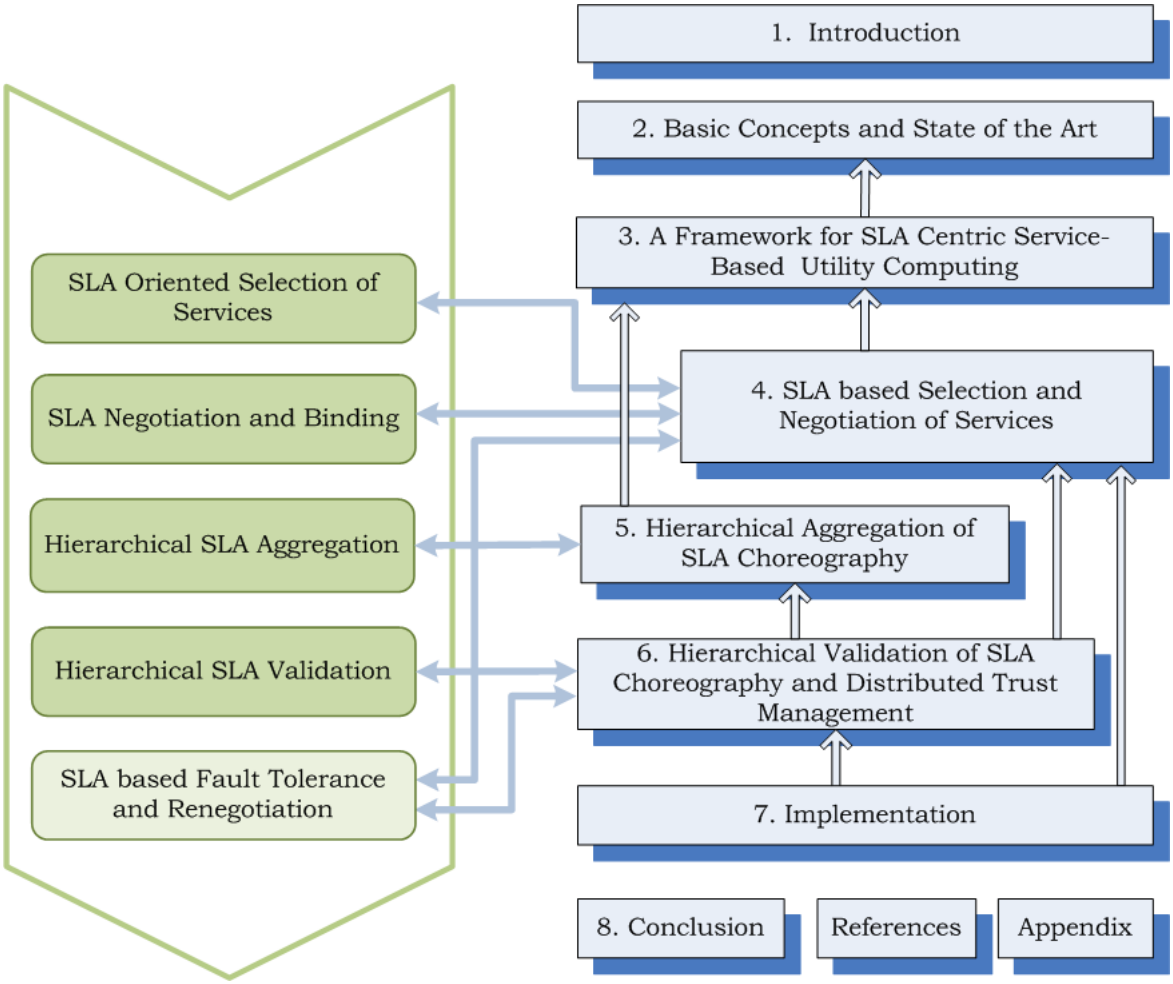


Figure 1.3.: Organization of the Thesis

shown as different phases within a process container are mapped via horizontal arrows to different chapters of the thesis where they have been discussed. The first four dark colored phases represent the essential contribution of the proposed framework whereas the light colored phase at the bottom indicates a partly covered topic.

The thesis is organized into chapters as follows:

Chapter 2 Basic Concepts and State of The Art:

There are numerous notions that require comprehension in order to completely understand the role of SLAs at various stages during the life cycle of service provision. This chapter explains these interrelated concepts and presents a survey of the related work. Key questions answered in this chapter include:

- What are the concepts related to the overall role of SLAs in connection with the service provision in Utility Computing based infrastructures?

- What state of the art research has been carried out regarding SLAs and their interrelated issues? What has been achieved and what is still missing?
- What can be learned from the trend of the research going on in terms of the major SLA focused projects?

The related work is organized around representation, formalization, negotiation, aggregation, and validation of SLAS, and their closely related issues such as trust and security, workflow interaction etc.

Chapter 3 A Framework for SLA Centric Service-Based Utility Computing:

This chapter presents the framework discussing the holistic interaction of various solutions components elaborated throughout the thesis. The key question being answered in this chapter include:

- How can an SLA centered service oriented Utility Computing framework be visualized and designed?
- What is the significance of having such a framework?
- How can we represent this framework as a process diagram in terms of a systematic interaction of various activities, their related milestones and the enabling tools?

This chapter elaborates the interaction among various components of the framework and narrates a motivational scenario that will be used as a running example throughout the thesis to highlight different phases of the SLA life cycle as a pivotal technology to enable Utility Computing.

Chapter 4 SLA-Based Selection and Negotiation of Services:

This chapter discusses user-directed and SLA oriented selection and negotiation of services. Requirement-driven SLA oriented service selection, its composition and its negotiation brings in third parties to business models thus empowering players at the micro-economy level. The open questions answered in this chapter include:

- How can an optimized QoS based selection can be made for a set of services while keeping the service-level and the set-level QoS constraints intact?
- How can the selection algorithms be adapted to changing service parameters or updated user requirements?
- How can a flexible SLA negotiation mechanism be designed for configurable services?

Chapter 5 Hierarchical SLA Aggregation:

This chapter focuses on the hierarchical composition of SLAs. In this research work, it has been shown that we need to have a formal model for a multi-level or hierarchical SLA aggregation to couple with service value chains and value networks. The open questions include:

- What kind of SLA structures emerge in connection with service value chains and service choreographies?
- How can a step-by-step aggregation of SLAs corresponding to a service choreography be formalized and realized?
- What are the basic aggregation patterns for hierarchical SLA aggregation?
- How can the privacy of stakeholders be preserved within SLA structures corresponding to the service value chains and service choreographies?

The notions such as SLA Choreography to describe hierarchical SLA structures, SLA Views to preserve the privacy of a stake holder within SLA Choreography and SLA aggregation patterns have been introduced and formalized in this chapter.

Chapter 6 Hierarchical SLA Validation and Trust Management:

SLA Choreography and its aggregation requires validation. In order to describe this hierarchical SLA validation, one needs to take into account several interrelated issues such as trust, privacy and security. This chapter presents a rule-based validation mechanism that closely couples with a hybrid trust management system based on PKI and reputation to help navigate validation queries across heterogeneous organizational boundaries. The third party trust manager is employed to foster trust, ensure fair trade and to enable the transfer of business recommendations through reputation transfer mechanism. The open questions catered in this chapter include:

- What are the enabling requirements for the hierarchical validation of SLA Choreographies?
- How can aggregated SLAs be represented as distributed set of rules and what kind of distributed query processing mechanism is required for hierarchical validation across heterogeneous Virtual Enterprises?
- What is the role of trust management system in the hierarchical SLA validation?
- What kind of rule based systems are necessary to realize the hierarchical SLA validation model?

This chapter presents a rule based validation model based on Rule Responder and Rule Based SLAs (RBSLA) architectures and elaborates the distributed query processing mechanism across heterogeneous boundaries of collaborating stakeholders.

Chapter 7 Implementation:

This chapter provides a detailed account of implementation of two solution components of the proposed framework:

- The user-directed SLA oriented service selection
- The hierarchical SLA validation system for the running example described in this thesis

The chapter describes the simulation environment, algorithms and data handling for both the solution components. Finally, the results are evaluated and explained.

1.5. Disclosure and Acknowledgements

This work summarizes PhD research contributions funded through Pakistan Scholarship for Science, Technology and Engineering granted by Higher Education Commission (HEC) Pakistan.

Chapter 4 is mainly derived from the following publications:

- A Parallel Branch and Bound Algorithm for Workflow QoS Optimization , Kevin Kofler, Irfan Ul Haq, Erich Schikuta The 38th International Conference on Parallel Processing (ICPP2009) - Vienna, Austria [80].
- A Two-Phase, Heuristic Algorithm for Workflow QoS Optimization with Dynamic User Requirements, Kevin Kofler, Irfan Ul Haq, Erich Schikuta Europar2010 - Ischia, Italy [81].
- Dynamic Service Configurations for SLA Negotiation, Irfan Ul Haq, Kevin Kofler, Erich Schikuta Europar 2010, Workshop CoreGrid 2010 - Ischia, Italy [124].
- Using Blackboard System to Automate and Optimize Workflow Orchestrations, Irfan Ul Haq, Erich Schikuta, Kevin Kofler The 5th IEEE Conference on Emerging Technologies (ICET 2009) - Islamabad, Pakistan [129].

Chapter 5 is mainly derived from the following publications:

- Aggregating hierarchical Service Level Agreements in Business Value Networks, Irfan Ul Haq, Altaf Huqqani, Erich Schikuta Business Process Management Conference (BPM2009) - Ulm, Germany [66].
- A Conceptual Model for Aggregation and Validation of SLAs in Business Value Networks, Irfan Ul Haq, Huqqani Altaf Ahmed, Erich Schikuta The 3rd International Conference on Adaptive Business Information Systems (ABIS 2009) - Leipzig, Germany [123].
- Aggregation Patterns for Service Level Agreements, Irfan Ul Haq, Erich Schikuta, International Conference on Frontiers of Information Technology 2010, Islamabad, Pakistan [127].

Chapter 6 is mainly derived from some of the already listed as well as the following publications:

- Rule-Based Validation of SLA Choreographies, Irfan Ul Haq, Paschke Adrian, Erich Schikuta, Boley Harold to appear in the Journal of Super Computing 2010 [130].
- Rule-Based Workflow Validation of Hierarchical Service Level Agreements, Irfan Ul Haq, Paschke Adrian, Boley Harold, Erich Schikuta 4th International Workshop on Workflow Management (ICWM2009) in conjunction with the The 4th International Conference on Grid and Pervasive Computing (GPC 2009) - Geneva, Switzerland [126].
- Distributed Trust Management for Validating SLA Choreographies, Irfan Ul Haq, Rehab Alnemr, Adrian Paschke, Erich Schikuta, Harold Boley, Christoph Meinl SLAs in Grids Workshop as part of Grid 2009 Conference - Banff, Canada [65].

Chapter 2.

Basic Concepts and State of The Art

> A man should look for what is, and not for what he thinks should be.
>
> *(Albert Einstein)*

Contemporary computing paradigms such as Cloud Computing, Service Oriented Computing, Commodity Computing and Utility Computing all pursue the same industrial goal, which is: to enable consumers to access and utilize the shared resources on demand as consumable services. This chapter explains how this vision strongly depends on the role of SLA and its enabling technologies. The overall contribution of the chapter consists of:

- An introduction to SLA-Centric Utility Computing and its sister technologies.
- A comprehensive analysis of the role of SLAs in Utility Computing.
- A survey of the state of the art emphasizing the role of SLAs at various stages during service provision in Utility Computing.
- An introduction to important projects focused on SLA based Utility Computing.

2.1. Service Oriented Architecture (SOA)

In the forthcoming era of Service Based Utility Computing, services will be the basic blocks of complex software systems. Services will be like atoms joining together and forming composite structures of varying granularity. Similar to atoms of molecules that have different levels of affinity for each other, services have their own chemistry. This helps to predict, which services can combine together. Their binding affinity is determined by the mapping of their Quality of Service (QoS) attributes to consumer requirements. Services can search each other on the basis of these attributes. Just like atoms, services with common interests may make bond through a Service Level Agreement (SLA). SLAs are machine-processable electronic contracts established among two or more parties; these parties being services and their customers. These contract contain terms of business and rules of cooperation.

A service is defined as [29] a function that is well-defined, self-contained, and does not depend on the context or state of other services. Service Oriented Architecture (SOA) speaks of a collection of services, which communicate with each other, e.g., simple data

passing or two or more services coordinating an activity [29]. The SOA services follow the pattern of publish, find and use. The services are published through registration so that other services or users can discover them. After the discovery of a service, that service is contacted and then can be used. *Service Oriented Computing (SOC)* is rapidly gaining popularity with an objective to change the life of individuals organizations and society in a similar way as the Internet and the Web have done in the past decade. The SOC pledges the revolution of the Internet by a novel and advanced support for collaboration. In SOC, *Virtualization* is the process of creating virtual version of IT resources such as of Operating System, hardware, memory, storage, etc. The process of virtualization yields virtual resources, which build the basis of SOA, for instance in storage virtualization, multiple storage devices are virtualized to appear as one source of the offered storage services. As the consumers' demand grows, new storage devices can be added in that single virtual resort. The related technologies like Grid and Cloud Computing provide homogeneous access to virtual resources without revealing the heterogeneous nature of the underlying real infrastructure. In SOA, *XaaS* refers to X as a Service architecture where X can be interpreted as anything, everything or all. XaaS is based on the concept of virtualization. The most popular of XaaS type of service are grouped in the SPI model [**?**] , which categorizes three types of services i,e., Software as a Service (Saas), Platform as a Service (Paas) and Infrastructure as a Service (Iaas). Other popular XaaS types are Hardware as a Service (HaaS), Communication as a Service (CaaS) and Network as a Service (NaaS) etc.

Service orchestration is the process of composing several interacting services participating for example in a business process. The resultant orchestration can become available again for further composition. *Service choreographies* have been described [46] to capture the complex conversations between the orchestrations from a global perspective, i.e., internal service invocations within one partner are hidden. Service orchestrations are populated within one administrative domain whereas service choreographies can span across multiple administrative domains.

The concept of *value chain* was coined by Michael Porter in 1985 [112]. In IT-Based Service Economy, a *service value chain* is the chain of value adding services. These services can be activities of a workflow, a Business Process Model, a slice of service choreography or service orchestration. The role of service value chains is extremely important in enabling service markets as they realize the underlying complex supply chains and numerous value adding activities culminating into the finished product. In this thesis the role of service value chains has been considered with reference to service markets and service choreographies. *Autonomic Computing* speaks of the autonomic self-* services. These services are self-managing, self-protecting, self-composing, self-optimizing and self-healing etc. Achieving this autonomy is part of the bigger effort: to take the major shift from machine-centered to human-centered computing. This means that devices should be able to understand and work for humans. A major requirement to accomplish this is to enable devices to communicate with each other seamlessly. This autonomy finds its crucial test in heterogeneous environments: where it needs to perform *interoperability*. Interoperability

also allows Virtual Organizations to cater adaptable business workflows and service orchestrations. In Virtual Organizations (VO) or Virtual Enterprizes resources may be dispersed geographically but function as a coherent unit.

2.2. Service Based Utility Computing

Utility computing is not a new concept. The term was first coined by John McCarthy during his address to MIT Centennial in 1961:

"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry". After fifty years, the *computer utility* is being seen as the basis of a new and important industry.

Almost fifty years after John McCarthy's dream of Utility of Computing, Rajkumar Buyya, in his recent paper [36], "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility", carries on his vision with these words:

"Computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony. In such a model, users access services based on their requirements without regard to where the services are hosted or how they are delivered. Several computing paradigms have promised to deliver this utility computing vision and these include cluster computing, Grid computing, and more recently Cloud computing".

The term Utility Computing is often used synonymously with "On-demand Computing", which talks about the pay-per-use services. Each of them is treated as a business model based on "resource metering" usage, or a "pay as you go" approach. In the utility or on-demand business model the utilities or services are not charged at a flat rate but are metered, which means the payment is calculated based on usage amount. With the advent of Service Oriented Architecture (SOA), the vision of Utility Computing can be realized in form of metered-services hence this thesis takes the liberty to use the term, *"Service-Based Utility Computing"* to elaborate this vision.

Cloud computing can be defined as the convergence and evolution of several concepts from virtualization, distributed application design, Grid and enterprize IT management to enable a more flexible approach for deploying and scaling applications based on Service Oriented Infrastructure (SOI) driven by the business motif of on-demand service provision. Several Cloud Computing architectures and models have been proposed [36, 27, 44, 33].

Cloud Computing materializes SOA and utility model in form of Software as a Service (SaaS), Platform as a Service (Paas) and Infrastructure as a Service (Iaas) categories of pay-per-use utilities contracted under SLAs. In Cloud Computing, service provision is done through SLAs, which is of prime importance for the service consumer as it compensates consumer's high dependency on the service provider. For service provider, SLA provides legal assurances. No matter how abstract the cloud infrastructure [36] may be, the role of

SLA must be considered in the process of service provisioning.

Utility Computing is a generic business focussed approach to provide computing as a utility. Cloud Computing pursues the same business objective as that of the Utility Computing but also addresses the technical challenges related to the infrastructure design and deployment of such a utility based business model. Cloud Computing focusses on SPI type of XaaS service model in which services built on virtualized resources are self-healing, SLA-driven, multi-tenancy and linearly scalable. SLAs play a pivotal role in accessing and utilizing services offered in a Cloud infrastructure.

2.3. Service Level Agreements (SLA)

Service Level Agreement is a formal, legal contract between a service provider and a customer that specifies, in quantifiable terms, what service level guarantees the service provider will deliver, and it defines the consequences (penalties) if the service provider fails to follow through with said commitments. SLAs were originally used by Information Technology (IT) organizations and adopted by telecommunications providers [90] to manage the quality of service(QoS) expectations of their products. SLAs are a means of QoS assurance to service consumers in SOA and have widely been adopted in Grid and Cloud Computing paradigms.

The Tele-Management Forum [18] defines an SLA as "A Service Level Agreement (SLA) is a formal negotiated agreement between two parties. It is a contract that exists between the Service Provider (SP) and the Customer. It is designed to create a common understanding about service quality, priorities, responsibilities, etc. SLAs can cover many aspects of the relationship between the Customer and the SP, such as performance of services, customer care, billing, service provisioning, etc. However, although a SLA can cover such aspects, agreement on the level of service is the primary purpose of a SLA". Today, SLAs between contracting parties are used in all areas of IT-services (e.g. hosting and communication services, help desks and problem solution). In addition, different organizations have different definitions for crucial IT parameters such as Availability, Throughput, Downtime, Bandwidth, Response Time, etc. [58]. Service Level Agreement (SLA) has been defined by several authors with almost the same underlying meaning: Jin et al. [75] explain the role of SLA as: "A Service Level Agreement (SLA) between a service provider and its customers will assure customers that they can get the service they pay for and will obligate the service provider to achieve its service promises. Failing to meet SLAs could result in serious financial consequences for a provider. Hence, service providers are interested in gaining a good understanding of the relationship between what they can promise in an SLA and what their IT infrastructure is capable of delivering. Similarly, consumers are interested in understanding the impact of the SLAs they sign on their own productivity".

For each domain, an appropriate set of metrics should be defined,because there exist no general set of metrics fitting to all domains. A set of metrics should be small enough to easily control and big enough to cover all essential application areas. So, metrics should be easy to measure and to collect in order to allow effective SLA enforcement and they

should be within the service providers' control [118].

IBM researchers define [84] SLA as: "An SLA is part of the contract between the service provider and its consumers. It describes the provider's commitments and specifies penalties if those commitments are not met". Tlhong and Reeve [121] extend this view as: "Service Level Agreements (SLAs) have traditionally been considered as a legal binding between a service provider and a customer. However, the advent of Service Oriented Architectures(SOA) and service based business models has seen the IT industry move away from considering SLAs only as a legal document but instead as means of enforcing and managing user requirements and expectations".

Masche and Mckee [92] while elaborating the importance of SLAs in B2B systems describe: "the confidence of the consumer is established through a contract with the provider of the service. Such contracts, commonly known as Service Level Agreements (SLA), set out the quality of service (QoS) and the terms and conditions that a consumer and provider of a service have agreed . The SLA also specifies how the service is priced and the compensation terms if the SLA is violated. In a service oriented computing landscape, every service needs to have a SLA".

2.3.1. SLA Specification

A typical SLA is composed of several terms or components [101, 79, 75], the most common of them are explained below.

- Purpose describes the background of the agreement.
- Parties describes all concerned parties.
- Validity Period defines the current period of the SLA.
- Scope specifies the services covered in the agreement.
- Restrictions specifies the valid restrictions.
- SLA Parameters have names, types and units and describe Quality of Service (QoS) properties of service object.
- Every SLA parameter refers to one (composite) Metric, which aggregates one or more other metrics. A metric either defines a Function or it has a Measurement Directive. A Function represents a measurement algorithm that specifies how a composite metric is computed i.e. formulas of arbitrary length containing mean, median, sum, etc. A Measurement Directive specifies how an individual metric is retrieved from the source. (i.e. uniform resource identifier, protocol message, etc).
- Service-level Objectives describe the agreed service level, which should be achieved.
- Service-level Indicators are measurable parameters, they are the basis for the SLOs.

- Penalties describes the consequences in the case of breaking the SLOs (i.e. due to an undervalued availability the service provider grants a discount).
- Exclusions defines the services, which are not included in the SLA.
- Administration describes the processes to control and measure the SLOs created in the SLA.

2.3.2. SLA Negotiation and Renegotiation

A SLA normally can ne offered to either of the participants initially as a template, which needs to go through a negotiation process before being legalized into binding a SLA. The negotiating process uses a negotiation protocol defined as a sequence of interactions between the involved parties to reach a binding SLA. An SLA negotiation process can result in the acceptance or rejection of the offered SLA by either of the parties. Renegotiation is a process similar to negotiation with the only difference that it is carried out to redefine the terms of an already established SLA, therefore in case of failure of renegotiation the existing SLA persists. The Service Negotiation and Acquisition Protocol (SNAP) [43] is one of the foremost negotiation protocols for Grid-based services. It distinguishes three kinds of resource-independent service level agreements (SLAs), formalizing agreements to deliver capability, perform activities, and bind activities and supports reliable management of remote SLAs. SNAP can be deployed within the context of the Globus Toolkit. SLA@SOI [22], which is an ongoing European project, also aims at the SLA negotiation process and protocol. WS-Agreement which has lately become a de facto standard for Service Level Agreement description, is in the process of finalizing SLA negotiation and renegotiation protocols [145]. Yan et al. [137] present an agent-oriented SLA negotiation protocol based on a formal model and coordinated with end-to-end quality of service requirements for dynamic service composition. Parkin et al. [102] put forth their multi-round re-negotiation protocol for SLAs keeping in view network failures causing lost, delayed, duplicated or reordered messages. However, none of these approaches identifies the significance of computing flexible SLA configurations to smoothen up the negotiation process.

2.3.3. SLA Formalization

Aiello et al [24] present a formal description of SLA. Their approach is based on WS-Agreement. They extend the WS-Agreement standard by introducing a new category of terms called Negotiation Terms. They built an automaton representation of SLA states to describe the negotiation process. Their formal model is too vague and they do not explain how this model will describe the sub-entities in WS-Agreement. Unger et al [131] present a rigorous formal model for SLA aggregation. They follow BPEL and WS-Policy whereas our formal model adheres to WS-Agreement standard. Masche et al. [92] propose that the SLA should contain terms that only relate to business level objectives (BLO) whereas the deployment and management details of a service are hidden by virtualization in the provider's domain and therefore should not be expressed in the SLA.

2.3.4. SLA Aggregation

Service Level Agreement is a contract between a service and its client; the client being a person or yet another service. WSLA [136], WS-Agreement[56] and SLANG [83] are some popular languages of Service Level Agreement. Service composition directly implies the SLA composition. However a little research has been carried out towards dynamic SLA composition [31, 62, 131]. The research area corresponding to the management of such aggregated SLAs is still wide open. Blake and Cummings [31] have defined three aspects of SLAs which are Compliance, Sustainability and Resiliency. Compliance means suitability i.e. the consumer receives what is expected. Sustainability is the ability to maintain the underlying services in a timely fashion. Resiliency directly corresponds to the maintenance of services to ensure their performance over an extended period of time. The authors then subdivide these three categories into six aspects of SLA, namely, problem resolution, renegotiation, cost, uptime, service rate, maintenance to construct a formal model, and an algorithm of SLA aggregation. The six aspects make their approach rather specific. Moreover they assume services to exist only at one level. The research area corresponding to the management of such aggregated SLAs is still wide open. Ganna Frankova [62] has highlighted the importance of this issue but she has just described her vision instead of any concrete model. Unger et al's work [131] is directly relevant to our focus of research. They focus on aggregation of SLAs in context of Business Process Outsourcing (BPO). They synchronize their work with Business Process Execution Language (BPEL) and WS-Policy. Their model is based on SLO aggregation of SLAs. One of the limitation of their approach is that they take into account services related to one process in one enterprize because they focus on BPO. Our approach describes cross-VO SLA aggregation and strictly adheres to WS-Agreement.

2.3.5. Rule-Based SLAs

Adrian Paschke and Martin Bichler have done an extensive work on the Rule-Based SLAs (RBSLA) project [107]. RBSLA transforms SLAs into logical rules to automate their management and monitoring. They discuss knowledge representation of SLAs with complex business rules and policies. RBSLA [108, 107] uses a combination of Horn Logic, Deontic Logic and ECA (Event-Condition-Action) rules. RBSLA also covers many related areas such as the breach management, authorization control, conflict detection and resolution, service billing, reporting, and other contract enforcements. RBSLA employs query driven, backward reasoning for SLA management. RBSLA is a useful tool to transform text based SLAs. The approach presented in this thesis adheres to the WS-Agreement standard which is a structured document thus making the challenge of automation more convenient. Oldham et al [100] have extended WS-Agreement by building a rule based ontology on the WS-Agreement. Their SWAPS schema [100] transforms constructs from the Guarantee terms into predicate based markup language. They admit that their schema is limited to a specific domain. This research work proposes a distributed validation model for an SLA orchestration applicable in Business Value Networks.

2.4. SLA Languages and Their Implementations

SLAs are written in various XML based [119] languages. Following is an analysis addressing the characteristics, evolving features and shortcomings of some of these languages.

2.4.1. SLAng

SLAng [71] [83] has been developed in University College London keeping in view the requirements of electronic business. SLAng meets [83] multiple objectives: as a format for negotiation of QOS properties; as a means to capture these properties unambiguously and as a language for automated reasoning.Based on a hierarchical model [83] SLAng defines two types of SLAs: Horizontal SLAs between peers of the model i.e. same layers and Vertical SLAs between the subordinated pairs.

Vertical SLAs include communications SLA (between network elements and host OS), hosting SLAs (between host OS and application server), persistence SLAs (between host OS and database) and application SLAs (between web services and applications servers). Horizontal SLAs include networking SLAs (between network elements), container SLAs (between application servers) and service SLAs (between web services). SLAng aims to establish cross-organizational relationships. Lamanna et al [83] mention B2B and B2C type of relationships.

There are seven different types of SLAs out of which four are of the vertical type and three horizontal type. SLAng discusses the role of SLAs from service management perspective. It focuses on performance metrics and automation of system management.

SLAng has been modelled in UML. SLA metrics is a key concept of SLAng. The exact metrics depend on the domain of SLA. For the application service provider (ASP) domain, metrics are categorized to four QoS characteristic groups: service backup, service monitoring, client performance and operational QoS characteristics. Nurmela [99] highlights various problems of SLAng such as: it does not focus on a runtime selection or negotiation behavior, rather setting design time validation as the main goal. This also excludes it from having an attachment mechanism to WSDL: QoS is modeled as part of the application in web services consumer and producer application logic. SLAng constraints that define the service level objectives (SLOs) are formally defined using Object Constraint Language. Currently available actual formal definitions limit themselves to defining ASP reference model and behavioral model.

SLAng contains no support for breach and bonus management and billing. Likewise reuse of SLAs is not within SLAng scope. Also, there is no dependency expression between different types of SLA metrics, only between different types of participants. In terms of eContracting, SLAng is seen as the main mechanism to extend BPEL all the way to eContracting requirements. However, given that the language has to be extended to other domains beyond ASP and lacks breach and bonus management support, this seems problematic.

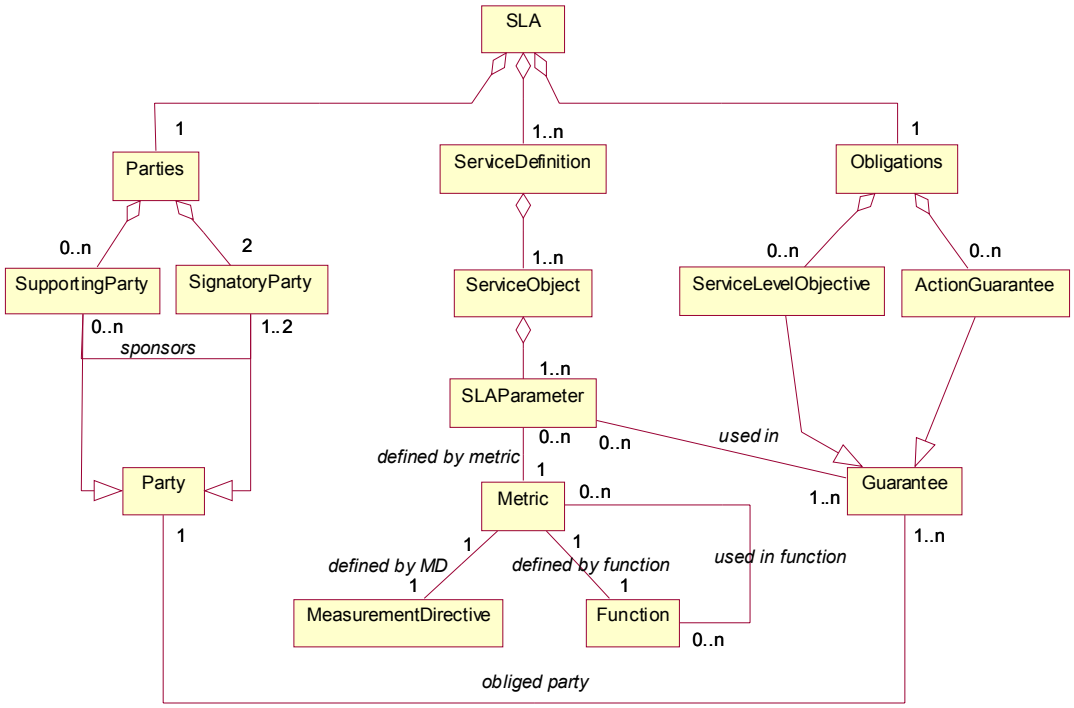


Figure 2.1.: Overview of WSLA structure

2.4.2. WSLA

Web Services Level Agreement (WSLA) [136] is a framework that has been developed by IBM during 2000-2003. The framework is capable of measuring and monitoring promised quality of services and can report violations. This framework has been planned to execute with the IBM Emerging Technologies Tool Kit (ETTK) which enables automated management [45] and compliance monitoring. WSLA contracts attach to web services by pointing to the WSDL description of the involved services. The structure of a WSLA based agreement comprises of three sections:

1. Parties which can be signatory or a supporting party. There can be third parties. Third parties include measurement (i.e. monitoring) providers, condition evaluators and management providers (i.e. breach management handlers).

2. Service Descriptions that contain characteristics of a service and its observable parameters.

3. Obligations are Service Level Obligations (SLO), which are in fact guarantees and constraints imposed on the SLA parameters.

Figure 2.1 presents an overview of the WSLA structure. The SLA parameters support hierarchies. Every SLA parameter refers to one metric which may aggregate one or more

other metrics. Composite metrics can be either directly mapped or aggregated to SLA parameters. A composite metric is defined by a function that is in fact - an algorithm, formula or statistical operator - specifying how the composite metric has been computed. Billing and service usage reports are generated on the basis of accumulated service usage and breach reports. Billing is defined in the agreement. Contract documents are represented as contract objects. WSLA also offers SLA templates. A template is [45] a WSLA document that contains fields to be filled in during the subscription process. WSLA templates are used to describe service offering through the negotiation process. WSLA contracts contain the SLA parameters and SLOs formed based on the WSLA template offered to the consumer. Seidel et al [72] report that the WSLA agreement management life cycle has five stages: 1. Establishment of an agreement after negotiation phase 2. Distribution of the SLA document after proper validation 3. Measuring the SLA parameters and comparing them against guarantees and reporting the results 4. Breach management in case of SLO violations; as part of Corrective Management Actions. 5. Termination of SLA after expiration date or with consensus of concerned parties.

WSLA defines means to express actions based on breaches, yet it does not provide information on meaning of any of the third party functions regarding monitoring, evaluation and breach management. Composition of contracts is also possible that enables multi-party SLA. This also means a contract can be split into multiple sub-contracts. Although the dependencies through aggregation of metrics can be represented but the dependencies among e.g. SLA parameters cannot be expressed. WSLA enhanced process designs seems problematic even based on the basic language specification. [72] further adds that a comprehensive support infrastructure is required to provide suitable support for applications that wish to utilize WSLA. WSLA has been utilized in various projects. TrustCoM [19] was part of EU 6th Framework Program (Networked Business and Government). It uses WSLA for negotiation, monitoring and SLA breach detection. Web Services Management Network (WSMN) [57] was a project of HP. The SLA engine of WSMN contains multiple subsystems to support functionality of WSLA. Furthermore it also implements SLO validation and evaluation management.

2.4.3. WS-Agreement

The Web Services Agreement Specification [56] from the Open Grid Forum (OGF) [12] describes a Web Services protocol for establishing agreement between two parties, such as between a service provider and consumer, using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties. WS-Agreement defines a language and a protocol to represent the services of providers, create agreements based on offers and monitor agreement compliance at runtime. The agreement can be dynamically established and dynamically managed. Figure 2.2 presents the basic structure of a WS-Agreement based SLA.

The section after the (optional) name is the context, which contains the meta-data for the entire agreement which includes the parties, duration of an agreement, template name etc. The section Terms contains two types of terms. The service terms provide information

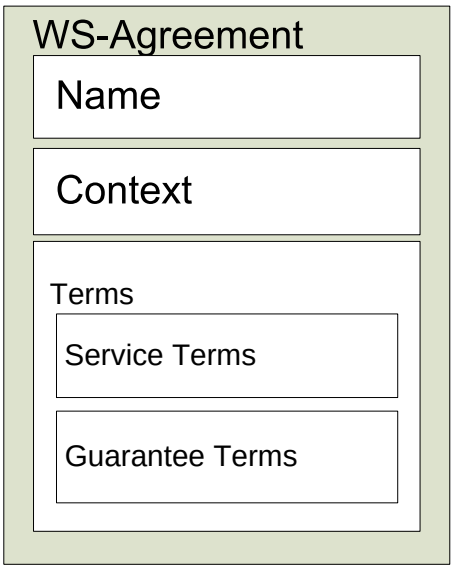


Figure 2.2.: Overview of WS-Agreement structure

needed to instantiate or otherwise identify a service to which this agreement pertains and to which guarantee terms can apply. These are further refined as service description, service reference and service property terms. Service Description Terms (SDTs) define the functionality that is delivered under an agreement. A SDT includes a domain specific description of the offered or required functionality (the service itself). The guarantee terms specify the service levels that the parties are agreeing to. Management systems may use the guarantee terms to monitor service and enforce the agreement.

There are certain guarantee terms in every agreement that ensure the consumer a certain service quality. This assurance may be in terms of certain bounds on availability of service. These bounds are referred to as Service Level Objectives (SLO). WS-Agreement has a Penalty-Reward system. Each violation of a guarantee term during an assessment will incur a certain penalty. Agreements can also be negotiated by entities acting on behalf the provider and/or the consumer. WS-Agreement also offers predefined model agreements called agreement templates. The structure of an agreement template is the same except that it may contain a section defining constraints with certain values required to create that agreement. The purpose of templates is to give guidance on what forms of offer an agreement responder wishes to receive. An agreement creation process usually consists of three steps [72]: The initiator retrieves a template from the responder, which advertises the types of offers the responder is willing to accept. The initiator then makes an offer, which is either accepted or rejected by the responder. WS-Agreement Negotiation which sits on top of WS-Agreement furthermore describes the re/negotiation of agreements. Guarantee Terms define assurance on service quality of the service described by the SDTs. The specification of domain-specific term languages is explicitly left open.

WS-Agreement has been widely adopted in the current research projects. Seidel et al [72] provide a list of projects where WS-Agreement has been used in the domain of resource management and scheduling. Ludwig et al [89] also discuss the orchestration of resources in several projects based on WS-Agreement. WS-Agreement experience document published by OGF [12] mentions several projects including AssessGrid [2], AgentScape [49], ASKALON [1], BREIN [3], VIOLA [20] etc., where WS-Agreement has been employed. VIOLA project [20] utilizes WS-Agreement as part of its Meta Scheduling Service (MSS). It is implemented as a web service receiving a list of resources pre-selected to a resource selection service. The resource reservation is based on WS-Agreement. AssessGrid [2], another European project focuses on risk management in Grid environment. It also discusses risk of an SLA violation by a resource provider. One of the components of AssessGrid uses WS-Agreement-Negotiation Protocol for negotiating SLAs with external contractors. The risk factor is also included in the SLA as an additional attribute. ASKALON [1] is a Grid research project at University of Innsbruck, Austria. It also employs WS-Agreement to make SLAs for a specified timeframe. Community Scheduler Framework (CSF) [4] has been developed by Platform Computing Inc. CSF also employs WS-Agreement specifications. AgentScape [49] has bee developed in Vrije University Amsterdam. It uses mobile agents to access computing resources. It uses WS-Agreement based negotiation infrastructure to negotiate terms of conditions and quality of service of resource access with domain coordinators. Ws-Agreement has been adopted in a Japanese Business Grid Project [114]. Ludwig et al [89] report that the WS-Agreement development is currently pursuing towards Renegotiation mechanisms and Interoperability between different implementations of WS-Agreement.

2.4.4. Comparative Analysis

The comparison table of SLA languages depicted in Figure 2.3 provides a snapshot of the discussed SLA languages.

Following are the definitions of the characteristics of SLA languages on the basis of which the comparison was made.

XML based description: An SLA language has an XML based description if the SLA is represented and processed in XML.

WSDL Support: An SLA language has this support if it is able to coordinate with WSDL.

SLA Composition: An SLA language has this support if it allows the services to compose and support integrated SLAs.

Support for Dynamic SLA: An SLA language has this support if it allows the formation of the SLA in a dynamic manner i.e. on the basis of current status of resources.

Negotiation: If an SLA language allows the party to negotiate before reaching a final SLA then it possesses the negotiation attribute.

Renegotiation: After the negotiation process, SLA is finalized. But due to any number of reasons if a party would like to renegotiate on certain terms and the SLA language frameworks allows this behavior then it is said to support the renegotiation attribute.

	SLANG	WSLA	WS-Agreement
XML based description	**X**	**X**	**X**
WSDL Support	**—**	**X**	**X**
SLA Composition	**X**	**X**	**X**
Support for Dynamic SLA	**—**	**X**	**X**
Negotiation	**—**	**X**	**X**
Renegotiation	**—**	**—**	Under development
Monitoring Support	**X**	**X**	**X**
Breach Management	**—**	**X**	**X**
Support for Pricing	**—**	**—**	**X**
SLA Template Design	**—**	**X**	**X**
Support for Sub-Contracting / Multiparty	**—**	**X**	**—**
Support for Different SLA Classes	**X**	**X**	**X**
Runtime Evaluation	**X**	**X**	**X**
Measuring and Reporting	**—**	**X**	**X**
Corrective Management Actions	**—**	**X**	**X**
Reusability	**—**	**X**	**X**
Interoperability	**—**	**—**	Under development

Figure 2.3.: Comparison of SLA Languages

Monitoring Support: As part of the Service Level Management, an SLA language keeps on monitoring the status of SLA until it expires.

Breach Management: In case of an SLA violation certain penalty actions needs to be taken.

Support for Billing: An SLA language may also compute the expenses consumed on resources and services depending upon the usage time and service quality etc.

SLA Template Design: SLA languages support SLA templates which are partially filled when created and are completed after a negotiation.

Support for Sub-Contracting / Multiparty: An SLA language may support multiparty interaction or subcontracting features which are also dependant on the nature of business relationships.

Support for Different SLA Classes: SLA languages may support different types of SLA expressing different types of business relationships.

Support for Runtime Evaluation: An SLA language may support runtime evaluation of the an SLA

Measuring and Reporting: An SLA language may offer some reporting facility during the SLA lifetime after measuring certain parameters.

Corrective Management Actions: If the breach has been identified due to some problem then SLA language may try to rectify it acting on certain rules.

Reusability: Whether an already established SLA can be reused somehow among the old partners or new similar partners.

Interoperability: Whether the different implementation of an SLA language in different organization are flexible enough to interoperate.

WS-Agreement is an initiative of Open Grid Forum (OGF) [12] and is the most discussed SLA language now a days. WSLA is developed by IBM but is almost obsolete now. As IBM is now a member of OGF consortium, so the researchers who developed WSLA have contributed in WS-Agreement development. Nurmela [99] points out many requirements to develop a family of aspect languages for NFAs : "Each language should have a sufficient set of joint basic concepts so that aggregations can be negotiated over them in a sensible way. Consequently, each broad category of business services has a separate set of concepts and related metrics, so that these are understandable to the business process designers in business terms. At the more technical level, it is required that each concept and metrics has a supported transformation to technical terms in a transparent way. Also, it is necessary that the technical level concepts and metrics are provided for communication service business". Concepts such as dynamic SLA composition, SLA negotiation and renegotiation and aggregated SLA management are continuously pushing to stretch the capabilities of these languages. SLA@SOI [22]is a European FP7 project that has been launched on June 2nd, 2008, and is committed to research, engineer and demonstrate technologies that can embed SLA-aware infrastructures into the service economy. The goal of this project include predictability of quality of services, Transparent SLA management and automation of SLA activities such as: negotiation, delivery and monitoring.

2.5. Related Work

2.5.1. Virtual Organizations and NESSI business models

The concept of Virtual Organizations (VO) was born from Grid Computing [60]. Ian Foster et al [61] describe flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources, what they refer to as Virtual Organizations (VOs). Nurmela [99] defines Virtual Organizations as loosely-coupled, inter-enterprize collaboration, and points out the need of common facilities for managing contract-governed collaborations and the autonomous business services between which those collaborations are formed. He further highlights the significance of Service Level Agreements (SLAs) in the formation of such business collaborations. Nurmela provides a detailed description of Service Level Agreement Management in Federated Virtual Organizations by comparing the Non Functional Aspects of some of the popular SLA languages. NESSI (Networked European Software and Services Initiative) [10] is a consortium of over 300 ICT industrial partners. According to NESSI-Grid's Strategic Research Agenda (SRA) [10], Virtual Enterprize Organizations (VEOs) form when two or more administrative domains overlap and share resources. NESSI considers VEOs as a business model and lists various business requirements for it. NESSI defines Business Value Networks as ways in which organizations interact with each other forming complex chains including multiple providers/administrative domains in order to drive increased business value. NESSI in its Strategic Research Agenda (SRA) has highlighted the importance of Business Value Networks [10] as a viable business model in the emerging service oriented ICT infrastructures. In addition to the notion of Business Value Networks, NESSI has pointed out various other possibilities for similar inter-organizational business models; Hierarchical Enterprizes, Extended Enterprizes, Dynamic Outsourcing, and Mergers to name a few.

2.5.2. Optimization against QoS constraints

There are a number of Workflow Management Systems available for the Grid [47] [120] [142] [26]. Workflow QoS constraints are an essential part of the workflow design and play an important role in their scheduling[139, 74, 35, 51, 117]. These have been defined [138] to consist of five components which are Time, Cost, Fidelity, Reliability and Security. These constraints can be defined at the workflow level or at the task level [139] with in a workflow. Traditional workflow systems lack the ability to be dynamically modeled and scheduled.

Mapping abstract workflows on to service enriched environments such as Grid has been a challenging research area [47] [142]. QoS constraints are an essential part of the workflow design and play an important role in service selection and scheduling [139] [38]. They can be defined at the workflow level or at the task level [139] within a workflow. Binder et al. [30] extract the resource requirements of the services from the OWL-S [91] descriptions that allow defining nonfunctional properties of the components. A mathematical model then computes the execution cost of the workflow and afterwards a genetic algorithm is used to optimize the workflow execution. This approach very successfully maps the resources on workflow tasks but does not discuss dynamically changing conditions. Ambrosi et al.

[54] propose to optimize workflow scheduling through reactive and proactive actions. They have proposed a concept of optimization by assessing the current situation and forecasting the best possibilities for the future. Their approach lacks concrete examples. Huang et al. [69] present a very good approach to workflow optimization by dynamic web service selection. An optimal service is selected based on historical data and real-time data. Their approach does not discuss the case of adapting to user-defined QoS constraints. Jia Yu et al. [139] propose a QoS-based workflow management system and scheduling algorithm for the service Grid that minimizes the execution cost and yet meets the time constraints imposed by the user. The QoS-level constraints can be defined at the task level as well as at the workflow level.

Tao Yu et al. [141] [140] have made an extensive study on service selection algorithms for composing web services with multiple QoS constraints. They have devised two kinds of models to address this problem: the combinatorial model and the graph model. Their combinatorial model describes this problem as the Multidimensional Multi-choice Knapsack Problem (MMKP) and the graph model defines it as the Multi-Constraint Optimal Path (MCOP) problem. They have evaluated different heuristic and non-heuristic algorithms after sequential executions and found the Branch and Bound algorithm to be optimal but slow. The research conducted as part of this thesis studied their approach and have gone a step ahead by implementing a parallel version of the Branch and Bound algorithm to retain the optimality while coupling it with a sequential heuristic algorithm to ensure the efficiency.

Blackboards [42] are a mechanism to solve complex problems and have been successfully employed in various fields. Following [133], the author of this thesis, has contributed in [115] in terms of the development of a blackboard [42] approach coupled with an A^* algorithm to automatically construct and optimize the Grid-based workflows.

2.5.3. Views and e-Contracts in Workflows

The concept of Workflow Views is used to maintain the balance between trust and security among business partners [37, 48, 52]. Schulz et al [78] have introduced the concept of view based cross-organizational workflows and call it as coalition workflows. Liu et al have contributed in the process-view model [48, 52]. Chiu et al [40] present a meta model of workflow views and their semantics based on supply chain e-service but their model lacks an integrated cross-organizational perspective. Other authors [116, 85], however, do propose a global view or a decomposition process based on the views. But none of them have focussed on the dynamic workflows in their approach. Static and dynamic verification of temporal constraints [38, 39] is very crucial in workflows to avoid any temporal violations during the workflow life cycle. Eder et al [53] employ the concept of views to calculate the temporal consistency of interorganizational workflows by using abstraction and aggregation operators of views but their approach is also limited to static or predefined workflows. Chebbi et al [37] provide a very comprehensive approach that is view based, web services focused and is applicable to dynamic inter-organizational workflow cooperation. This means that the cooperation across organizations is described through views without specifying the internal

structure of participating workflows. This concept of contracts is similar to that of SLA although SLAs are more dynamic due to negotiation, renegotiation and fault tolerance features. There is a very relevant work done by Chiu et al. [41] in terms of a contract model based on workflow views. They demonstrate how management of contracts can be facilitated. They start with an example, highlight domains of different participating organizations and then develop a model to identify the corresponding workflow views. They go on further to develop an e-contract model that defines e-contracts in plain text format. This is an old paper and the modern form of e-contracts are Service Level Agreements which are XML based, more complex and more dynamic due to features such as negotiation, renegotiation, and fault tolerance, etc. Furthermore their approach starts with defining views in an inter-organizational workflow and then describing e-contracts to enforce the obligatory communication links in the views. The model presented in this thesis allows SLAs to maintain their individual identity. Therefore views are defined directly on the SLA aggregation structure rather than on workflows. Moreover the proposed approach also provides a formal description of hierarchical SLAs and their aggregation.

2.5.4. Distributed Trust and Security

In the Trust-EC project, Jones [76] defines trust as "the property of a business relationship, such that reliance can be placed on the business partners and the business transactions developed with them". Transport Layer Security (TLS) is a popular authentication protocol in VOs [50]. It is derived from the Secure Sockets Layer (SSL) protocol [63] and uses X.509 public key certificate [68], which binds a Distinguished Name (DN) to a public key. The binding is attested to by a Certification Authority (CA) [70]. Kerberos [98] is an authentication system designed to allow a single sign-on to many machines within a single administrative domain. The Grid Security Infrastructure (GSI) and the security modules of middle-ware, provide a set of security protocols for achieving mutual entity authentication between a user (actually a user's proxy) and resource providers [144]. Each party has a public-key based cryptographic credential in the formulation of a certificate. GSI uses X.509 proxy certificates (PCs) to enable Single sign-on and Delegation [86]. PCs can be created on the fly without requiring any intervention from conventional CAs. In the cross-CA Hierarchical trust model [86][144], the top most CA is called the root CA that provides certificates to its subordinate CAs. These subordinates can further issue CA to other CAs (subordinates), services or users. Community Authorization Service (CAS) [111, 64] allows the expression of policies regarding resources distributed across a number of sites. Similarly, the Virtual Organization Membership Service (VOMS) [21] also gives the capability to provide authorization information by a secure server that the local site has chosen to trust.

2.6. Projects Relating SLA Management

This section provides a summary of some of the important SLA related projects. Most of the chosen projects are still in progress and represent the state of the art of the SLA

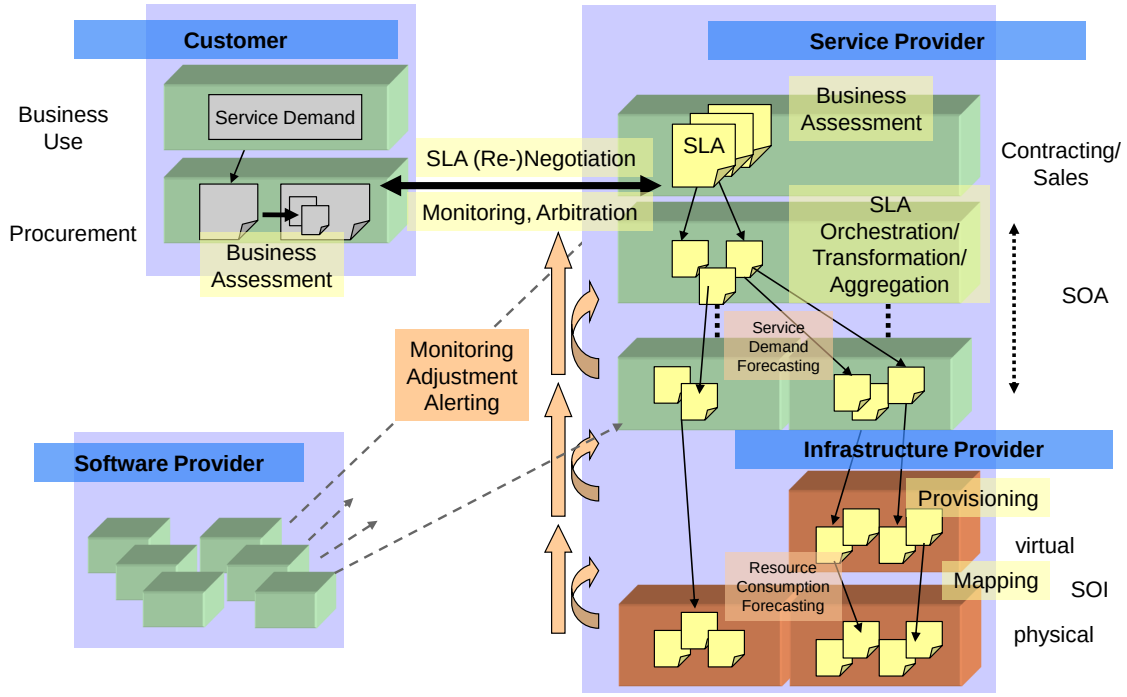


Figure 2.4.: SLA@SOI Project

related research but some of them have been chosen on the basis of either their historical significance or a special relevance with the research presented in this thesis. European Community's Framework Programme (FP) is the main funding source of the overall scientific activities going on in the European research landscape. FP7 or Seventh Framework Programme is European Union's Framework Programme for research and technological development for funding research over the period 2007 to 2013. Information and Communication Technology is one of the themes of FP7, whose Challenge 1 states as: *Pervasive & Trusted Network & Service Infrastructures*. The Objective 1.2 of this challenge is: *Internet of Services, Software and Virtualisation*. Within the objective "Internet of Services", some of the projects broadly divided into the areas of *Service Architectures* and *Virtualized Infrastructures* are directly or indirectly address SLA related issues. Some of the important projects of FP7 are: SLA@SOI, SOA4All, mOSAIC, Romulus and NEXOF-RA. The summary of the SLA related activities going on as part of these projects will be presented in this section. The introductory information of these projects has been extracted (some times copied) from their respective web sites, which have been appropriately cited with the description. The order of the projects is associated with their relevance to the theme of this thesis.

2.6.1. SLA@SOI

SLA@SOI [22] is a project within the European Union's 7th Framework Programme for Research and Technological Development, Theme Information and Communication Technologies. The project is coordinated by SAP AG, Germany and was launched on June 2008. SLA@SOI is the most important and most comprehensive FP7 project in context with SLA management. The vision [22] of the project is: a business-ready service-oriented infrastructure empowering the service economy in a flexible and dependable way. This project very comprehensively discusses the issues regarding SLAs in Service Oriented Computing. The project focusses on 3 goals: Predictability and Dependability of services at runtime; Transparent SLA Management; and Automation of SLA formation process. Figure 2.4 highlights the overall approach of SLA@SOI project.

The technical approach of SLA@SOI is to define a holistic view for the management of service level agreements (SLAs) and to implement an SLA management framework that can be easily integrated into a service-oriented infrastructure (SOI). The main innovative features of the project are:

1. an automated e-contracting framework,
2. systematic grounding of SLAs from the business level down to the infrastructure,
3. exploitation of virtualization technologies at infrastructure level for SLA enforcement, and
4. advanced engineering methodologies for creation of predictable and manageable services.

SLA@SOI aims to contribute in the evolution towards a service-oriented economy and a flexible instantiation of dynamic value networks. One of the ongoing tasks of SLA@SOI as highlighted in Figure 2.4 is multi level SLA aggregation in connection to the service composition. This topic is very much similar to the aggregation problem addressed in this research work. The research work presented in this thesis is expected to complement the output of SLA@SOI.

2.6.2. FOSII

Foundations of Self-Governing ICT Infrastructures (FOSII) [5] is funded by WWTF (Vienna Science and Technology Fund) and is being carried out at Technical University Vienna. The project investigates the problems arising from the lack of dynamism and adaptivity in so-called service-oriented architectures (SOA).

FOSSI project has introduced the Layered Approach for SLA-Violation Propagation in Self-manageable Cloud Infrastructures (LAYSI) and Low level Metrics to High level SLAs (LOM2HiS) models. LAYSI talks about several plug and play components joined through SLAs, which constitute a Cloud infrastructure to provide service utilities. A proactive validation approach can adjust and fine tune these components to bring under control the violation threats associated with the non-functional attributes of the services. LoM2HiS

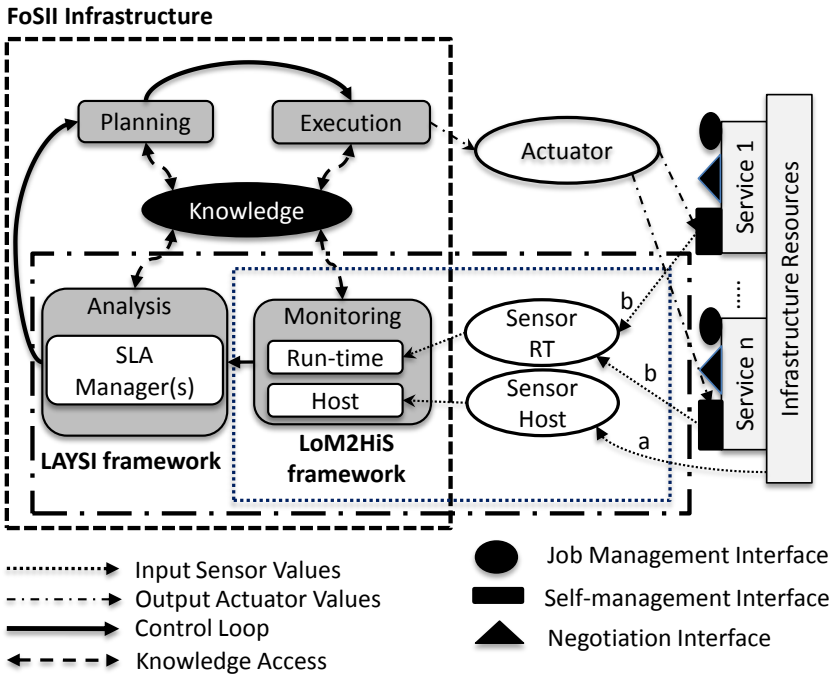


Figure 2.5.: LoM2HiS model as part of FOSII Infrastructure

is focussed on mapping high level SLA objectives to low level resource metrics so that the resources can be adjusted in case of SLA violation threats.

Figure 2.5 depicts the LoM2HiS model as part of the FOSII infrastructure. The overall management is done through the MAPE model whose stages are described as follows:

- Monitoring: The QoS managed element is monitored using adequate software sensors shown in Figure 2.5 .
- Analysis: The monitored and measured metrics corresponding to SLOs such as response time, availability, etc. are analyzed using base knowledge (condition definition, condition evaluation, etc.).
- Planning: Based on the evaluated rules and the results of the analysis, the planning component delivers necessary changes on the current setup, e.g. renegotiation of SLAs which do not satisfy the established QoS guarantees.
- Execution: Finally, the planned changes are executed using software actuators and other tools.

As part of FOSII, some very useful work has been carried out in the domain of SLA mappings between service users and service providers. VieSLAF Framework [34] presents an architecture that enables the service providers to publish their SLA templates in a

repository. The user requirements are also transformed into an SLA template, which is matched with the published templates to find the closely matching offers.

The author of the thesis has worked with the FOSII team and has published an architecture [122, 128] combining the validation framework proposed in this dissertation and the LAYSI and LoM2HiS models of the FOSII infrastructure. This collaboration is perceived to move forward in the coming months.

2.6.3. RBSLA

The Rule Based Service Level Agreements (RBSLA) project [14] focuses on sophisticated knowledge representation concepts for Service Level Management (SLM) of IT services. At the core of an RBSLA contract and service level management tool are rule-based languages to describe contracts such as service level agreements or policies in a generic way. The research draws on basic knowledge representation concepts from the area of artificial intelligence (AI) and knowledge representation (KR) and as well as on new standards in the area of web services computing and the semantic web. A particular interest is the investigation of expressive logic programming techniques and logical formalisms such as defeasible logic, deontic logic, temporal event/action logics etc. An implementation of a rule-based service level management tool (RBSLM) has been built with a computational model based on the ContractLog KR and the open source rule engine Prova. The research concerned with thesis utilized some of the RBSLA constructs for the simulation of SLA validation. RBSLA rules were employed in a distributed environment based on Rule Responder architecture [23] to simulate cross enterprize validation of SLA Choreography.

2.6.4. COSMA

COSMA [88] is an approach developed by the University of Leipzig to extend SLA management in the context of composite or hierarchical service provision. This project has objectives very similar to those addressed in this thesis. The requirements of composite service providers refer to the SLA management of so called atomic services, the service requestors of composite services and the depiction of dependencies and relations of these service structures, to control and optimize the SLA management activities. COSMA addresses the service composition with reference to the significance of Composite Service Providers (CSP) in service markets. The CSP is responsible for the reliability of the composite service and acts as a single point of accountability to the customer. COSMA covers the life cycle of SLA in composite services and focuses especially on SLA creation, negotiation, monitoring and evaluation.

2.6.5. SLA4D-Grid

The SLA4D-Grid Project [16] aims to design and realize a Service Level Agreement layer for the Germany's national Grid infrastructure known as D-Grid. The Service Level Agreement layer offers the users, the D-Grid community, and the service providers the required QoS assurances and pre-defined business conditions. With SLA4D-Grid, SLAs for the D-Grid

services will be automatically created, negotiated, monitored in an economically efficient way in accordance with respective business models.

2.6.6. LIBRA

LIBRA [6] in *CloudLab*, University of Melbourn, Australia, aims at *An Economy-Driven Cluster Schedulng and Service Level Agreements (SLA)-based Resource Allocation System.* The main objectives of the project include:

- development a QoS-based scheduler for resource management on a homogenous cluster,
- user-directed time and cost optimization of the scheduler for sequential and parallel jobs,
- testing the scheduler by simulations

The main target is to allocate resources keeping in view user-centric job priorities.

2.6.7. MASCHINE

The MASCHINE project [7] is a collaborative effort between University of Michigan and Cornegie Mellon University USA. MASCHINE stands for stands for MultiAttribute Supply CHaIn NEgotiation. This project focuses on supply chains and aims to support negotiation over many attributes in dynamic, multilevel supply chains. The objectives include general two-sided matching mechanisms for both iterative and one-shot negotiation and real-time decision support based on summarized assessments of production states.

2.6.8. mOSAIC

moSAIC [9] is one of the latest FP7 projects, which focusses on SLA negotiation for inter-Cloud infrastructures. The mOSAIC project aims to develop an open-source platform that enables applications to negotiate Cloud services as requested by their users.

Figure 2.6 provides an overview of the moSAIC architecture. The platform is designed as a multi-agent brokering mechanism that provides requirement-driven search out of multiple Cloud infrastructures and facilitates service composition in absence of a direct solution.

2.6.9. SOA4All

SOA4All [17] is a project within the European Union's 7th Framework Programme for Research and Technological Development, Theme Information and Communication Technologies. SOA4All expects to support the paradigm of service orientation in information technology based solution provision for a large-scale of parties. The overall goal of SOA4All is to provide a framework that combines approaches and technologies like SOA, Web 2.0 and semantic web with web principles and context management into a domain-independent service delivery platform.

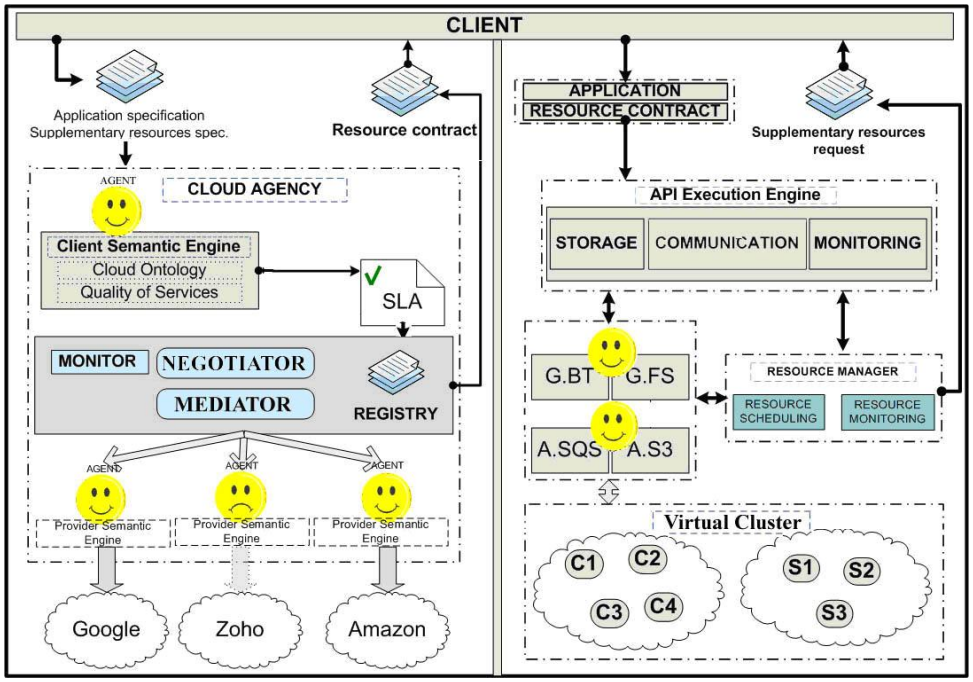


Figure 2.6.: moSAIC Architecture

As shown in Figure 2.7, SOA4All architecture supports various types of services connected through an Enterprize Service Bus (ESB). The platform services of SOA4All project deliver service discovery, ranking and selection, composition and invocation functionality.

2.6.10. BREIN

BREIN [3] is an EU project and stands for Business objectives driven Reliable and Intelligent Grids for Real Businesses. BREIN aims at taking the e-business concept of "dynamic virtual organizations" towards a more business-centric model, by enhancing the system with methods from artificial intelligence, intelligent systems, semantic web etc. Cloud Computing has attracted BREIN team and they have also expanded to cloud based service provision models. In this regard BREIN has a strong emphasis on SLA based service provision as is evident from the BREIN objectives shown in Figure 2.8.

2.6.11. NEXOF-RA

NEXOF-RA (NEXOF Reference Architecture) [11] project is the first step in the process of building NEXOF the generic open platform for creating and delivering applications enabling the creation of service based ecosystems where service providers and third parties easy collaborate. NEXOF-RA main results will be the Reference Architecture for NEXOF, a proof of concept to validate this architecture and a road map for the adoption of NEXOF

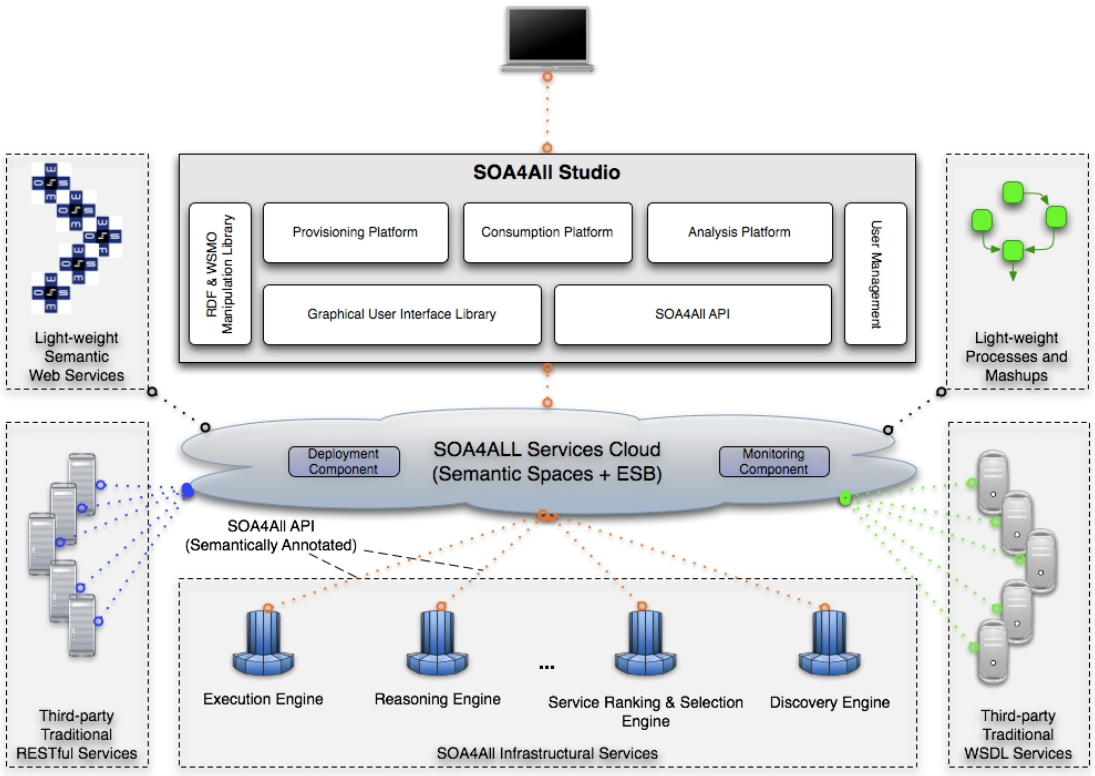


Figure 2.7.: SOA4All Architecture

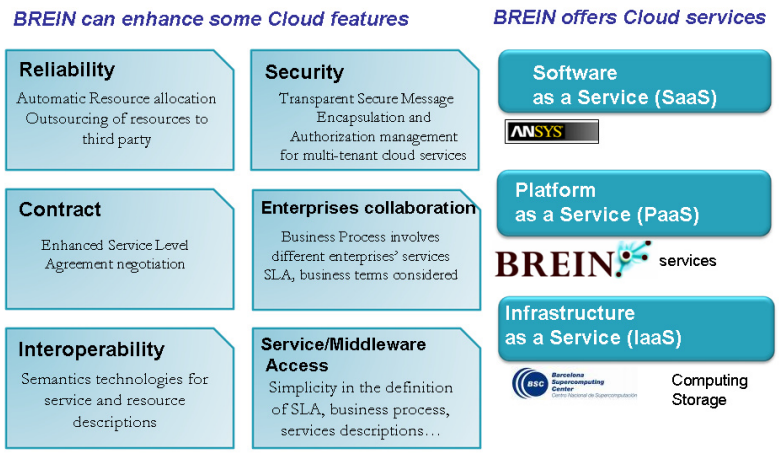


Figure 2.8.: BREIN Objectives

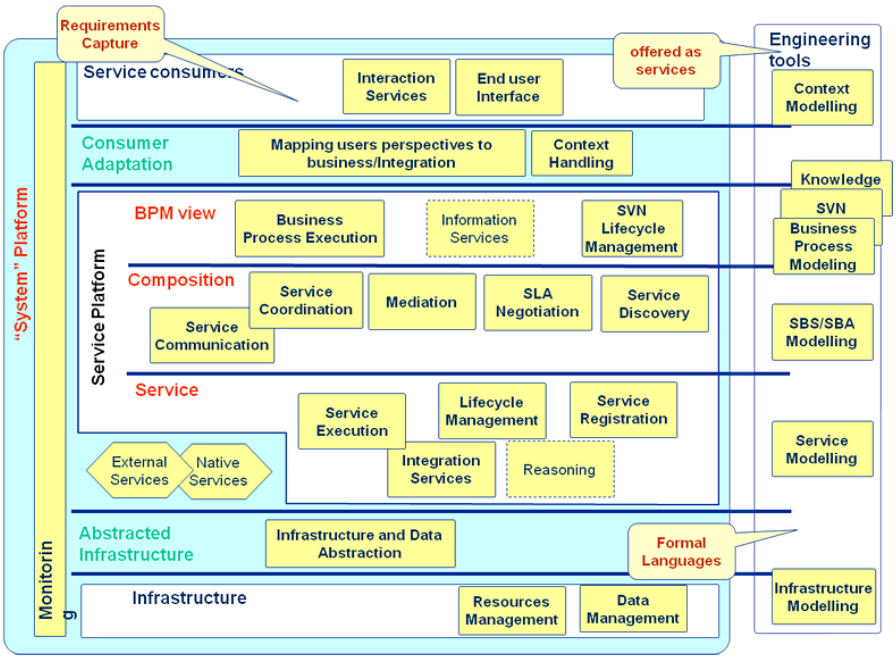


Figure 2.9.: NEXOF Architecture

as a whole. To build the specifications for the Open Framework Architecture, an open process has been defined to allow the involvement of all relevant initiatives and organizations concerned on building a Reference Architecture for the “Future of Internet".

The composition layer of the service platform depicted in Figure 2.9 is of special importance within the context of SLA-based provision of services.

2.6.12. MASTER

Managing Assurances, Security and Trust for Services (MASTER) [8] aims at providing methodologies and infrastructures that facilitate the monitoring, enforcement, and audit of quantifiable indicators on the security of a business process, and that provide manageable assurance of the security levels, trust levels and regulatory compliance of highly dynamic service-oriented architecture in centralized, distributed (multi-domain), and outsourcing contexts.

2.6.13. Romulus

ROMULUS [15] is a project within the European Union’s 7th Framework Programme for Research and Technological Development, Theme Information and Communication Technologies. ROMULUS goals include integration of Mashup oriented development to support reusability and to increase development productivity and integration of soft goals

related to quality (non-functional) requirements like reliability, security, traceability of scalability in the software development process.

2.7. Summary

This chapter introduced basic concepts that will be used in the next chapters as well as the state of the art in the field of SLA management, aggregation and validation. This chapter can be logically divided into four parts. Utility Computing and its sister technologies such as Cloud Computing, Service Oriented Computing etc. were briefly explained in the first part of the chapter. In the second part, the notion and significance of SLA was explained. Several definitions, a survey of SLA languages and the an explanation of the constitutional parts of SLA were used to elaborate the significance of SLA in SOA. A survey of the related concepts was presented in the third part of the chapter whereas the fourth part highlighted some of the most important SLA related projects in various universities and research institutes. This chapter aims at building a comprehensive understanding of the role of SLA in SOA and Utility Computing. This knowledge is intended to help the reader understand SLA-based concepts introduced in the next chapters.

Chapter 3.

A Framework for SLA Centric Service-Based Utility Computing

The whole is more than the sum of its parts.

(Aristotle)

This chapter elaborates the overall framework for enabling SLA centric service-based Utility Computing. The framework consists of four components:

- SLA oriented selection and negotiation of services
- SLA Choreography and its hierarchical aggregation
- Hierarchical validation of SLA Choreography, and
- Enabling requirements in terms of privacy, trust and security for the stakeholders. An architecture of the framework is presented and discussed. The framework is also explained with the help of a phased process diagram.

3.1. Service-Based Utility Computing

In service based Utility Computing SLA is essentially important for the service consumer as it compensates consumer's high dependency on the service provider. Service composition directly implies the need for composition of their corresponding SLAs. So far, SLA composition was mostly considered as a single layer process [31]. This single layer SLA composition model was insufficient to describe supply-chain based business networks. The research community has just started taking notice [22] of the importance to describe this hierarchical aggregation. In a supply-chain, a service provider may have sub-contractors and some of those sub-contractors may have further sub-contractors or fourth party logistics making a hierarchical structure. This supply-chain network may emerge as a Business Value Network across various Virtual Organizations. Thus a major challenge to enable IT-based Service Markets is to foster these hierarchical service composition scenarios and their underpinning business networks and supply chains. From business' point of view, the most important asset is the extraction of value from every node of such business networks in a transparent and secure manner. To enable these supply-chain networks as Service Oriented

Infrastructures (SOI), the case of the Service Level Agreements needs to be elaborated and its issues resolved. The most important of these issues include selection, negotiation, aggregation and validation of services and their enabling requirements such as privacy, trust, and security. Regarding enabling requirements, for instance, it is not sensible to expose the complete information of SLAs across the whole chain of services to all the stakeholders. Not only because of the privacy concerns of the business partners, but also disclosing it could endanger the business processes creating added value. SLA@SOI [22] is a European project that focusses on SLA issues in SOI. On its agenda is the provision of such Service aggregators, that offer composed services, manageable according to higher-level customer needs. In SLA@SOI's vision, service customers are empowered to precisely specify and negotiate the actual service level according to which they buy a certain service. Although SLA@SOI discusses the importance of service chains but due to a very focused approach several important issues do not find a place on its agenda e.g., the role of SLAs in value multiplication, flexible negotiation models, trusted service and distributed validation etc. In this regard, this research complements the research agenda of SLA@SOI project.

3.2. A Framework for SLA Centric Utility Computing

SLAs play a pivotal role in the realization of service based Utility Computing. On-demand service provision is ensured through SLAs both for clients and service providers. This research visualizes the whole life cycle of service provisioning in Utility Computing centered around SLAs. Starting from the service selection, SLAs steer through the negotiation, hierarchical aggregation and validation, all the way up to the fault tolerance of the services. The proposed framework presents a set of solution components that collaborate in fabricating a basic infrastructure for enabling service oriented Utility Computing. The framework consists of four basic components, which have an individual significance when considered separately as well as a systematic importance when align together.

3.2.1. Architecture

Figure 3.1 shows the architecture of the proposed framework for enabling SLA centric service oriented Utility Computing. The framework consists of four main components or models, which are designed to provide solutions to the open questions asked in Chapter 1. The overall goal of this architecture is to elaborate the role of SLAs as an enabling technology for service oriented Utility Computing. The architecture consists of the following models:

1. SLA oriented service selection and negotiation
2. Hierarchical aggregation of SLA Choreography
3. Hierarchical validation of SLA Choreography, and
4. Privacy trust and security

We discuss these components one by one.

3.2.1.1. SLA Oriented Selection and Negotiation of Services

For Utility Computing, user-driven service selection is very important to ensure users' QoS requirements. The situation becomes complex in case of service composition when user constraints require translation not only locally i.e. addressing at the level of individual services but also globally i.e. directing the desired behavior of the composed services. Examples of global constraints are cost and total allowed time. Another example is the sum of reputation of the selected services. This is ensured by the reputation based trust model, which complements the selection model in this framework. The reputation of the services play a crucial rule in service selection. The reputation is updated depending upon the performance of services in the validation process. These details have been addressed through a formal model, which serves as a basis of a two-phase selection algorithm employing a combination of Branch and Bound and heuristic approaches. The formal model for selection is then extended into a (re)negotiation model for configurable services allowing flexible negotiation with the selected services. The renegotiation is required as part of the fault tolerance process in case of service failure detection through the validation model of the framework. SLA negotiation results in binding SLAs, which connect partners across different layers to form hierarchical structures such as service value chains or business value networks. The corresponding service scattered across various administrative domains result in service choreographies.

3.2.1.2. Hierarchical Aggregation of SLA Choreography

In this research, it has been argued that with every service choreography, there must be an associated SLA choreography that for the purpose of comprehension, needs to be formally described and its enabling requirements identified. A formal model of SLA Choreography not only provides a better understanding of the problem but also facilitates a generic platform for designing and implementation of different use cases. The hierarchical SLAs across the SLA Choreography require a mechanism for step-wise aggregation to automate the formation of service value chains and business networks in service enriched Utility Computing environments. Several patterns for hierarchical SLA aggregation have been discovered and formalized as part of this research.

In the formal model, the concept of SLA-views has been introduced to preserve the privacy of a contributing stakeholder. Each business partner has its own view comprising of its local SLA information. The aggregated effect of these views emerges as the overall SLA Choreography. The concept of privacy is highly entangled with trust and security in fact they complement each other in the proposed framework.

3.2.1.3. Hierarchical Validation of SLA Choreography

The Validation model is a rule based intelligent system which coordinates very closely with the Trust model. An aggregated SLA is represented by a logical rule with different premises representing various SLOs. During validation this aggregated rule becomes a distributed query, which is decomposed across the SLA Choreography with its parts getting

Figure 3.1.: An SLA-Centric Framework for Service-Based Utility Computing

validated in their respective SLA-Views, scattered across different VOs and connected via the distributed trust model. The Delegation of Validation (DOV) approach facilitates the validation query with the required interoperability while keeping the local management schemes intact. The Multi Agent System (MAS) weaves together different components by representing them through agents equipped with knowledge bases. The validation can invoke penalty enforcement process for violating services. The reputation of the under-performing services is degraded during the validation process, which subsequently reduces their chances of selection in the next phase of service selection process. The reputation of well performing services are upgraded in the same way. The validation can also lead to renegotiation in an attempt to tuneup a service's attributes or to upgrade a low performing service.

3.2.1.4. Enabling Requirements: Privacy, Trust and Security

For the selection, negotiation, hierarchical aggregation and validation of SLAs there are certain business and technical requirements, which not only help enable these activities but also play an essential role in bringing these activities within one framework. The most important of these enabling requirements are privacy, trust and security of the stakeholders. Privacy is taken care of by SLA Views. SLA Views are easily implemented by different agents of the Rule Responder architecture. For security and trust, a hybrid trust model is designed that combines the attributes of PKI and reputation based trust models. The PKI based characteristics provide the distributed queries the functionality of single sign-on and delegation. The reputation based trust management plays a crucial role during service selection, SLA validation and fault tolerance processes. The hybrid trust management system also provides a third party trust manager which can be easily extended for mediation during reputation management, penalty enforcement, payment etc. In addition to these three basic requirements, there are several others, which are either based on or can be extended on to these such as business automation, distributed query processing, fair trade, transparent payment etc.

3.2.2. Phased Process Model

Figure 3.2 elaborates the process view of the proposed framework with even more detail. The process column shows four process phases spanning across the whole life cycle of SLAs corresponding to a service choreography.

1. SLA Oriented Selection of Services: A pool of services is published using WSDL or WS-Agreement (SLA templates) and thus available for selection. Services are selected on the basis of user-defined QoS constraints and mapped on an abstract workflow. A formal model for service selection has been implemented using Branch and Bound and Heuristic algorithms. The output milestone of this phase are service mappings on the activities of an abstract workflow.

2. SLA Negotiation and Binding: The services short listed as a result of the previous phase go through a flexible process of negotiation. The SLA negotiation protocol based on the concept of dynamic service configuration helps fine tune the mutual requirements by adjusting the service parameters through a negotiation/renegotiation protocol. The output milestone of this phase results into SLA bindings.

3. Hierarchical SLA Aggregation: The SLAs formed at various points in the service choreography are represented through the formal model for SLA Choreography and can be aggregated by using various aggregation patterns. The output milestone of this phase is the SLA Choreography and its aggregation in connection with the underlying service choreography and its aggregation.

4. Hierarchical SLA Validation and Fault Tolerance: The SLA Choreography and aggregation needs to be validated for consistency and fault tolerance reasons. Rule

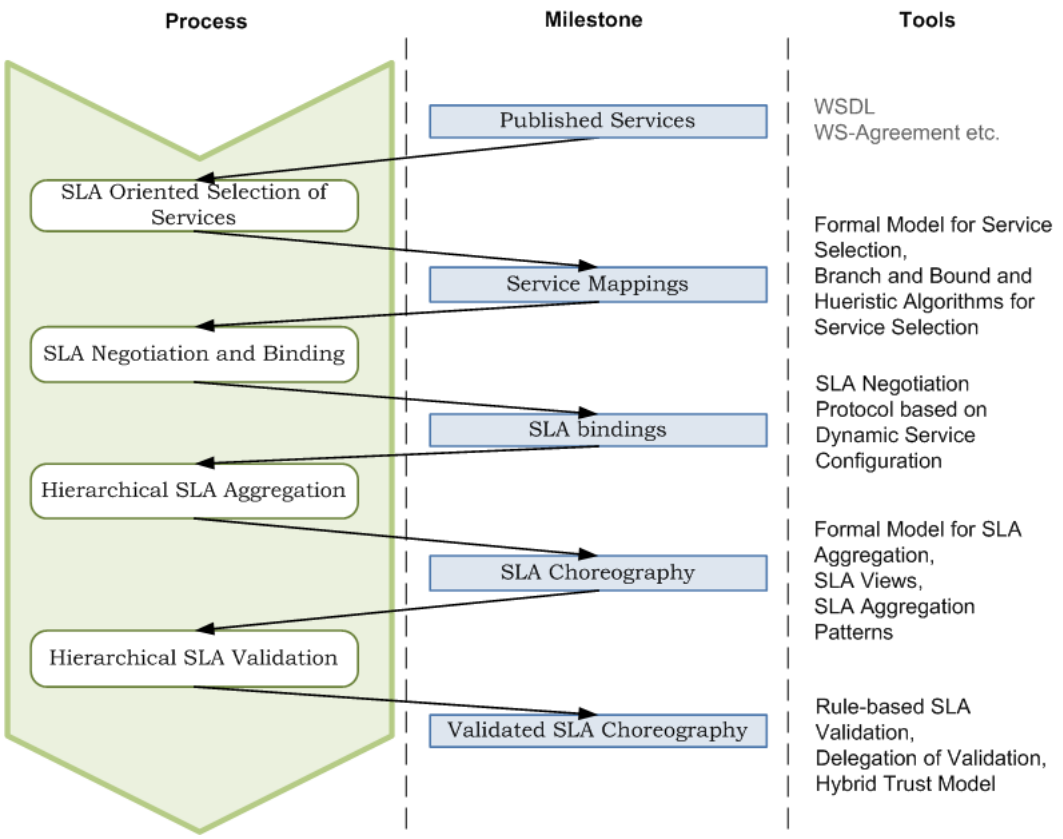


Figure 3.2.: Phased Process Model for an SLA-Centric Framework for Service-Based Utility Computing

based SLA validation couples with a hybrid trust system based on the PKI and reputation based trust models facilitates this distributed hierarchical SLA Validation mechanism. The output milestone of the phase is a validated SLA choreography or SLA violation detection.

3.3. Motivational Scenario

A running example based on a motivational scenario will be used throughout the thesis in order to better comprehend various concepts employed in different components of the framework. The running example will pass through a metamorphosis as these concepts will gradually mature and evolve through various stages.

In our motivational scenario, Arfa is a graphics designer and she has just finished designing an animation involving thousands of high resolution images. Now she needs to carry out hi-tech multi-media operations such as rendering and editing. She plans on utilizing online services to accomplish these tasks. Afterwards she would like to utilize some graphical compression service to compress the rendering output and a youtube like hosting

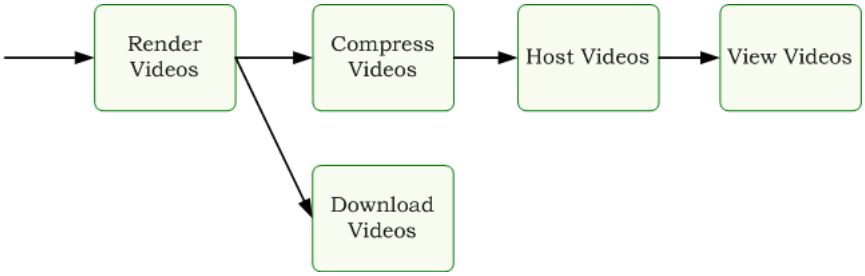


Figure 3.3.: Motivational Scenario

service for quick view. She would also like to store the detailed rendering output in the original high quality. She chalks out these activities in an automated workflow tool which can search appropriate cloud services to map out the workflow activities. This has been depicted in Figure 3.3.

It will become evident in the later chapters that this simple scenario to be practically realized requires numerous SLA oriented activities. From the selection of the best services matching user requirements up to the penalty enforcement for the failing services, the role of SLA is the most critical.

3.4. Summary

This chapter proposes an SLA-centric service based framework for Utility Computing. The framework covers various aspects of the role of SLA in the entire life cycle of service provision and utilization. Service selection, negotiation, composition, validation and fault tolerance are the different stages where the crucial role of SLAs has been identified and integrated in the form of the proposed framework. The design of the framework is generic enough to be applicable in various types of Utility Computing implementations. However, a special emphasis has been given to the micro-economical players in Cloud Computing for enabling novel business models based on service aggregation and service value chains.

Chapter 4.

SLA-Based Selection and Negotiation of Services

What you seek is seeking you.

(Mowlana Jalaluddin Rumi)

The contributions of this chapter are listed as follows.

- A formal model is presented to automate the selection of optimal services fulfilling user requirements and to facilitate the process of SLA negotiation between the client and the service provider. The client can express its requirements and priorities according to which the best matching services are selected from a service-enriched environment.
- A branch and bound algorithm built upon the formal model parallelizes the optimization process regarding the selection of services fulfilling user requirements.
- A heuristic algorithm copes with dynamic changes in user requirements by updating an existing solution.
- A multi-round negotiation/renegotiation protocol is formalized and presented. The client can also negotiate with the best complying service provider and can request to improve the service parameters according to the client's specific requirements. The service provider would try to come up with a configuration closest to the client's specifications.

4.1. Background

With the popularization of SLA-based Utility Computing especially in the form of Cloud Computing infrastructures, there is a high likelihood for an IT-based service economy to cause a major shift from Capital Expenditure (CAPEX) to Operational Expenditure (OPEX) based enterprize setups. This will bring about new business models which will encourage resellers and Composite Service Providers [88] [36], not only affecting Small and Medium Enterprizes (SME) but also directly promoting the micro-economical sector. For this, services of varying granularity and customizable configurations will be contracted through SLAs as on-demand consumable resources similar to the metered public utilities such as electricity, gas, water and telephone. This will attract both service providers and

service consumers alike with its promise of reduction of cost based on the pay-per-use model and the shift from the usual capital upfront investment model to an operational expense [73].

The pay-per-use service models not only help in the reduction of cost, but various services after being composed together with the help of orchestration technology become available again as more capable composite services with a guaranteed level of service. Services of different granularity are required to be searched, selected and short-listed in compliance with the user requirements. This requires consumer-directed QoS-based selection of the most appropriate services, which can be composed together to fulfill client's requirements. There may be several constraints imposed by the client, which need to be taken care while selecting a set of services best compliant to these restrictions. To device a generic methodology, this problem should be first formalized in order to device suitable algorithms, which present a list of the most appropriate services. The consumer needs to establish SLAs with the short-listed services before using them.

However, computing utilities are very different from other commodities due to their highly dynamic nature and flexibly configurable attributes. This requires new trade mechanisms. A supermarket approach [96] i.e., a take-it-or-leave-it negotiation model, is drastically insufficient to harness the optimal business value of IT-based service markets. In such a market, a single service can be packaged into several different products depending upon its varying configurations. Moreover these configurations cannot be prepackaged due to customized requirements of clients. To cope with this situation, in addition to many other enabling requirements, there is a strong need for dynamic and flexible negotiation mechanisms, which allow service providers to dynamically compute customizable service configurations against consumer specifications following the business policies of the service provider at the same time. User directed service selection followed by SLA negotiation results into various SLAs formed between services and the client. The same selection and negotiation algorithms can be employed by services to form composite services.

4.2. Running Example – User-Driven Service Selection

Referring to the running example introduced in Chapter 3, the user Arfa needs to carry out hi-tech multi-media operations such as rendering and editing and searches for online services. She draws an abstract workflow depicted in Figure 4.1. Her workflow tool is also equipped with social networking capabilities and recommends to her the services which have attributes closest to her matching requirements as well as highest reputation among her trusted friends.

She also mentions her constraints in terms of minimum CPU power, graphics resolution, total time and total cost that she can afford as well as the minimum reputation that she requires from the services. A set of services best fitting to these requirements is selected through a Branch and Bound algorithm. Seeing that she still can afford to spend more, she raises her requirement for computation efficiency and again submits her request to search for the best complying services. This time, she gets results much sooner than before as a

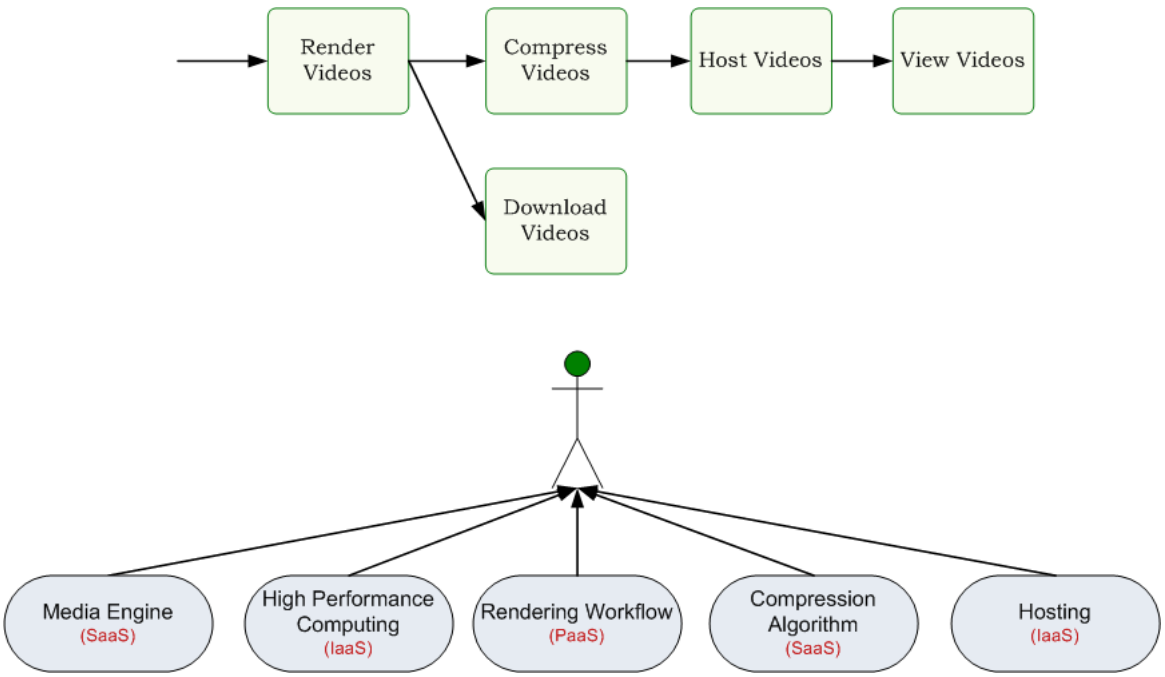


Figure 4.1.: Running Example-User Driven Service Selection

heuristic-based algorithm replaces only one of the services in the workflow and proposes a solution. The next step is to establish SLAs with the selected services. Arfa observes that the hosting service demands for a one year hosting contract. She contacts the service by sending an offer for 3 months hosting. After a couple of negotiation rounds a reasonable price for 3 months hosting is fixed between the two parties and an SLA is established.

4.3. SLA-Based Selection of Services

Customer satisfaction, which is primarily based on the fulfillment of user-centric objectives, is a crucial success factor to excel in IT-based service markets [87]. Consumers will be able to access services from the Cloud under their desired level of service by mentioning the Quality of Service (QoS) requirements. Consumers and service providers will be bound to these requirements through Service Level Agreements (SLAs). The most convenient way for the end-user to specify her requirements is in form of an abstract workflow, which allows a user to draw a sequence of activities representing his desired services along with the user's functional and nonfunctional requirements. Suitable services satisfying user requirements are then searched from a pool of services, and mapped on the abstract workflow. The optimal service composition requires the best selection out of the available services. The selection of the services is based on user requirements. These requirements can be functional requirements such as order of the required services or non-functional such as

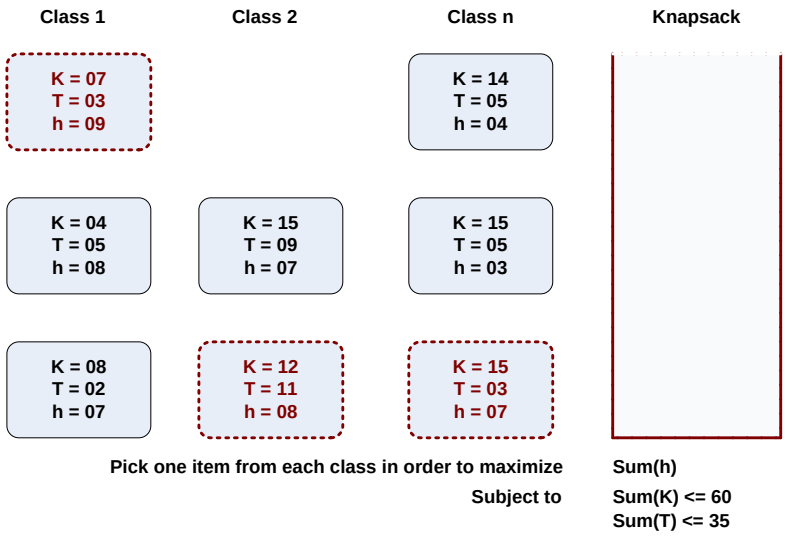


Figure 4.2.: An example of the Knapsack problem with K representing total cost, T total time, and h as degree of happiness

total cost, total response time, availability, reliability and trust etc.

This leads to a Multi-dimensional Multi-choice Knapsack Problem (MMKP) [140][141], where optimum selection of weighted items based on multiple parameters is required to be made from various sets such that only one item can be selected from each set and the combined weight of the selected items must be below a certain limit. Figure 4.2 describes the Knapsack problem with variables K, T and h.

4.3.1. Formal Model for Service Selection

To develop a criterion for workflow QoS optimization, first one needs to define various parameters of the contributing knowledge sources. The knowledge sources are considered to offer services that are mapped on the activities of the abstract workflow. A service is selected on the basis of how much it fulfills user requirements. This is done by matching the attributes of a service with user requirements. User requirements may specify two values: a minimum value that is a must requirement and a desired value that is his highest wish-level. It is required to describe all these interrelated concepts in an explicit and formal manner.

4.3.1.1. Definitions

4.3.1.1.1. Definition (Service Attribute and Attribute Value) A *service attribute* is a pair $a_{il} = (D_{il}, n_{il})$ where D_{il} is a set called the *definition domain* (most commonly, we will have $D_{il} \subseteq \mathbb{R}$, this also covers booleans if we identify *true* as 1 and *false* as 0) and $n_{il} : D_{il} \to [0, 1]$ is a map called the *normalization map* for the attribute. It represents a

QoS parameter such as the compression rate. The definition domain specifies the possible values the QoS attribute can take, the normalization map specifies how to map those values to a quality between 0 and 1, where 0 is the worst possible quality and 1 the best one. The map can be increasing or decreasing: it will be increasing for attributes which directly indicate a quality such as the reputation of a service; it will be decreasing for attributes such as latencies where less is better. An *attribute value* is a value $(Q_{0i})_l \in D_{il}$. It specifies a concrete value the attribute can take.

4.3.1.1.2. Definition (Service Class) A *service class* c_i is a list of attributes $a_{i1}, \ldots, a_{im}$ (i.e. we define a service by its attributes). It models a set of equivalent services, e.g. the video compression services. The *normalization map* $n_i : D_i = D_{i1} \times \ldots \times D_{im_i} \to [0,1]^{m_i}$ for the service is the map mapping each attribute value $(Q_{0i})_l$ to $n_{il}((Q_{0i})_l)$. In other words, each component of the normalization map for the service class is the normalization map for the respective attribute. A set $C = \{C_1, \ldots, C_n\}$ denotes service classes. $n = |C|$ is the number of service classes.

4.3.1.1.3. Definition (Service) A *service* s_{ij} of class c_i is an *attribute vector* $Q_{0ij} \in D_i$, i.e. a service is defined by its attribute values. It is assumed that all the relevant properties of the service are given as such QoS attributes, so those fully define the service. Note that this is a vector of attribute values $(Q_{0ij})_l \in D_{il}$. This attribute vector maps under n_i to a *quality vector* $Q_{ij} = n_i(Q_{0ij}) \in [0,1]^{m_i}$.

The set $S_i = \{s_{i1}, \ldots, s_{ik}\}$ is defined as the set of services of class c_i. $k_i = |S_i|$ is the number of such services.

Let $S = S_1 \cup \ldots \cup S_n$ be the set of all services. Without loss of generality, It is assumed that $S_i \cap S_{i'} = \emptyset \ \forall i \neq i'$, i.e. each service belongs to exactly one service class. If there are polyvalent services which can fulfill different tasks, they have to be modeled as several different services, one for each task (which happen to be offered by the same provider). At each step of the workflow, at most one of those virtual services will be relevant (the one for the class needed at this step), therefore this is a valid abstraction to make. Let $c : S \to C$ be the function which maps each service s_{ij} to its class c_i and $k = k_1 + \ldots + k_n = |S|$ be the total number of services.

4.3.1.1.4. Definition (Abstract Workflow) An *abstract workflow* specifies the requirements the user has for the workflow. It is given by a directed graph whose nodes are the steps in the workflow, and for each node, the needed service class, minimum ("must") requirements for the attribute vector, desired ("should") requirements for the attribute vector and weights given to the desired requirements, indicating how much value is given to the "should" request. Mathematically, an abstract workflow can be defined as $W_0 = (V, E, f_0, R_{m0}, R_{d0}, w)$ where (V, E) is a directed graph and the other components are maps assigning values to each node in the graph: $f_0 : V \to C$, the *class selection* map, picks the desired service class for each node such that $\forall v \in V$:

$$f_0(v) = c_i \Rightarrow R_{m0}(v) \in D_i, R_{d0}(v) \in D_i, w(v) \in \mathbb{R}_+^{m_i},$$

where the vectors R_{m0}, R_{d0} and w are the minimum requirements, the desired requirements and the weights, respectively. As for the services, one can define normalized quality requirements R_m and R_d such that, $\forall v \in V$:

$$f_0(v) = c_i \Rightarrow R_m(v) = n_i(R_{m0}(v)), R_d(v) = n_i(R_{d0}(v))$$

and thus $\forall v \in V$:

$$f_0(v) = c_i \Rightarrow R_m(v) \in [0,1]^{m_i}, R_d(v) \in [0,1]^{m_i}.$$

We assume $\forall v \in V : R_d(v) \geq R_m(v)$, i.e. the desired requirements are always at least the minimum requirements.

4.3.1.1.5. Definition (Concrete Workflow) A *concrete workflow* specifies a concrete workflow instance which should match the user's requirements. A concrete workflow $W = (V, E, f)$ is defined as a directed graph (V, E) with a *service selection map* $f : V \to S$.

4.3.1.1.6. Definition (Sensible Workflow) W is *sensible* for $W_0 = (V, E, f_0, R_{m0}, R_{d0}, w)$ (note that V and E are the same for W and W_0, a workflow is never sensible for an abstract workflow on a different graph) if $f_0 = c \circ f$ (the service selection function is *compatible* with the class selection function), i.e. the service for each node is actually of the requested class for that node.

4.3.1.1.7. Definition (Feasible Workflow) W is *feasible* for W_0 if W is sensible for W_0 and $\forall v \in V : Q_{f(v)} \geq R_m(v)$ (componentwise, i.e.
$\forall v \in V : (Q_{f(v)})_1 \geq (R_m(v))_1, \ldots, (Q_{f(v)})_{m_{f_0(v)}} \geq (R_m(v))_{m_{f_0(v)}}$), i.e. if all the minimum requirements are satisfied.

4.3.1.1.8. Definition (Happiness Measure) A *happiness measure* quantifies how happy the user is with a given workflow considering his/her desired requirements and weights. For all pairs (W_0, W) where W is sensible for W_0, we define $h(W_0, W)$ as

$$h(W_0, W) = \sum_{v \in V} h_{v f_0(v)}$$

where for each choice of service s for v

$$h_{vs} = \sum_{l=1}^{m_{f_0(v)}} w_l(v) h_{vsl}$$

and

$$h_{vsl} = \begin{cases} 0, Q_{sl} < (R_m(v))_l \\ 1, Q_{sl} \geq (R_d(v))_l \\ \frac{Q_{sl} - (R_m(v))_l}{(R_d(v))_l - (R_m(v))_l}, \text{else} \end{cases},$$

i.e. 0 for infeasible qualities, 1 for qualities at least as high as desired and linearly increasing between the minimum and the desired requirement. Note that this is a linear happiness measure. We assume it makes sense to define such a linear happiness, which is a requirement on the normalization maps: they have to be such that linear combinations of the quality values (the normalized attribute values) are useful.

4.3.1.2. Mathematical Problem Statement

Now after having the required definitions, it is possible to state the problem in mathematical terms: Given an abstract workflow W_0, The goal is to find a concrete workflow W which optimizes:

$$\begin{array}{ll} \max & h(W_0, W) \\ \text{s.t.} & W \text{ is feasible for } W_0 \end{array}$$

4.3.1.3. Special Cases

There are some special cases for user requirements which are important to consider:

- The user has no minimum requirement for a given attribute: one can simply set $(R_m(v))_l$ to 0 for that attribute.
- The user wants to get an as high quality as possible from a given attribute: one can set $(R_d(v))_l$ to 1 for that attribute. A weight has to be set to prioritize that attribute compared to others.
- The user has no desired requirements beyond the minimum: one can set $(R_d(v))_l$ to $(R_m(v))_l$ and $w_l(v)$ to 0 for that attribute. (Other choices with the same effect are also possible: $(R_d(v))_l = (R_m(v))_l$ with any arbitrary $w_l(v)$ or $1 \geq (R_d(v))_l \geq (R_m(v))_l$ with $w_l(v) = 0$. In both cases, the resulting optimization problem is equivalent to the one with the canonical choice.)

4.3.1.4. Aggregate Constraints

One can choose any shared attribute of the services and can define a bound on it as a global constraint. These additional constraints are called *aggregate constraints*, because they are the constraints which aggregate the QoS parameters of the different services, whereas workflow feasibility considers each node individually. The most common aggregate constraints are the overall cost or the overall response time of the generated workflow. The proposed approach can also handle other similar constraints. Here the total cost, total time and total reputation are chosen as to demonstrate how to formalize user defined global requirements.

4.3.1.4.1. Maximum Cost Requirements: The cost can be modeled as a QoS parameter a_{i1} with decreasing normalization function n_{i1}. A cap K on the total cost of all services

can be added as an additional constraint, which is linear over the unscaled attribute values:

$$\sum_{v \in V} \left(Q_{0f(v)} \right)_1 \leq K.$$

4.3.1.4.2. Maximum Time Requirements: Likewise, the execution time of the entire workflow can be modeled as a QoS parameter a_{i2} with decreasing n_{i2}, yielding a constraint:

$$\sum_{v \in V} \left(Q_{0f(v)} \right)_2 \leq T.$$

Assuming linear execution of the workflow, this is a cap T on the total execution time of the workflow. In general, the sum on the left hand side will be an upper bound for the execution time and the actual execution time will still be bounded by T, but the constraint may be overly restrictive.

4.3.1.4.3. Minimum Reputation Requirements: One can model the sum of the reputation of all services as a QoS parameter a_{i3} with increasing normalization function n_{i3}. A cap R on the total reputation of all services can be added as an additional constraint, which is linear over the unscaled attribute values:

$$\sum_{v \in V} \left(Q_{0f(v)} \right)_3 \geq R.$$

This approach can also handle other similar constraints.

It can be easily seen that with these added constraints, after filtering out the services which do not satisfy the minimum requirements, our problem becomes equivalent to a Multidimensional Multi-choice Knapsack Problem (MMKP): the utilities in the MMKP are our happiness values h_{vs}. This formal model allows for a subsequent efficient and effective description of the branch and bound solution approach. This clear and unambiguous description even builds a general formal basis for further research as well in this problem domain.

4.3.2. Algorithms for Service Selection

In this section, to realize the motivational scenario, the proposed algorithms are distributed in two phases: a pre-computation phase in which we aim at the QoS-aware optimization of service composition and an updating phase using heuristics to react to dynamic changes in user requirements or reuse solutions for users with similar requirements.

For the pre-computation phase, a branch and bound algorithm is used, which is one of the most successful, flexible and easy to parallelize algorithms for optimization problems. The drawback of the branch and bound algorithm is that it is very expensive. Thus, in order to react to dynamically changing conditions such as service failures or changing user requirements, a heuristic-based strategy is employed to reuse the existing solution to the

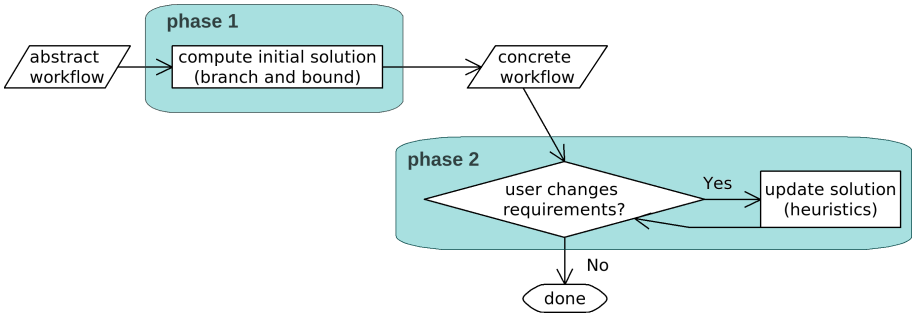


Figure 4.3.: The two phases of the optimization algorithm

original problem as a starting point and update it for the changed requirements. These two algorithms are explained in detail as follows.

4.3.2.1. A Parallel Branch and Bound Algorithm for Optimized Service Selection

In this section, an algorithm to solve the optimization problem for QoS-aware workflow composition is presented in the motivational scenario while focusing on the ease of parallelization, which allows a distributed implementation exploiting the power of a heterogeneous multiprocessing environment without shared memory, such as the Grid. It will then be explained how the results of the optimization can be reused even under dynamic changes to the user's requirements or the availability and characteristics of the offered services.

4.3.2.1.1. Overview and Design Considerations The branch and bound algorithm is one of the most successful, flexible and easy to parallelize algorithms for optimization problems. Branch and bound algorithms work over a tree of nodes and each node in the tree may be sent to a different computing node in the Grid for solution. A node is solved when either it can be pruned through bounding or all of its children are solved. The problem is solved when the root node of the tree is solved.

Our solutions will be represented as *decision vectors* $d = \{d_1, \ldots, d_p\}$, each entry d_v of which corresponds to the choice of a service for the node v. Partial solutions will be represented as partial decision vectors, in which some components are left unassigned.

The presented algorithm requires some input data, which has to be globally accessible to all nodes, but remains constant throughout the execution of the algorithm. It is composed of a vector $V = \{v_1, \ldots, v_p\}$ of nodes in the workflow graph and a vector S of vectors $S_i = \{s_{i1}, \ldots, s_{ik_i}\}$ of services (one vector S_i for each service class). The entries in the vectors are simple structures: a node has an integer f_0, the *service class* required for the node in the abstract workflow and vectors R_m, R_d, w, the *normalized requirements* and *weights* for each attribute of the service class; a service has vectors Q, its *normalized QoS parameters* and Q_0, its *actual QoS parameters* (used for the aggregate constraints).

The amount of variable global data is minimized to ease parallelization. Only one global variable is required i.e. a floating-point scalar `lbound`, which represents the lower bound for

the maximal happiness at the current state of our branch and bound algorithm. Each time a feasible solution is found, its happiness value is a lower bound for the optimal happiness.

The upper bounds are computed locally for each subproblem. They are upper bounds for the highest achievable happiness without changing the decisions already made in this subproblem. The Multidimensional Multi-choice Knapsack Problem is structured very similarly to a regular multidimensional knapsack problem. Therefore, upper bounds for the proposed branch and bound algorithm are similarly easy to compute: for each node, we pick the optimum service from the desired class, without taking the aggregate constraints into account. Dropping those constraints leads to a *relaxation* of the original problem. The happiness value for its solution is thus an upper bound for the maximal happiness for the original problem. The optimum service is thereby defined as the service which maximizes the happiness measure defined in the model (while satisfying the minimum quality requirements). Concretely, given a partial decision vector d, the upper bound can be computed as follows: For each node v which is not set in d, it is required pick the service s which maximizes h_{vs}, the happiness value h_v when s is chosen for v. For the nodes with an already fixed $d_v = s_0$, we pick $s = s_0$ and compute h_{vs_0}. The upper bound is then the sum of all the happiness values h_{vs} computed.

As a side effect, a complete decision vector is obtained corresponding to the upper bound (the vector composed of all the s picked). It will correspond to a concrete workflow which is feasible when not taking the aggregate constraints into account, but may violate those constraints.

4.3.2.1.2. Main algorithm Each node in the branch and bound decision tree is an instance of the main program. Every such instance has two parameters: the partial decision vector d, which corresponds to the fixed decisions for the subproblem being considered, and its parent node in the decision tree. A node is solved when either it can be pruned through bounding or all of its children are solved. The process starts with a single instance, the root instance, which gets a completely unassigned partial decision vector and the full problem as its parent (i.e. the problem is solved when the root node is solved). Nodes of the problem can be individually assigned to computation nodes in the Grid. Once the node did its main computations, it branches into child nodes, which are spawned as separate nodes, and blocks waiting for messages. Thus the computing node in the Grid is free to handle one or more of the resulting child problems.

Each node of the decision tree runs the same function, described by algorithms 1 and 2.

```
   Input: partial decision vector d, node parent
   /* first, we bound, in an attempt to prune the node                    */
1  (u, d') ← find_upper_bound(d);
2  if u ≤ lbound then
   |  /* cannot improve the optimum by more than tol, prune              */
3  |  message(parent, no solution);
4  else
5  |  if the upper bound is actually a feasible solution, i.e. the aggregate constraints are
   |  satisfied then
   |  |  // we found a solution
6  |  |  message(parent, solution found, u, d');
7  |  |  if u ≤ lbound then
8  |  |  |  lbound ← u;
9  |  |  |  broadcast(new lower bound, u);
10 |  |  end
11 |  else
   |  |  // branch
12 |  |  pick any node v with d[v] unassigned;
13 |  |  i ← f_0(v); // service class for v
14 |  |  if S[i] is empty then
15 |  |  |  message(parent, no solution);
16 |  |  |  stop;
17 |  |  end
18 |  |  foreach s in S[i] do
19 |  |  |  d' ← d;
20 |  |  |  d'[v] ← s;
21 |  |  |  spawn decision tree node for d', this node;
22 |  |  end
   |  |  // initialize variables
23 |  |  best ← −∞;
24 |  |  solved ← false;
   |  |  /* now wait for the child nodes to complete                       */
25 |  |  enter event loop waiting for messages (alg. 2)
26 |  end
27 end
```

Algorithm 1: Node in the branch and bound decision tree

```
1  on new lower bound (l) do
       /* updated lower bound, check again if we can prune - we have to be
          careful here:  we cannot prune this node if the new lower bound
          comes from one of our own subproblems (descendants in the tree)  */
2      if l ≥ u and not from descendant of this node then
3          message(parent, no solution);
4          stop;
5      end
6  end
7  on no solution do
8      solved ← solved + 1;
       // check if all children are solved
9      if solved = length(S[i]) then
           /* check if we found a solution which is better than the current
              upper bound                                                    */
10         if best > −∞ and best ≥ lbound then
11             message(parent, solution found, best, dbest);
12         else
13             message(parent, no solution);
14         end
15         stop;
16     end
17 end
18 on solution found (sol, dsol) do
19     solved ← solved + 1;
       // update the current best solution
20     if sol > best then
21         best ← sol;
22         dbest ← dsol;
23     end
24     check if all children are solved and proceed as above
25 end
```

Algorithm 2: Event loop for a decision tree node

4.3.2.1.3. Feasibility Test for Pruning One issue with the above algorithm is that it only prunes when it finds a feasible solution. (In this context, *feasible* means that the workflow is feasible, i.e. the quality requirements are fulfilled, and the aggregate constraints are satisfied.) This means that for infeasible or near-infeasible problems, a lot of branching is needed. The solution to this is to prune clearly infeasible problems too.

This is achieved by computing lower bounds for the aggregate constraints, which can be done in a similar way as computing the upper bound for the happiness. The only difference is that in this case we ignore the happiness and pick the cheapest resp. fastest service for

each undecided node. (We can again ignore the services which do not satisfy the minimum quality requirements, as those will never be part of a feasible solution.) We can then prune the node if the computed lower bound already exceeds the cap K resp. T. This is called a *feasibility test*.

4.3.2.2. A Heuristic Algorithm for Optimization of Service Selection

After the pre-computation phase in which the goal was the QoS-aware optimization of service composition an updating phase using heuristics is presented to react to dynamic changes in user requirements or reuse solutions for users with similar requirements.

The drawback of the branch and bound approach is that it is expensive. Its worst-case performance can be proven to be exponential. Its practical performance is highly dependent on input data, and in the case of a threaded or distributed implementation, also non-deterministic. (The algorithm is non-deterministic due to thread or process scheduling, in the distributed case also timing of network communication. Only the resulting happiness is deterministic, as the algorithm is guaranteed to find the exact optimum.) Unfortunately, this is not merely an issue with the implementation: the problem is NP-hard, due to the total cost and time constraints, which make it equivalent to a two-dimensional knapsack problem. One of the ways to deal with this scalability problem is to parallelize the implementation. As explained above, in order to efficiently react to changed user requirements, a different approach is needed, based on heuristics. The proposed solution is to reuse the existing solution to the original problem as a starting point and update it for the changed requirements. This can be done efficiently (in low-order polynomial time) and generally results in a near-optimal solution to the modified problem. However, this is only a heuristic: the optimality of the obtained solution cannot be guaranteed. Only recomputing everything, which is NP-hard (as discussed above), can guarantee that.

Algorithm 3 describes the approach used for the total cost constraint. Exactly the same procedure is also used for the total time constraint, and in principle similar updates could also be done for other changes in user requirements or service offerings, e.g.:

- changes in user's quality requirements or weights
- changes in service parameters
- added / removed services.

As this procedure is very efficient, it is also possible to use it for runtime changes, e.g. service failure. In this case, one has to consider the structure of the workflow, as it does not make sense to change a service which already completed or with which a non-refundable SLA has already been agreed to. Thus, one would have to replace the failed service with another service, then make up for cost or time overruns, if any, by replacing the services which are not fixed yet by cheaper resp. faster services.

The proposed solution is to reuse the existing solution to the original problem as a starting point and update it for the changed requirements. This can be done efficiently (in low-order polynomial time) and generally results in a near-optimal solution to the modified

problem. However, this is only a heuristic: the optimality of the obtained solution cannot be guaranteed. Only recomputing everything, which is NP-hard (as discussed above), can guarantee that. The heuristics are based on the quotient $q = \frac{h_s - \text{old } h_s}{cost - \text{old } cost}$, where h_s is the happiness with the service s and *cost* its cost. The old values are the ones for the service which was used for the given node in the old solution. When looking for a cheaper service, i.e. in the case of cost overruns, service are looked for where q is as large as possible, indicating that the user saves the most money while losing as little quality as possible. If on the other hand one has money left and intends using it to improve the quality, one looks for the largest values of q instead, meaning that we get the most bang for our buck. Exactly the same procedure can be used for the total time constraint.

As this procedure is very efficient, it can be used even for runtime changes, e.g. service failure. In this case, one has to consider the structure of the workflow, as it does not make sense to change a service which e.g. already completed. Thus, one would have to replace the failed service with another service, then make up for cost or time overruns, if any, by replacing the services which are not fixed yet by cheaper resp. faster services.

However, this heuristic approach still has a need for an initial solution (in this case, the branch and bound optimization process), the computation of which takes up the bulk of the time.

The heuristic update can also be employed to reuse a solution designed for a given client for a new client with similar requirements. This case can be treated just as if the original client changed their requirements.

Another possibility worth trying would be to avoid the branch and bound procedure entirely and rely only on the updating heuristics:

1. A solution is computed without the constraints on K and T. This can be done in polynomial time. We use this solution as our starting solution.

2. That solution is updated heuristically to honor K and T.

```
1  Compute cost of old solution;
2  if K > old K then
3  |  foreach node do
4  |  |  Compute happiness for currently chosen service;
5  |  |  foreach service satisfying minimum requirements, happiness hs > old hs and
   |  |  cost > old cost do
6  |  |  |  Compute q = (hs − old hs)/(cost − old cost);
7  |  |  end
8  |  |  Add best (largest) q to priority queue;
9  |  end
10 |  while queue not empty do
11 |  |  Pick top queue entry;
12 |  |  if new cost ≤ K then apply update;
13 |  end
14 |  Check solution;
15 |  if infeasible then return FAILURE;
16 |  Output updated solution;
17 else if cost > K then
18 |  foreach node do
19 |  |  Compute happiness for currently chosen service;
20 |  |  foreach service satisfying minimum requirements, happiness hs < old hs and
   |  |  cost < old cost do
21 |  |  |  Compute q = (hs − old hs)/(cost − old cost);
22 |  |  end
23 |  |  Add best (smallest) q to priority queue;
24 |  end
25 |  while queue not empty and cost > K do
26 |  |  Pick top queue entry;
27 |  |  Apply update;
28 |  end
29 |  Check solution;
30 |  if infeasible then return FAILURE;
31 |  Output updated solution;
32 else
33 |  Output old solution;
34 end
35 return SUCCESS;
```

Algorithm 3: Heuristic update for K

Algorithm 3 outlines the proposed heuristic based algorithm. Figure 4.4 represents the same algorithm in form a flow chart.

This procedure would be significantly faster than branch and bound, but the drawback

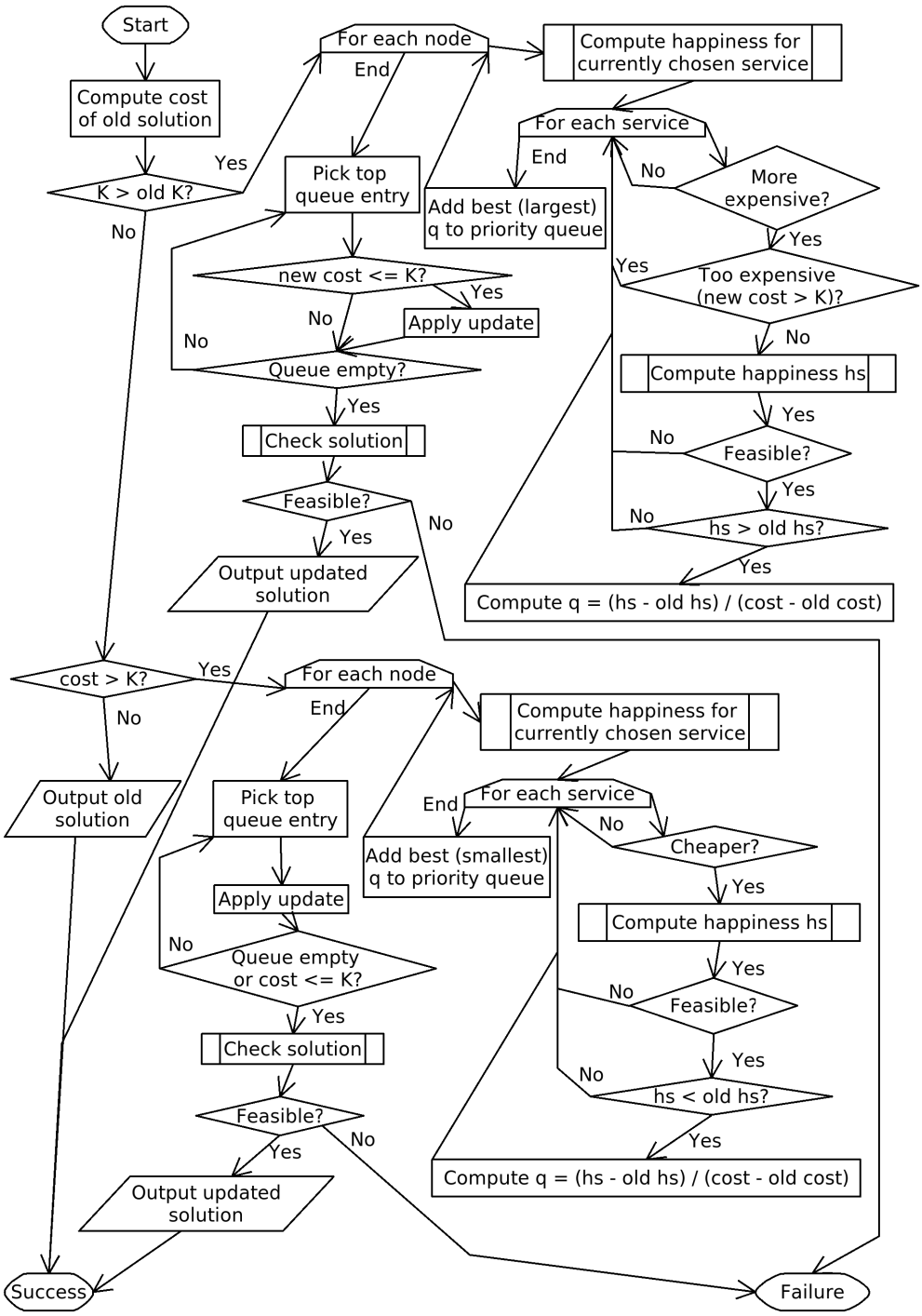


Figure 4.4.: Flow Chart for the algorithm based on updating heuristics

is that it would no longer be guaranteed to find an optimal solution. Another issue is that the current heuristics may fail entirely, because adjusting for the new value of one constraint can violate the other. Thus, a *failsafe* version of algorithm 3 was implemented, which considers only those alternative services which do not take more time when adjusting for cost and vice-versa. Algorithm 3 is tried first, then if it fails, the failsafe version. When adjusting for an increased constraint, this will always lead to a feasible solution which is at least as good as the initial one. For a decreased constraint, it can still fail, in which case our implementation falls back to recomputing a new solution using branch and bound. A completely failsafe approach to maintaining feasibility is NP-hard, just like the original problem, because it amounts to minimizing one constraint while satisfying the other, which is equivalent to a knapsack problem.

4.4. Running Example – Service Value Chains

In the motivational scenario it was stated that the automated workflow service exists independently of the Cloud environment. Considering the workflow tool also as a Cloud service brings about very interesting possibilities. Lets say that the user is directly connected to the workflow service and the workflow service further extends to a media service and a computational service forming a hierarchy. After the user chalks out these activities in an automated workflow tool, a set of appropriate cloud services are searched and mapped on the workflow activities. This has been depicted in Figure 4.5. From Arfa's view-point, all her tasks are done by only two services, i.e. the "rendering workflow" service and the "hosting service" identified as a "Platform as a Service" (PaaS) and a "Infrastructure as a Service" (IaaS) respectively by her Cloud-based service providers. The rendering workflow service allows Arfa to define a series of activities involving video rendering, compression etc. and promises to take care of these tasks single-handedly. Afterwards she only requires to use a hosting service to host the final compressed video on a dedicated server to quickly share the output of her work among colleagues. What Arfa's view does not cover is the fact that both of these services are themselves the result of an aggregation of even more basic services thus extending a supply chain type of structure beneath them. The rendering workflow subdivides into services such as the "media engine" and the "computing infrastructure" provided by different service providers in a public Cloud. On further investigation it may be revealed that the "media engine" is composed of even more basic services such as "the Physics engine", "the sound engine", and the computing Infrastructure too resells different qualities of computing and storage services with varying response times and calculation speeds thus the list goes on.

Several service providers while competing with each other to fulfill the advertised user requirements can also use the same selection algorithm to form a composite service by combining several basic service in order to present their solution to the client. A hierarchy of services is likely to emerge as a result of this divide and conquer process. At each level in the hierarchy the user requirements, the happiness criteria and weights are redefined for the new client. Every service in the chain is a client for the services immediately below

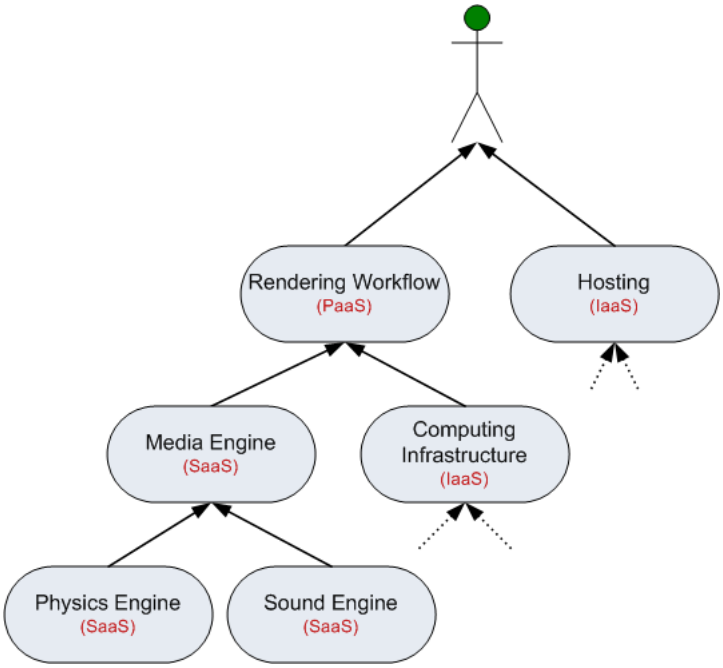


Figure 4.5.: Service Value Chains

itself, for instance, the workflow rendering service acts as a client for the media service and the computational infrastructure service. Thus the original problem is solved in a divide and conquer manner by services finding other services to solve those parts of the problems which are beyond the scope of their expertise.

4.5. SLA Negotiation of Configurable Services

After the short-listing of services, SLAs are required to be established. There could be a possibility that even some of the best fitting services do not fully comply to the user requirements and some parameters are slightly out of match. This problem can be resolved by contacting the service provider and negotiating about fine tuning the mismatching attributes. There is a strong need for dynamic and flexible negotiation mechanisms, which allow service providers to dynamically compute customizable service configurations against consumer specifications following the business policies of the service provider at the same time. For this, service providers and consumers express their demands and offers in an SLA template, which is finalized into a binding contract after both parties come to an agreement. An SLA template initiated by either a service provider or a consumer may pass through several rounds of negotiation before becoming a legal contract. The process of negotiation facilitates both the consumer and the service provider to sharply refine and finalize their expectations, demands and liabilities in accordance with the available resources. The interests of the client may go beyond cheap price and high quality of services and include

preferences demanding strict specifications in case of certain properties and relaxation for the others. For instance, a client may be very strict with the output resolution of an image processing service but may not care about the throughput of the service for a batch job. The service provider on the other hand, would make an utmost effort to find some ways to match the client's requirements while protecting its business rules and thus not risking the overall profit margin and deliverable QoS (Quality of Service) levels. For this purpose, the service provider must be able to configure services dynamically, in accordance with the client's preferences and compliant to the business rules. The resultant configurations may not exactly match the clients' requirements but would reflect the best that the service provider could offer. This may lead to another round of negotiation if the client slightly modifies its requirements or preferences in order to get a better quotation.

4.5.1. Dynamic Configuration of SLA Offers

In this section, it is explained how the service provider can customize service configurations dynamically in response to the client's requirements and priorities. It is assumed that both the service provider and the service consumer are able to express their requirements in their respective SLA templates and any of them can initiate the negotiation process by sending its SLA offer to the other. For the sake of the argument, let's say that the client initiates the process.

4.5.1.1. Service Consumer's Role

For the service provider to better understand the consumer's exact requirements and to reciprocate with its best offer, the consumer should be able to express its requirements precisely along with their priorities. This will allow the service provider more flexibility to come up with the cheapest and most desirable offer possible for the client. The client can express its requirements expressing the desired values of service attributes and assigning weights to them to highlight its priorities.

4.5.1.2. Service Provider's Role

The service provider is required to compute a configuration of the service fulfilling the client's requirements in accordance with its business rules, compute the corresponding price and respond to the client with its counteroffer. The counteroffer need not contain the exact configuration that the client required but the closest possible that the service provider can offer. The client, on examining this offer, can redefine certain values or weights of its requirements in order to expect a better offer.

4.5.1.3. Negotiation and Renegotiation

The negotiation round will go on until both parties agree on certain terms. In the next section, these concepts are formulated in a formal model that will serve as a basis for computing service configurations as part of a dynamic and flexible SLA negotiation protocol.

A similar communication pattern can be followed for a renegotiation round. In case of renegotiation the previously established SLA will remain intact even in case of a failure of the process whereas in case of negotiation an SLA does not exist before and in fact is the output of the process.

4.5.2. Formal Model for Negotiation and Renegotiation of Configurable Services

This section formalizes the concept of dynamic service configurations based on the client requirements and preferences. These service configurations will be presented to the clients in the form of SLA offers. A service is defined through its attributes and a service configuration as a set of specific values assigned to the service attributes. For the definitions of service attribute, service value and service, please refer to section 4.3.1.1. Please notice that it is not required to define service classes in order to define service configuration.

4.5.2.1. Definition (Configuration)

A *configuration* of the service s_i is an *attribute vector* $Q_{0ij} \in D_i$, i.e. a vector of specific attribute values for the attributed of a service. It is assumed that all the relevant properties of the service are given as such QoS attributes, so those fully define the service. Note that this is a vector of attribute values $(Q_{0ij})_l \in D_{il}$. This attribute vector maps under n_i to a *quality vector* $Q_{ij} = n_i(Q_{0ij}) \in [0,1]^m$.

4.5.2.2. Definition (Set of Feasible Configurations)

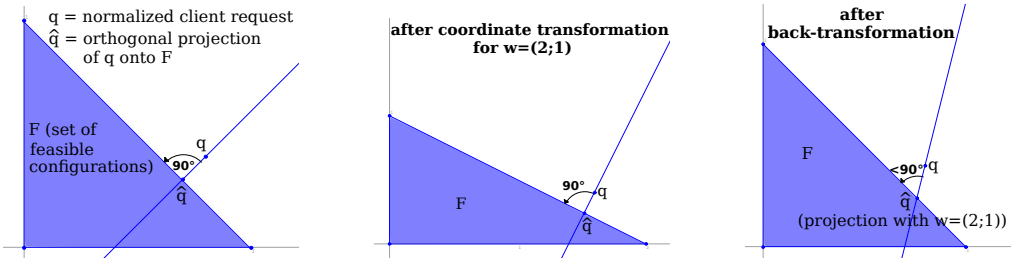
For each service, it is assumed that only a subset F of the set $D_i = D_{i1} \times \ldots \times D_{im}$ of all possible configurations can actually be fulfilled by the service provider. F is called the *set of feasible configurations*. An attribute value q_0 will be called *feasible* if and only if $q_0 \in F$, *infeasible* otherwise. The exact nature of F will in general only be known to the service provider, not to the client.

4.5.2.3. Definition (Price Function)

Each service has a given *price function* $f : D_i \to \mathbb{R}_+$ which maps each feasible attribute value q_0 to its monetary cost $f(q_0)$. We set $f(q_0) = \infty$ for infeasible q_0. This price function will also usually only be known to the service provider.

4.5.2.4. Definition (Weights)

A vector $w \in \mathbb{R}_+^m$, where m is again the number of attributes of a given service s, will be called a vector of *weights* corresponding to the service s. During the renegotiation process, it allows the client to define which attribute values carry most importance to him, which influences the service provider's idea of the closest feasible point.



(a) For the trivial weights $w = (1; 1)$, g corresponds to an orthogonal projection.

(b) Nontrivial weights w correspond to a coordinate stretch by factors w.

(c) The effects of the coordinate transformation in the original coordinates.

Figure 4.6.: Geometric interpretation of the negotiation function g and the distance d_w

4.5.2.5. Definition (Negotiation Function)

If the client requests an infeasible configuration q_0, the service provider computes the closest feasible configuration $\hat{q}_0 = g(q_0, w)$ using a *negotiation function* defined as follows:

$$g(q_0, w) = \begin{array}{l} \arg\min_{\hat{q}_0} d_w(\hat{q}_0, q_0) \\ \text{s.t. } \hat{q}_0 \in F, \end{array}$$

i.e. the $\hat{q}_0$ in the set F of feasible configurations which minimizes $d_w(\hat{q}_0, q_0)$, where

$$d_w(\hat{q}_0, q_0) = \|n(\hat{q}_0) - n(q_0)\|_w$$

and $\|u\|_w = \sqrt{\sum_{i=1}^{m} w_i^2 u_i^2}$ is the 2-norm weighted by w.

If we write $q = n(q_0)$ and $\hat{q} = n(\hat{q}_0)$, d_w can be written as

$$d_w(\hat{q}_0, q_0) = \sqrt{\sum_{i=1}^{m} w_i^2 (\hat{q}_i - q_i)^2}.$$

Figure 4.6 shows a geometric interpretation of the weighted 2-norm distance d_w defined above in an example with a 2-dimensional, triangular set of feasible configurations F. In the absence of weights, the 2-norm is the Euclidean norm and the closest point under the 2-norm is given by an orthogonal projection. Setting weights w corresponds to stretching, for all i, the i^{th} coordinate axis by a factor w_i (the i^{th} coordinate of w). This deforms the orthogonal projection, yielding a point which deviates less in the coordinates weighted higher at the expense of those weighted lower. An analogous geometric interpretation is possible in higher dimensions.

4.5.3. Running Example – Negotiation

After successfully building a service-based rendering and hosting application the user in the running example realizes that she also needs a permanent long term storage service to

Data Storage Service
(Client's Preferences)

Service Attribute	Desired Value (qo)	Weight (w)
Availability	99.9 %	1
Bandwidth	10 Mbps	0.8
Disk Size	5 GB	0.5
Compression	2	0.5
Degree of Replication	2	1
Reputation	9	0.9

(a)

Data Storage Service
(Rendering Workflow's Options)

Availability	BandWidth	Disk Size	Compression	Degree of Replication
99.9 %	8 Mbps	2 GB	1	1
_	15 Mbps	4 GB	2	2
_	20 Mbps	8 GB	4	3
_	30 Mbps	16 GB	_	4

(b)

Figure 4.7.: (a) Client's Preferences, (b) Service Provider's Options

store her data resulted from the rendering process executed from time to time. From her point of view, there are several important attributes that a storage service should have.

Higher bandwidth results in a shorter response time, which she considers as the major performance measurement for the data store. The user has the following requirements for the data storage service.

1. The minimum requirement for the bandwidth to access the data is 10 Mbps.
2. Due to parallel access, the available disk size may change dynamically, but the disk size at the storing location always has to be at least 5 GB.
3. For the application characteristics of the running example in focus, a high compression rate is desired.
4. The data needs to be replicated to at least one extra location.
5. A very high level (e.g. 99.9 percent) of availability of the service is desired.
6. A very high reputation i.e 9 out of 10 is desired.

Following the notions of service attributes, their values and their weights as defined in the formal model for service selection, the client's requirements and preferences have been formulated in Figure 4.7(a). Out of many competitors, there are two services, which are the closest in fulfilling the client's requirements but on the basis of the reputation, the happiness measure results in favor of the storage service offered by an already contracted service. Rendering workfow, which is a composite service, also offers storage, which is supplied to it by the computation infrastructure. This has been depicted in Figure 4.8. It must be noted that no service provider was in a position to fulfill all the preferences of the client. But they used the priorities of the client expressed in terms of weights and

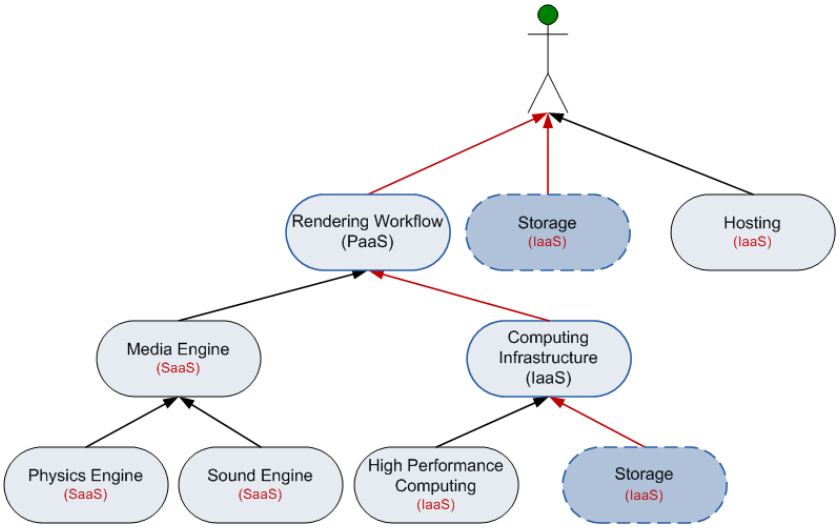


Figure 4.8.: The storage service is available from two service providers: by an independent provider with low reputation and through "rendering workflow", which has higher reputation

computed the most suitable configuration closest to the client's requirements following the negotiation function:

$$g(q_0, w) = \begin{array}{l} \arg\min_{\hat{q}_0} d_w(\hat{q}_0, q_0) \\ \text{s.t. } \hat{q}_0 \in F. \end{array}$$

So instead of the bandwidth of 10 Mbps and the desired diskspace of 5 GB, the rendering workflow offers the client a bandwidth of 8 Mbps and diskspace of 4 GB, which are the closest available values to the ones the client requested. The client can either accept this offer or can opt for further negotiation by modifying its requirements.

4.5.4. An SLA Negotiation/Renegotiation Protocol for Configurable Services

The focus of this section is to highlight the requirements of a flexible model for negotiation between the service provider and the service consumer leading to the establishment of an SLA between them. It is time to explain the step by step detail of the negotiation process based on the dynamic configuration of services as depicted in Figure 4.9.

1. Initiation of the Negotiation Process
   Any party can initiate the negotiation process. However, this is not a symmetric protocol because the real world is not symmetric. Both the service consumer and the service provider need to maximize their interests so their activities within their scopes vary from each other. In Figure 4.9, it is assumed that the client first gets the SLA template and fills in its preferences.

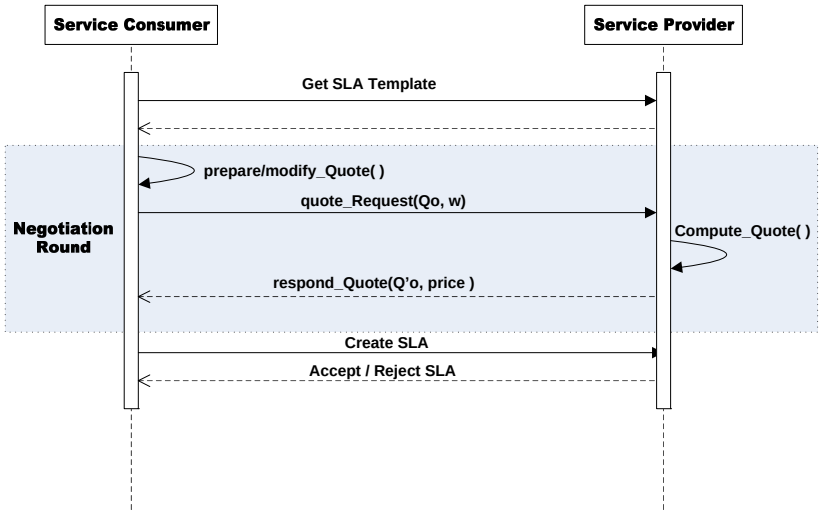


Figure 4.9.: Negotiation Protocol for SLA configuration

2. Preparation of SLA quotation by the service consumer
   The client provides two types of information to the service provider. It fills in the desired values of service attributes within the SLA template, and it also informs about its priorities regarding those attributes. This information can either be a part of the SLA template or can be sent separately. The idea is to give clues to the service provider about where adjustments can be tolerated and which attributes are a must-have requirement. According to the scenario, the service consumer will send the values shown in the Figure 4.7(a).

3. Computation of the best configuration offer by the service provider
   The service provider, following the difference function described in the formal model, computes a service configuration which is closest to the desired service configuration. It also computes the price using the price function described in section 4.5.2. As depicted in Figure 4.7(b), it is quite possible that no configuration exists that matches the service consumer's preferences exactly. In that case, during the configuration selection, a relaxation is assumed on the attributes with the least priority. The exact computational criteria have been described in the formal model. The service provider in our scenario will offer a bandwidth of 8 Mbps and a disk size of 4 GB while fulfilling the rest of the requirements of the client.

4. Analysis of the offer and modification of service preferences by the consumer
   After receiving the best possible configuration matching the consumer's request, the service consumer analyzes the offered configuration and can opt to proceed in three different ways, i.e., accept, reject or further negotiate the offer. The client can decide to further negotiate the offer either by changing/modifying certain attribute values or by relaxing certain priorities (changing weights). In case of a modified quote, the

negotiation process keeps on going until both parties agree or disagree to continue it further.

5. SLA establishment
   If the client agrees with the SLA offer of the service provider, it can opt to commit and send an acceptance call thus binding itself to the agreement. If the service provider also accepts then a contract is formed and an SLA is formally established. Conversely, if either of parties reject the SLA offer then the negotiation round is failed.

6. Renegotiation
   The same process can also be utilized for renegotiation. In case of a successful renegotiation process, the newly formed SLA takes the place of the old one, otherwise the previous SLA survives and remains intact.

4.6. Summary

This chapter highlights the role of QoS parameters of services during service selection and service negotiation. QoS parameters are directly related to SLAs as on the basis of QoS attributes, Service Level Objectives and guarantees are defined in SLAs. The proposed formal model and algorithms for service selection can be implemented in any Utility Computing infrastructure. The service selection algorithms can work equally good when service providers publish services in form of SLA templates. The negotiation algorithm is based on a very generic model and can be customized in accordance with special requirements and special conditions. The service selection as well as negotiation/renegotiation model play a crucial role in case of service failures within service choreography. The reputation of a service is an important service parameter, which can become a very significant factor in service selection. These situations will be discussed in Chapter 6.

Chapter 5.

Hierarchical Aggregation of SLA Choreography

> Though the hassle of the sea gives to none security, in the secret of the shell, self preserving we may dwell
>
> *(Allama Muhammad Iqbal)*

This chapter presents a formalized approach based on the concept of SLA Views and adherent to WS-Agreement standard, to automate the aggregation process of hierarchical SLAs in SLA Choreography. The overall contribution of the chapter consists of:

- a privacy model based on the concept of SLA-Views,
- a formal description of hierarchical SLA-Choreographies based on SLA-Views,
- a formal model for SLA aggregation in hierarchal SLA-Choreographies,
- a set of aggregation patterns applicable at different level and
- the customization of WS-Agreement to highlight the realization of hierarchical SLA aggregation model.

5.1. Background

Service composition directly implies the need of composition of their corresponding SLAs. So far, SLA composition has been considered as a single layer process [31]. This single layer SLA composition model is insufficient to describe supply-chain based business networks. In a supply-chain, a service provider may have sub-contractors and some of those sub-contractors may have further sub-contractors making a hierarchical structure. This supply-chain network across various Virtual Organizations may emerge as a Business Value Network. Business Value Networks [10] are ways in which organizations interact with each other forming complex chains including multiple providers/administrative domains in order to drive increased business value. NESSI (Networked European Software and Services Initiative), which is a consortium of over 300 ICT industrial partners, has highlighted the importance of Business Value Networks [10] as a viable business model in the emerging

service oriented ICT infrastructures. In addition to the notion of Business Value Networks, NESSI has pointed out various other possibilities for similar inter-organizational business models; Hierarchical Enterprizes, Extended Enterprizes, Dynamic Outsourcing, and Mergers to name a few. The process of SLA aggregation in such enterprizes is a hierarchical process. Research community has just started taking notice [22] of the importance to describe this hierarchical aggregation. To enable these supply-chain networks as Service Oriented Infrastructures (SOI), the case of the Service Level Agreements needs to be elaborated and its issues resolved. SLA@SOI [22] is a European project that focusses on SLA issues in SOI. On its agenda is the provision of such Service aggregators, that offer composed services, manageable according to higher-level customer needs. In SLA@SOI's vision, service customers are empowered to precisely specify and negotiate the actual service level according to which they buy a certain service. Although SLA@SOI discusses the importance of service chains but it does not highlight their relevance in terms of value multiplication. In novel eBusiness platforms (such as Girds and Clouds) SLA is essentially important for the service consumer as it compensates consumer's high dependency on the service provider. With the advent On-demand service infrastructure, there is a high potential for third party solution providers such as Composite Service Providers (CSP), aggregators or resellers [88][36] to tie together services from different external service providers to fulfill the pay-per-use demands of their customers. A cumulative contribution of such Composite Service Providers will emerge as service value chains.

It is not sensible to expose the complete information of SLAs across the whole chain of services to all the stakeholders. Not only because of the privacy concerns of the business partners, but also disclosing it could endanger the business processes creating added value. To achieve this balance between trust and security, the concept of SLA-Views has been introduced. The inspiration for this concept comes from the notion of business process views [48][52] and workflow views [53]. Each business partner will have its own view comprising of its local SLA information. The holistic effect of these views will emerge as the overall SLA-Choreography.

5.2. SLA Choreography

In service value chains, services corresponding to different partners are aggregated in a producer-consumer manner resulting in hierarchical structures of added value. Service Level Agreements (SLAs) guarantee the expected quality of service (QoS) to different stakeholders at various levels in this hierarchy. This in turn leads to a hierarchical structure of SLAs that may span across several Virtual Organizations (VOs) with no centralized authority. In this research it is termed as Hierarchical SLA Choreography or simply *SLA Choreography*, in accordance with the underlying Service Choreography. A formal model of SLA Choreography is required not only for a better understanding of the problem but also to provide a comprehensive platform for computation design and implementation of the system. It is also required to devise formal functions describing the hierarchical aggregation of SLAs. At the same time the formal model must be in compliance with the

Figure 5.1.: structure of an agreement in accordance with WS-Agreement specification

WS-Agreement standard. In the formal model, the concept of SLA-views is introduced. The inspiration for this concept comes from the notion of business process views [52]. Each business partner has its own view comprising of its local SLA information. The aggregated effect of these views emerges as the overall SLA orchestration. From a service provider's point of view, it is not possible to expose the complete information of SLAs spanning across the whole chain of services to the consumer. Not only it does not make sense to reveal the information of a business partner's sub-contractors but it will also endanger business processes creating added value. With the help of SLA Views, the SLA information pertaining to different providers is veiled at various levels in the SLA orchestration. At the same time, the partners of a Business Value Network need to share their resources on the basis of mutual trust. Such a balance between trust and privacy of the stake holders requires a distributed trust model. Some of the direct implications of this distributed trust may be realized during the validation, the fault tolerance and the renegotiation processes.

5.3. Formal Model of SLA Choreography and its Aggregation

5.3.1. SLA and SLA Choreography

A service level agreement is a contract that defines mutual understandings and expectations regarding a service between the service provider and the service consumer. WS-Agreement [56], a standardized SLA language from OGF (Open Grid Forum) [12], defines the structure of agreement as depicted in Figure 5.1. The contract should bear an official name. Agreement Context contains information about the initiator, the responder and the provider of the agreement; expiration time of the agreement; and its template Id. Service Terms define the functional attributes of the agreement whereas the Guarantee Terms contain the non functional attributes. Guarantee Terms further describe the conditions, Service Level Objectives (SLO) and Business Value List (BVL) related to the agreement. Business Value List may express the importance of meeting an objective as well as information regarding penalty or reward.

Referring to Figure 5.1, the Service Terms, and Guarantee Terms as part of the encap-

sulating section Terms can be formally defined as under:
Definition 1 (Service Term). A service term denoted by $term_s$ is an element of the set Service Terms denoted by $STerms$. A $term_s \in STerms$ is a tuple such that,

$$term_s =< name, value, type_a >$$

where name and value denote the name and value of a service term and $type_a$ describes its aggregation type.
This research proposed an extension of WS-Agreement standard by a new mandatory element, namely $type_a$. The $type_a$ element corresponds to the aggregation function that helps to automate the aggregation of SLAs. Its definition is postponed to the latter part of the paper with the discussion of the aggregation process.
Definition 2 (Guarantee Term). A guarantee term denoted by $term_g$ is an element of the set Guarantee Terms i.e, $GTerms$. A $term_g \in GTerms$ is a tuple such that:

$$term_g =< SLO, condition_q, BVL >$$

where SLO represents Service Level Objectives, $condition_q$ represents Qualifying Conditions and BVL represents Business Value List. Combining the above two definitions, now we can define the notion Terms in WS-Agreement.
Definition 3 (Term). A $term \in Terms$ is a tuple such that

$$term =< term_s, term_g >$$

where $term_s \in STerms$ and $term_g \in GTerms$.

Following the above definitions, SLA can now be formally defined as:
Definition 4 (SLA). A service Level Agreement (SLA) denoted by sla is a tuple

$$sla =< Name, Context, Terms >$$

where $Terms = \cup_{i=1}^{n} term_i$ and Context is a list of strings. Context defines the names of the SLA provider, the consumer and the initiators. It also contains the duration of the SLA. The parameter *Name* denotes the name of the SLA.

A Virtual Organization (VO) in business context is a temporary or permanent, coalition of geographically dispersed organizations expressing high level mutual trust to collaborate and share their resources and competencies in order to fulfill the customers' requests. Web services scattered across various administrative domains, when composed together, are said to form service choreographies. In these service choreographies many service-to-service SLAs are formed. The situation becomes even more complex in Business Value Networks, where services scattered across many of such Virtual Organizations (VO) collaborate to enable complex supply chain networks. One way to visualize this hierarchy is by dependency layers where each layer is dependent on the layer beneath it. A hierarchy of corresponding SLAs pertains to this chain of services. There is no multi-level SLA model that can describe the hierarchical aggregation of SLAs in such Business Value Network. This hierarchical aggregation of SLAs will be called as *SLA-Choreography* with relevance to the Service Choreography.

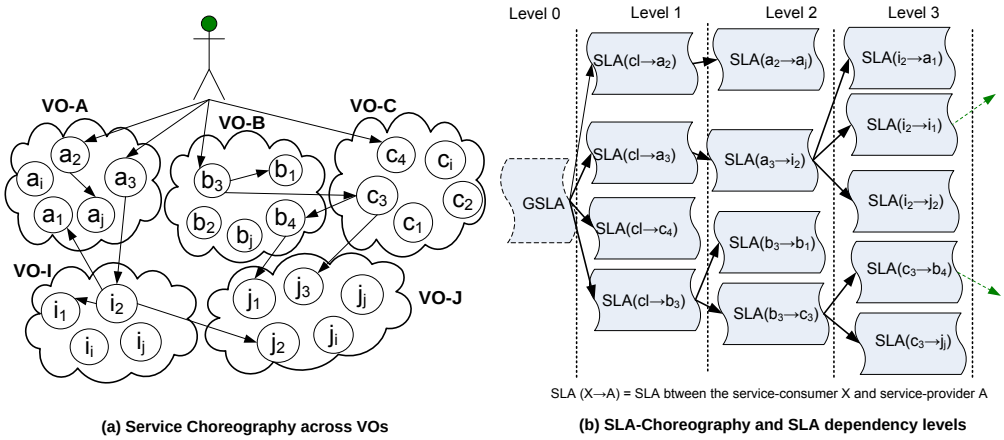


Figure 5.2.: Hierarchical Aggregation of SLAs

Figure 5.2, presents a simplified picture of a cross-VO choreography. The client (that may be a workflow process) is directly connected to some services scattered across three VOs: VO-A, VO-B, VO-C. These services are coordinating with other services to carry out their jobs. This coordination results into service chains, distributed across multiple Virtual Organizations. This scenario can be compared with a simple Business Value Network. The partner services play the producer-consumer roles in this service choreography. All of these services establish Service Level Agreements (SLA), thus giving rise to an SLA-Choreography in connection with the underlying service choreography.

Another way to visualize this SLA Choreography is in terms of hierarchical organization of SLAs. There may be several dependency layers in this SLA-Choreography. The aggregated effect of this dependency travels from the very bottom towards the topmost level. This SLA aggregation is depicted in Figure 5.2. In this hierarchy the SLAs, which are connected to the client process, are said to exist on level 1. This hierarchy indicates a supply chain type of correspondence among the services. These layers also denote the visibility levels of service providers and the client. The client has concerns only with the services immediately connected to it and can not see beyond. Similarly a service can see its coordinating services, i.e its providers and its consumers, with which it is establishing service level agreements. It has no information about the rest of the service choreography. Despite of its privacy concerns, a service is dependent on its lower services. The effect of SLAs formed among the services at lower levels is bubbled up through the upper layers.

One of the objectives of this chapter is to develop a formal model that can describe this SLA Choreography and construct an aggregation model for hierarchical SLAs while protecting the privacy concerns of the stakeholders at the same time. For this purpose the concept of SLA-Views is employed.

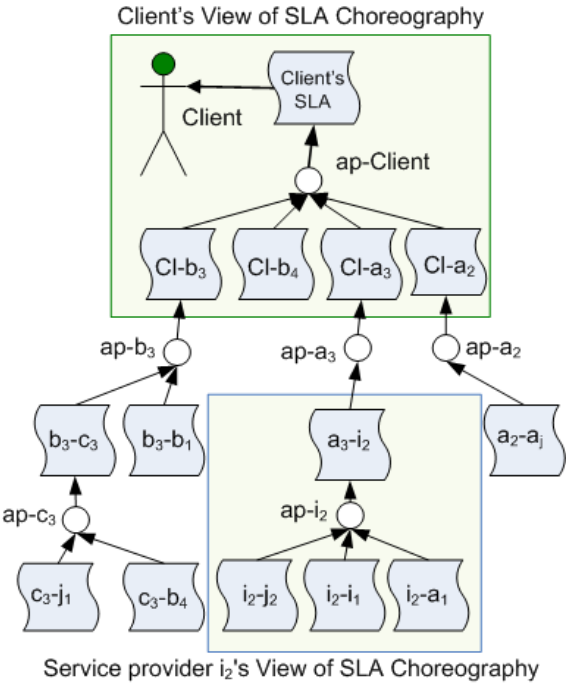


Figure 5.3.: Different Views in SLA Choreography

5.3.2. SLA Views and SLA Choreography

The concept of *Views* originates from the field of databases and has been successfully adapted in business workflows [37][52]. In workflows, a view can be a subset of that workflow or can be a representation of that workflow in aggregated or abstracted fashion. The notion of views has been employed to represent a subset of SLA-Choreography. As the matter of fact the notion of SLA-Views is related to that of workflow views only in a very abstract sense. From the formal point of view, SLA-Views are very much different from workflow views. SLA Choreography is not a workflow; so the rules of workflows are not applicable on it. For instance, in a workflow, rules such as: there should be a single start and single exit or every split should have a join, do not apply on SLA Choreography.

A view in an SLA-Choreography represents the visibility of a business partner. Every service provider is limited only to its own view. A partner (for example a service) makes two kinds of SLAs: the SLAs for which it acts as a consumer and the SLAs for which it is a provider. For clarity, these two types are named as the *consumer-oriented SLAs* and the *producer-oriented SLAs* respectively.

In Figure 5.3, SLAs are connected to small circles, which are called *aggregation points*, by certain edges called *dependencies*. There are two types of dependencies. Consumer-oriented SLAs can be connected to the aggregation points from below by the *consumer role dependencies*, indicating that the *ap* has a consumer role with respect to that SLA, whereas the producer-oriented SLAs are connected to the aggregation point from above by

the *producer role dependencies*. It must be noticed that the producer and consumer roles of SLAs are reflected through their respective dependencies with reference to a particular aggregation point (ap). Thus one SLA may have two roles with respect to two aggregation points, it is connected to. The notion of SLA View does not need to take into account any loops or cyclic graphs. An SLA View corresponds to a unique producer-oriented SLA. This important property plays a crucial role to track down the precise Value Chain corresponding to a specific composite service within an SLA Choreography. Cyclic situations can be accommodated by following the technique introduced in the section 5.4.2.3. To understand the overall picture of the SLA-Choreography, one needs to formalize these concepts.

Definition 5 (Aggregation Point). An Aggregation Point *ap* is an object such that

$$ap =< aggsla, KB >$$

where *aggsla* is the aggregated SLA produced by aggregating the consumer-oriented SLAs connected to it. KB denotes the *Knowledge Base* consisting of business rules, aggregation rules, policies and facts. The business rules and the aggregation rules inside KB play an important role during the negotiation, aggregation and validation [126] processes. In Figure 5.3 *ap-*i_2 is an aggregation point. An aggregation point is the point where the consumer-oriented SLAs (of the consumer service) are aggregated and on the basis of their aggregated content the service is able to decide what it can offer as a provider. The master-slave relationships in Business Value Networks are directly translated to producer-consumer model with one service provider (enterprize) as a producer and other as the consumer. So both the producer and the consumer enterprizes will have their own aggregation points connected together through their mutual SLA. However, for peer-to-peer relationships, both peers act as producer and consumer of services. This issue can be easily resolved by translating peer-to-peer relationships into producer-consumer model. This has been discussed in detail in Section 5.4.2.2.

Now let us define dependencies which have been shown in Figure 5.3(a) as edges joining the aggregation point with the producer and consumer oriented SLAs. The Aggregation Point *ap-*i_2 is connected to three consumer-oriented SLAs and one producer-oriented SLA through dependencies.

Definition 6 (Producer Role Dependency). A *producer role dependency* dep_{pr} is a tuple

$$dep_{pr} =< ap, sla >$$

where *ap* is the aggregation point and *sla* is the producer-oriented SLA. In Figure 5.3(a) it is represented by the directed edge from the aggregation point *ap-*i_2 to the producer-oriented SLA, $sla_{a_3-i_2}$.

Each $dep_{pr} \in Dep_{pr}$, where Dep_{pr} is the set of all producer role dependencies within the SLA-Choreography. Let

$$prodrole : (AP) \rightarrow Dep_{pr}$$

$prodrole(ap_i)$ is the unique $s \in Dep_{pr}$, for which a unique producer-oriented SLA exists with $s = (ap_i, sla_i)$. This means that the function *prodrole* maps each aggregation point ap_i to a unique SLA through a unique producer role dependency s.

Definition 7 (Consumer Role Dependency). A *consumer role dependency* dep_{cr} is a tuple

$$dep_{cr} =< sla, ap >$$

where ap is the aggregation point and sla is the consumer-oriented SLA. In Figure 5.3, it is represented by the directed edge from the consumer-oriented SLA i_2-i_1 to the aggregation point ap-i_2. The aggregation point ap-i_2 is connected with three consumer role dependencies.

Each $dep_{cr} \in Dep_{cr}$, where Dep_{cr} is the set of all consumer role dependencies within the SLA Choreography. Let

$$consrole : (AP) \rightarrow P(Dep_{cr})$$

where $P(Dep_{cr})$ is the power set of Dep_{cr}.

$consrole(ap_i)$ is the set $S_{cr} \in P(Dep_{cr})$, i.e. $S_{cr} \subseteq Dep_{cr}$ such that for each $s_i \in S_{cr}$ a unique consumer oriented SLA exists with $s_i = (sla_i, ap_j)$. This means that the function *consrole* maps a set of consumer-orieted SLAs to a unique aggregation point such that each consumer-oriented SLA sla_i is mapped through a unique consumer role dependency s_i.

Definition 8 (Dependency). A dependency Dep is a set that is the union of two sets namely Dep_{pr} and Dep_{cr}, which are pairwise disjoint, i.e.

$$Dep = Dep_{pr} \cup Dep_{cr}$$

$$Dep_{pr} \cap Dep_{cr} = \phi$$

Based on these definitions, it is evident in Figure 5.3 that the producer-oriented SLA (a_3-i_2) is dependent on the terms of the corresponding consumer-oriented SLAs, aggregated at ap-i_2 . For example the bandwidth and space aggregated at ap-i_2 would be the upper limit of what service i_2 can offer to service a_3. At the same time service i_2 will have to decide about its profit on the basis of the information about total cost in the aggregated SLA using business rules from within its KB. The aggregation point in this sense is also a decision point for a service.

With having all the related concepts formalized, now it is possible to provide a formal definition of the SLA-View.

Definition 9 (SLA View). An SLA View denoted by *slaview* is a tuple such that

$$slaview_i =< sla_{p_i}, dep_{pr_i}, ap_i, SLA_{c_i}, Dep_{cr_i} >$$

where sla_{p_i} is a producer-oriented SLA, SLA_{c_i} is a set of consumer-oriented SLAs, dep_{pr_i} is a producer role dependency between ap_i and sla_{p_i} and Dep_{cr_i} is the set of consumer role dependencies between the members of SLA_{c_i} and the ap_i. Each aggregation point ap_i in

the SLA Choreography corresponds to a unique *sla-view*$_i$.
In Figure 5.3 the SLA Views of the client and a service are highlighted.
Definition 10 (SLA Choreography). An SLA_{chor} is a tuple such that

$$SLA_{chor} =< SLA, APoints, Deps >$$

where SLA is the union of all the sets SLA_{c_i} and $\{sla_{p_i}\}$ corresponding to all $slaview_i$ within an SLA Choreography. $APoints$ is set of aggregation points ap, and $Deps$ is set of dependencies dep.
Definition 11 (Projection Function onto Aggregation Point). The projection function $\prod_{ap_i}$ onto an aggregation point ap_i is defined as:

$$\prod_{ap_i} : SLA_{Chors} \rightarrow SLAViews$$

$$\prod_{ap_i} : SLA_{chor} = slaview_i$$

i.e. it projects the slaview corresponding to a specific ap_i.

In terms of Business Value Networks, it should be noted that SLA View defines boundaries of a stakeholder. The aggregation process is performed at every aggregation point. Each aggregation point, which also denotes a dependency level, belongs to one of the service providers. Although each service provider is limited to its own aggregation information, this information is in fact dependent on the aggregation information at lower levels. The sustainability of this business network requires all the stakeholders to trust each other and their ability to maintain their privacy at the same time. SLA-Views maintain a balance between this privacy and trust.

5.3.3. Aggregation of Service Terms

In the aggregation process terms of the consumer-oriented SLAs are aggregated. WS-agreement has no direct support for such an aggregation. So an attribute for aggregation type, namely "$type_a$" was introduced in Definition 1. WS-Agreement gives the liberty to incorporate *any* external schema. Therefore $type_a$ can be made an essential part of the service terms and will describe, how the corresponding service will behave during the aggregation process. One can define $type_a$ in a formal way, as follows:
Definition 12 (type$_a$). A $type_a \in Types$ is a function that maps a set of terms to a single term, which is the aggregation of that set:

$$type_a : P(Terms) \rightarrow Terms$$

$$type_a(term_1, ...term_n) = term_{agg}$$

$type_a$ is defined as an aggregation function that aggregates n terms into one term. Its result is *aggsla* in the aggregation point (see Definition 5). The structure of aggsla adheres to the WS-Agreement standard. Each term in *aggsla* is computed by applying the type function for that term to the values of the terms for all the dependent (consumer-oriented) SLAs, which define that term.

5.3.4. Aggregation of Guarantee Terms

Guarantee Terms (GTs) can also be aggregated together similar to the Service Description Terms (SDTs) as described in the previous section. However there are certain peculiarities to be considered when it comes to the Guarantee Terms. First of all, Guarantee Terms are optional terms in context with the WS-Agreement standard. Secondly, even if two aggregating SDTs have GTs associated with themselves, the GTs may refer to different service level parameters, i.e. they provide guarantees in form of Service Level Objectives (SLOs) about different properties of the service.

An example is a service consumer who wants to aggregate two similar storage services. The aggregation of SDTs will give him the total sum of available disk space, but what if two vendors are providing GTs describing entirely different aspects, e.g. access time and availability of service? These two aspects are not related hence can not be aggregated. To solve this problem there can be many approaches:

- A solution to this problem can be found if the SLA negotiation process somehow facilitates the consumer to ask for guarantees upon the desired properties of services thus helping him setting up the identical guarantees with its different service providers. Not only this type of mechanism is very difficult to achieve, but by restricting the variability of SLA contents, it also turns out to be contrary to the automation requirements of the process for which it was originally designed for.

- A similar approach can be based on the renegotiation for a revision of SLA with new guarantees. This approach may not be successful every time because the service provider may not be in a position to offer the required type of guarantee.

- Some popular (or straightforward) guarantees may be standardized to be always offered for the relevant services by all the service providers. This approach will definitely improve the situation but there will be always new services with innovative properties expressed by unseen guarantees.

- If the guarantees translate to the quality of service then in some situations it may be desirable to use ORType aggregation in order to segregate the services on the basis of their guarantees. For this purpose the service terms should be declared as ORType.

- The most straightforward and safe method is to leave the guarantees disaggregated and the situation should be reported to the service provider to take some decision. In this way, we may allow each service provider in the supply chain to figure out and set up its own Guarantee Terms during the aggregation process based on its personal business rules.

The last approach also conforms to the proposed formal model. The aggregation point is also considered as the decision center of the service provider as well. Within this decision center, the aggregation of SLAs is performed to facilitate the formation of business objectives of the service provider. Therefore, when a stake-holder in the supply chain acts as a service provider, it needs to layout its business strategy at least once before starting the

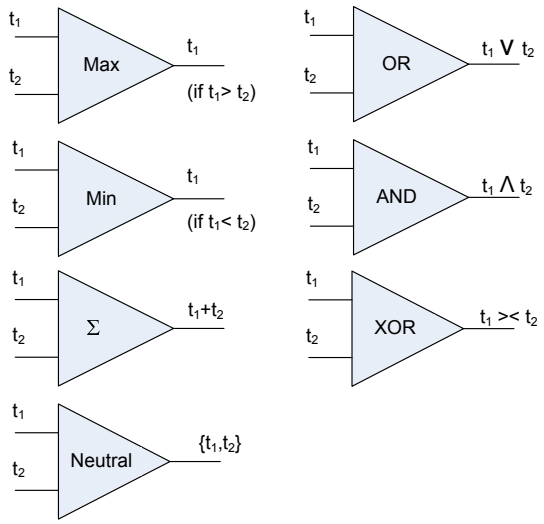


Figure 5.4.: SLA Aggregation Patterns for Service Composition

provision of services. The proposed aggregation model thus only promises a semi-automatic aggregation of Guarantee Terms. In that context, the aggregation of Guarantee Terms is purely a business issue and is interlinked with the business goals of the service provider. This approach also resolves another very crucial issue of aggregating reward and penalty expressions. Within the aggregation point, the reward and penalty expressions must be expressed in accordance with the business rules of the service provider. A subset of those business rules may be dedicated especially to facilitate the aggregation process.

5.4. Aggregation Patterns for SLA Choreography

SLA aggregation patterns are divided into two categories in context with the *resource provision* and the *infrastructure topologies*. These two types are called *Composite Service Provision Patterns* and the *Enterprize Structural Patterns*.

5.4.1. Composite Service Provision Patterns

Extending Definition 12, in the present context, seven types of terms are defined but the enumeration is extendable and new types can be added if the need arises:

$Types = \{sumtype, maxtype, mintype, neutral, ORtype,$
$ANDtype, XORtype\}$

These functions are depicted in Figure 5.4.

5.4.1.1. The Sumtype Pattern

The function *sumtype* can be formally defined as follows.

$sumtype \in Types(\Leftrightarrow sumtype : P(Terms) \rightarrow Terms)$
$sumtype(term_1, ...term_n) = \sum_{i=1}^{n} term_i.term_s.value$

$type_a$ is an aggregation function that aggregates n number of terms into one term. sumtype is of the type of $type_a$ and takes the summation of all terms. The dot operator facilitates the access to the value coordinate of the tuple $term_s$. Examples include terms for storage space, memory, availability and cost.

5.4.1.2. The Maxtype Pattern

$maxtype \in Types(\Leftrightarrow maxtype : P(Terms) \rightarrow Terms)$
$maxtype(term_1, ...term_n) = \max_{i=1}^{n} term_i.term_s.value$

maxtype is an aggregation function that aggregates n number of terms into one term. It does so by picking up the maximum of these terms, which represents the aggregation of all the input terms. If several terms addressing the same utility are being aggregated and their type has been declared as *maxtype*, then only the term pertaining to the maximum value will become part of the aggregated SLA. Examples include latency, which may become a bottle neck for the whole process and an activity with highest latency will directly contribute (though in a negative sense) to the throughput of a workflow sequence.

5.4.1.3. The Mintype Pattern

$mintype \in Types(\Leftrightarrow mintype : P(Terms) \rightarrow Terms)$
$mintype(term_1, ...term_n) = \min_{i=1}^{n} term_i.term_s.value$

mintype is an aggregation function that aggregates a number of terms into one term. It does so by picking up the minimum of these terms, which represents the aggregation of all the input terms. Similar to *maxtype*, when several terms addressing alike utilities are being aggregated and their type has been declared as *mintype* then only the term pertaining to the minimum value will contribute to the aggregated SLA. An example can be the bandwidth. In a sequence of activities the activity pertaining to the minimum bandwidth will become the bottleneck for the whole sequence making other activities with higher bandwidth ineffective.

5.4.1.4. The Neutral Pattern

$neutral \in Types(\Leftrightarrow neutral : P(Terms) \rightarrow Terms)$
$neutral(term_i) = \ term_i$
for any individual term $term_i$ and is defined on $P(Terms) \backslash Terms$.

neutral is an aggregation function that includes all the input terms separately without any processing. This function is applied on those terms which can not be mixed with other terms and need to be preserved in the aggregation process as separate terms. The terms declared as neutral are unaffected through the aggregation process because there are no similar terms in any of their peer consumer oriented SLAs. Therefore the neutral terms even after passing through the aggregation operation remain in their original form as they were in their parent consumer-oriented SLAs. They represent services which are

independent from similar services, for example identity of some valuable data in a certain organization or discount in a specific service, etc.

5.4.1.5. The ORtype Pattern

$ORtype \in Types(\Leftrightarrow ORtype : P(Terms) \rightarrow P(Terms)$

$ORtype(term_1, ...term_n) = \bigvee_{i=1}^{n} term_i.term_s.value$

ORtype is an aggregation function that aggregates a number of terms into one or more terms. It does so by applying a logical OR function on these terms and the result represents the aggregation of all the input terms. For instance, a service provider who wants to aggregate resources of varying qualities but would also like to segregate them under different levels of SLAs, may use *ORtype* aggregation function for this purpose. An example could be a reseller who buys computational resources of different speeds and qualities from different vendors and aggregates them using *ORtype* function so that later, he can offer SLAs of different levels such as gold, silver or bronze, etc. to its consumers. Another example could be a composite service provider with two possible suppliers such that supplies from either of them or from both can be required. For instance a VOIP service may resell services from two companies.

5.4.1.6. The ANDtype Pattern

$ANDtype \in Types(\Leftrightarrow ANDtype : P(Terms) \rightarrow P(Terms)$

$ANDtype(term_1, ...term_n) = \bigwedge_{i=1}^{n} term_i.term_s.value$ *ANDtype* is an aggregation function that aggregates a number of terms into the same number of terms. It does so by applying a logical AND function on these terms and the result represents the aggregation of all the input terms. An example could be two services which complement each other. For instance a specific payment method which is associated with a service. Another example can be of main memory service which is always sold with the computing service.

5.4.1.7. The XORtype Pattern

$XORtype \in Types(\Leftrightarrow XORtype : P(Terms) \rightarrow Terms)$

$XORtype(term_1, ...term_n) = X_{i=1}^{n} term_i.term_s.value$

XORtype is an aggregation function that aggregates a number of terms into one term. It does so by applying a logical AND function on these terms and the result represents the aggregation of all the input terms. An example could be a reseller with two possible suppliers with the first acting as the prime supplier and the second active only during the absence of the first. For instance a a reseller of internet bandwidth. Another example is of the survive clusters which always have a few sets of backup resources for the sake of reliability which are utilized in case of the failure of the preferred sets of resources.

5.4.2. Enterprize Structural Patterns

Not only services compose together through SLAs to form complex service value chains but different business enterprizes can also make collaborations under Service Level Agreements to form business networks. To apply the concepts of SLA Choreography on services composing together across the organizational boundaries within these business consortiums, we need to represent the structure of the business network in terms of SLA Views. The Enterprize structural patterns consist of the basic building blocks to represent the structure of collaborating enterprizes as composed of SLA Views. For this purpose, various basic structural relationships such as sharing, P2P, loops and nestings within organizational structures have been considered. Once the overall structure of a cross-enterprize collaboration is resolved by applying these structural patterns, it is possible to represent the resultant enterprize in terms of SLA Views.

5.4.2.1. The VEO Pattern

Here it is required to define a Virtual Enterprize Organization (VEO). According to NESSI's definition [10] VEOs are formed when two or more administrative domains overlap and share resources. If three enterprizes A, B and C are considered to share resources in order to form a Virtual Enterprize Organizations (VEO), their SLAs are aggregated at a virtual aggregation point (vap) that represents this VEO. The virtual aggregation point is important to be represented, because it describes the SLA view of the resulting VEO, which is different from the SLA views of A, B and C. The shared functionality of the VEO is described in the aggregated SLA computed within the vap-[ABC]. Note that the big brackets have been adopted to highlight the jointly contained capabilities of enterprizes A, B and C.

Once the architecture is resolved in terms of SLA views, the terms of services can be aggregated through aggregation functions described in Section 5.4.1.

5.4.2.2. The P2P Pattern

So far, the aggregation of SLAs has been discussed in context with the composition of services in a producer-consumer manner along service value chains. This service level SLA aggregation model can be scaled up to enterprize level. It can conveniently describe both master-slave and peer-to-peer relationships in cross-enterprize collaborations e.g. in Business Value Networks. Master-slave relationship can be simply mapped on the producer-consumer model where an SLA is formed between the service provider and the client. However, in a peer-to-peer relationship, both of the participating enterprizes are acting as a service provider and as a client at the same time. To form a WS-Agreement compliant SLA between them, one party can either be treated as a service provider or a service consumer in context with some service. Therefore a peer-to-peer relationship needs to be dissolved into two producer-consumer relationships with a separate SLA associated with each of them. NESSI, in its Grand Vision and Strategic Research Agenda (SRA) [10], defines Value Networks as the ways in which organizations interact with each other to drive

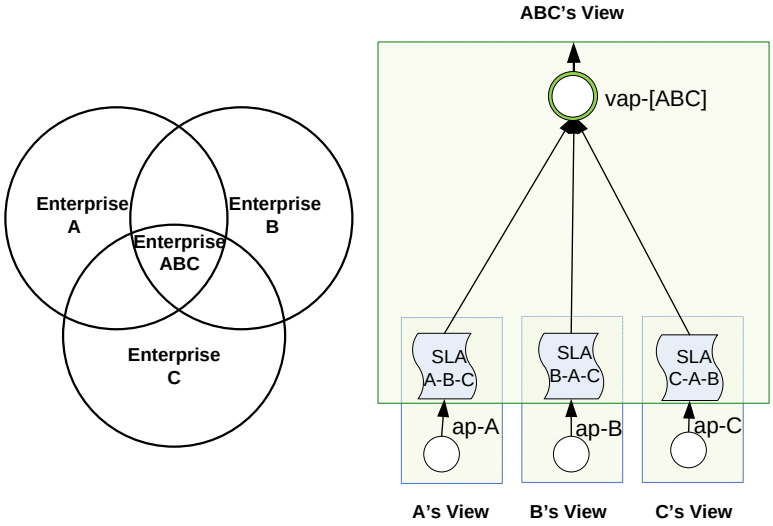


Figure 5.5.: SLA Aggregation Pattern for Virtual Enterprize Organization (VEO)

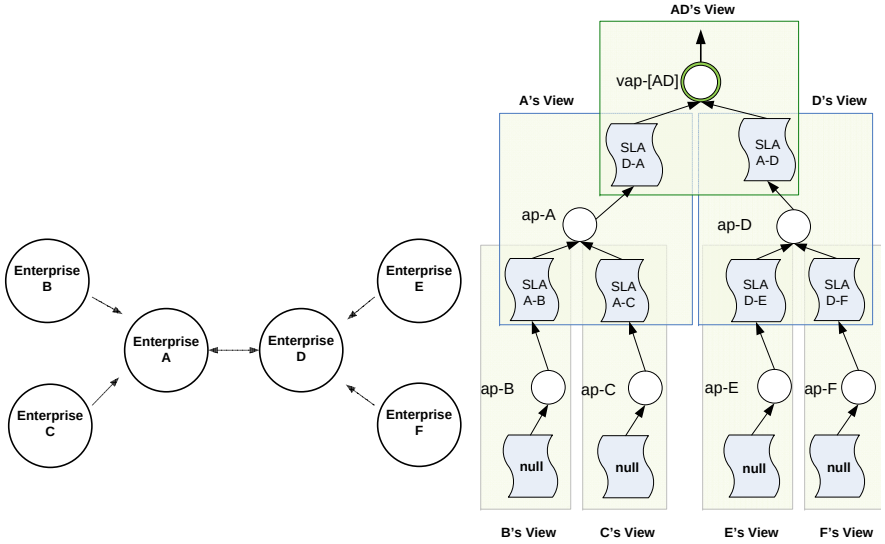


Figure 5.6.: SLA Aggregation Pattern for P2P relationships in a Value Network

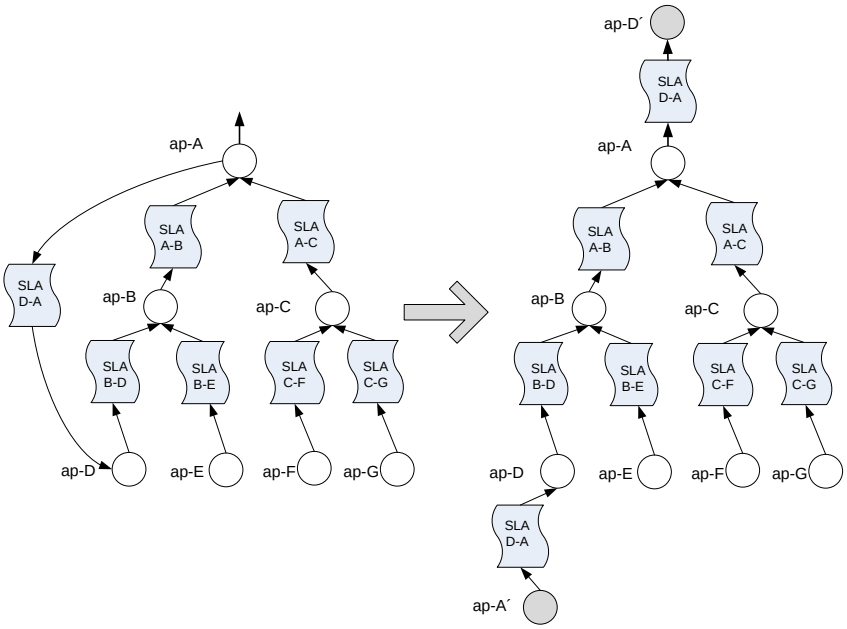


Figure 5.7.: SLA Aggregation Pattern for Cycles

increased business value. Figure 5.6 shows their example Business Value Network (BVN) where the Enterprizes A and D have been shown to collaborate on the development of a new product. Enterprize A has subcontractors B and C whereas the enterprize has E and F as subcontractors. The Enterprizes A and D form a peer-to-peer relationship between themselves.

The concept of VEO to peer-to-peer relationships is applied in Figure 5.6. If one considers the enterprizes A and D to form a Virtual Enterprize Organizations (VEO), their SLAs are aggregated at a virtual aggregation point (vap) that represents this VEO. The shared functionality of the VEO is described in the aggregated SLA computed within the vap-[AD]. The terms of services can now be aggregated through aggregation functions described in Section 5.4.1.

5.4.2.3. The Cyclic Pattern

The Cyclic Pattern as shown in Figure 5.7, is used to resolve cyclic graphs into acyclic graphs. While translating an service choreography in terms of SLA Choreography, it is possible to encounter cyclic graphs. The knowledge representation notations of SLA Choreography do not support cyclic situations. However, in Figure 5.7, it has been shown how can we translate a cyclic graph into an acyclic graph by duplicating an aggregation point in two different SLA views. The aggregation points A and D have been duplicated in Figure 5.7 to highlight the consumer and the producer roles of these two service providers (or stake-holders) separately. An example of this pattern is the situation when a reseller

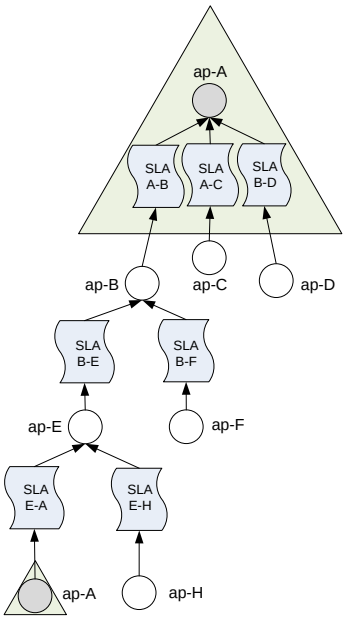


Figure 5.8.: SLA Aggregation Pattern for Nesting

rents services from one of its subsequent clients through the value chain.

5.4.2.4. The Nesting Pattern

The Nesting Pattern as depicted in Figure 5.8, represents iterations within business processes. While translating an organization's business process (or a workflow) in terms of SLA Choreography, it is possible to encounter repeating sequences of business activities. An example of this pattern can be the sequence of activities required for payment transactions. As payment transactions can be used at various points of a business process, therefore SLA structure involving the bank, secure transaction channels and the trust managers etc. can be reused without redefinition. In Figure 5.8, the aggregation point A represents the entry point into a nesting involving three players.

5.5. Running Example – Aggregation of SLAs

In this section the running example is employed to realize the aggregation mechanism presented above. in Figure 5.9, there are two services, namely the rendering workflow service and the hosting service. The rendering workflow service aggregates the media engine service and the computational infrastructure service to get the videos rendered whereas the host video service downloads the video from a specified location, archives it and makes it available online. An authenticated user can play the video in a YouTube like style.

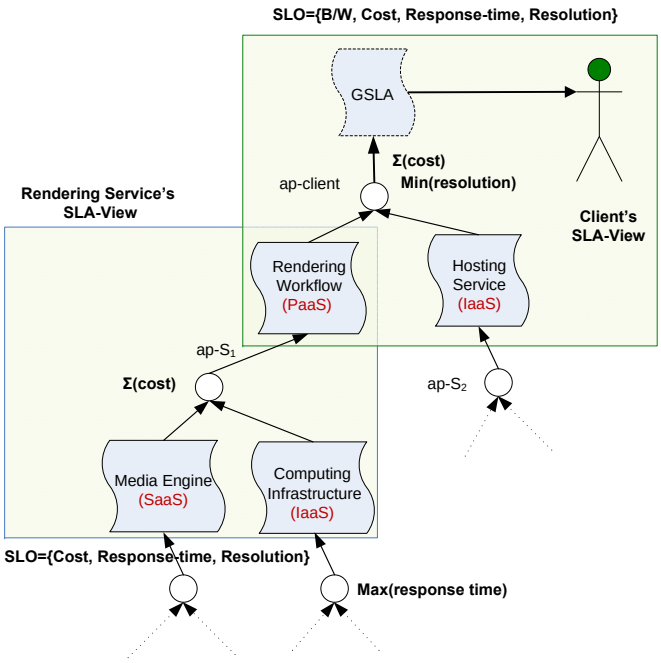


Figure 5.9.: Running Example - Hierarchical Aggregation of SLAs

The SLA-Choreography resulting from this scenario is depicted in Figure 5.9. The aggregation functions described in Figure 5.4 are being applied in the scenario shown in Figure 5.9. It is evident that the resolution provided to the end-client is the minimum of the hosting service and rendering workflow service. So at the aggregation point *ap-client*, the aggregation function Min will choose only minimum of the two resolutions. On the same grounds the total cost that the client has to pay is the sum of the cost incurred on hosting and the cost spent on rendering workflow, because cost has been declared as "sumtype".

Here, the liberty has been taken to import an external schema into WS-Agreement's Service Description Terms' section. The following chunk of Schema allows this.

```
<xs:complexType name="ServiceDescriptionTermType">
   <xs:complexContent>
      <xs:extension base="wsag:ServiceTermType">
         <xs:sequence>
            <xs:any namespace="##other" processContents="strict"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

The above schema enables us to include an XML structure of elements adhering to any external Schema. This makes it possible to incorporate the aggregation type (typea) element

inside a Service Description Term. A simple schema to accomplish this can be written as follows.

```
<?xml version="1.0" encoding="utf-16"?> <xs:schema
xmlns:myns="http://schemas.xyz.com" xmlns="http://www.mynamespace.com"
targetNamespace="http://www.mynamespace.com"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:simpleType name="aggregationType">
 <restriction base="xs:string">
   <enumeration value="Mintype"/>
   <enumeration value="Maxtype"/>
   <enumeration value="sumtype"/>
   <enumeration value="neutral"/>
  </restriction>
</xs:simpleType> ...
  <xs:element name="Resolution">
    <xs:complexType>
      <xs:sequence>
        <xs:complexType name="ResolutionXY">
          <xs:sequence>
            <xs:element name="ResolutionX" type="xs:integer"/>
            <xs:element name="ResolutionY" type="xs:integer"/>
          </xs:sequence>
        <xs:element name="aggregationType" type="xs:aggregationType"/>
       </xs:complexType">
      </xs:sequence>
    </xs:complexType>
  </xs:element>
... </xs:schema>
```

Then the service Description Term namely "resolution" for the Enhance-Video service may be expressed as follows.

```
<wsag:ServiceDescriptionTerm wsag:Name=Resolution"
wsag:ServiceName="Enhance-Video">
   <myns:ResolutionXY>
      <myns:ResolutionX>1920</myns:ResolutionX>
      <myns:ResolutionY>1080</myns:ResolutionY>
   </myns:ResolutionXY>
   <myns:aggregationType> mintype</myns:aggregationType>
</wsag:ServiceDescriptionTerm>
```

The aggregationType (i.e. $type_a$) declares Resolution as a minType term. When it will be aggregated with other minType terms, only the minimum of these terms will become

part of the aggregated SLA. Other aggregation types listed in the schema can be expressed and aggregated in a similar fashion.

5.6. Summary

This chapter addresses the very important topic of SLA aggregation along hierarchical structures such as service value chains and business value network. The notions of SLA Choreography and SLA View have been put forth in this chapter. SLA Choreography realizes the network of SLAs existing side by side with the the service choreography and describes it in a formal manner. SLA Views preserve the privacy of different stakeholders in a SLA Choreography. This formal approach facilitates the process of aggregation and results into the discovery of several aggregation patterns at the architectural and provision levels of services. These aggregation patters are likely to play an important role in the analysis of SLA-centric service value chains, value networks, and business process management etc.

Chapter 6.

Hierarchical SLA Validation and Distributed Trust Management

> Happy are those who find fault in themselves instead of finding fault with others.
>
> *(Hazrat Muhammad S.A.W)*

This chapter presents a multi-agent rule-based validation model for SLA choreography and a corresponding distributed trust management model. The distributed trust management model is not only an essential part for the proposed validation model but also serves as an enabling requirement for the overall framework for SLA-centric service-based Utility Computing.

The validation model for SLA choreography has following characteristics associated with it.

- It is a Multi Agent System (MAS), where different agents correspond to various indigenous components of distributed nature.
- It represents aggregated SLAs as a set of distributed rules located within the knowledge-bases of different agents.
- It employs a validation mechanism based on the approach of distributed query processing across heterogeneous enterprize boundaries.

The distributed trust management model has following attributes.

- It is a hybrid trust system based on Public Key Infrastructure (PKI) and reputation-based trust models.
- It has a third party trust manager, which plays a key role as root Certification Authority (CA) as well as a root trust reputation manager.

6.1. Background and Challenges

Validation of hierarchical SLA Choreography is a distributed problem. The service choreography may be distributed across several Virtual Organizations and under various administrative domains. It has been discussed that the SLA Choreography is realized on the

basis of a formal model that utilizes the concept of SLA Views to preserve the privacy of stakeholders.

This hierarchical choreography of heterogeneous services is only possible through a well defined distributed trust schema. Another challenge in this regard is the step-wise aggregation of SLAs for the series of service providers at different levels in the service chain.

The complete information of aggregated SLA at a certain level in the service chain is known by the corresponding service provider and only a filtered part is exposed to the immediate consumer. This is the reason why during the validation process, the composed SLAs are required to be decomposed in an incremental manner down towards the supply chain of services and get validated in their corresponding service providers's domain. A validation framework for the composed SLAs, therefore, faces many design constraints and challenges including:

- a trade-off between privacy and trust
- distributed query processing
- business automation
- local SLA implementation
- interoperability etc.

Addressing these challenges, the foundations of the validation framework can be laid down.

6.2. Enabling Requirements

The aforementioned challenges bring in a cross-section of enabling technologies depicted in Figure 6.1. The privacy concerns of the partners are ensured by the SLA View model, whereas the requirement of trust can be addressed through a distributed trust model. Distributed query processing, automation and the support for local SLA implementations can be achieved through a Multi Agent System. Each agent can represent a stake-holder thus corresponding an SLA View. The agents can have their local knowledge bases and distributed query can be directed across the chain of agents to be validated. Different parts of the WS-Agreement compliant SLAs can be transformed into corresponding sets of logical rules, which can compose together during the process of SLA composition and can be decomposed into separate queries during the process of validation.

Figure 6.1 shows four enabling technologies for the validation of hierarchical SLAs.

6.2.1. Multi Agent System

Hierarchical SLA validation requires an approach based on Multi Agent Systems (MAS) because of the following characteristics of these systems.

- Autonomy: Different stakeholders contributing in the SLA Choreography as independent administrative entities are required to be treated as autonomous systems.

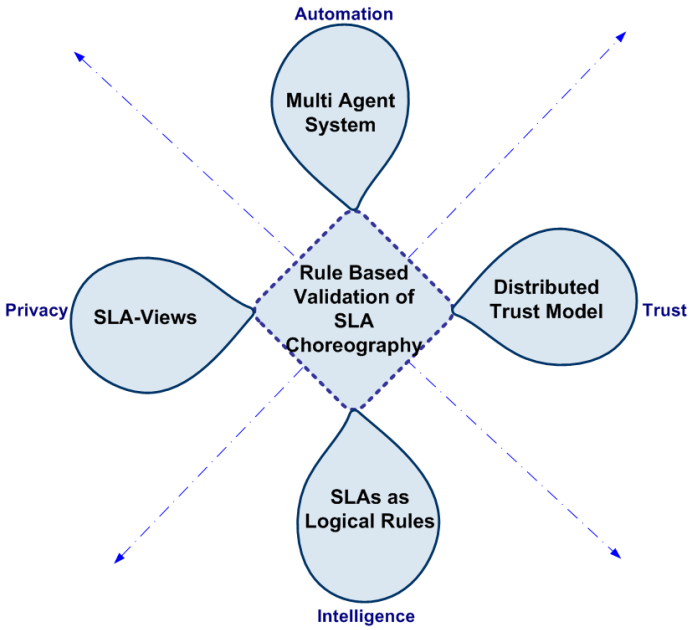


Figure 6.1.: Validation of SLA Choreographies as a cross-section of enabling technologies

- Local Views: The overall SLA Choreography is a distributed systems with every stakeholder represented by its personal agent. Each personal agent has its local knowledge base and local policies. The local View model is in absolute harmony with the notion of SLA Views.

- Decentralization: Their is no central body directly controlling these interacting stakeholders. The system is in this sense decentralized where the validation query needs to cross administrative boundaries and get transformed in local implementation formats.

6.2.2. SLA-Views

The privacy of different stakeholders is a prime requirement. The privacy is a manifold concept which goes beyond the concept of information concealment and also represents the organizational boundaries of an entity. It also represents local specifications and peculiarities. SLA Views protect the privacy of a stakeholder in SLA Choreography. An SLA View can be practically realized with the help of an intelligent agent. The agent can have its local knowledge base and a specific implementation. The world outside an SLA View interacts with it only through a published interface. This interface includes the exchanged data format and supported communication protocols.

6.2.3. Rule Based SLAs

For distributed query processing in connection with the validation requests over hierarchical SLA Choreography, the SLAs are required to be represented as sets of rules within various SLA Views. These scattered set of rules line up making rule chains in response to validation queries. The Service Level Objectives (SLO) and penalty conditions must be transformed into rules in accordance with some knowledge representation standards. A component within the overall distributed system should be able to utilize its own rule format and rule engines. There must be an agreed upon rule exchange format to help validation queries and their answers navigate across organizational boundaries.

6.2.4. Distributed Trust Model

The distributed trust model is essential for the very existence of SLA Choreography as it binds together heterogeneous components into a single distributed systems. The distributed trust model plays a key role in the hierarchical SLA validation system. The distributed trust system come in two main flavors i.e. either based on PKI or reputation. The validation query needs to navigate across organizational boundaries for which the PKI based trust model facilitates with single sign-on and delegation mechanism. In case of failing services, a reputation based trust model can be helpful to find alternate services and keep the system running.

6.3. A Validation Framework for Hierarchical SLA Choreographies

Figure 6.2 depicts the proposed framework for validating hierarchical SLAs in SLA Choreographies. The validation framework comprises of components from three different but highly entangled systems i.e., SLA Choreography, Distributed Trust and Security and the Rule Based Validation Model. The contributing components have been outlined by red rectangles. A multi agent rule based system known as Rule Responder [109] provides the necessary agent based technology highlighted in Figure 6.1. Every Rule Responder based agent is also a direct implementation of an SLA View. Rule Based SLAs (RBSLA) is a project that specializes on knowledge representation techniques to represent SLAs as logical rules. The knowledge representation techniques of RBSLA project have been utilized in the proposed validation framework. For distributed trust, a hybrid trust model based on the PKI and reputation oriented trust techniques has been utilized. The aggregated SLAs at various levels in the SLA Choreography are represented as logical rules in such a way that the complete self consistent set of rules is distributed along the choreography in the order of the formation of the value chain. As SLA Views have already been discussed in the previous chapter, the rest of the components constituting the validation framework for hierarchical SLA Choreographies are discussed below.

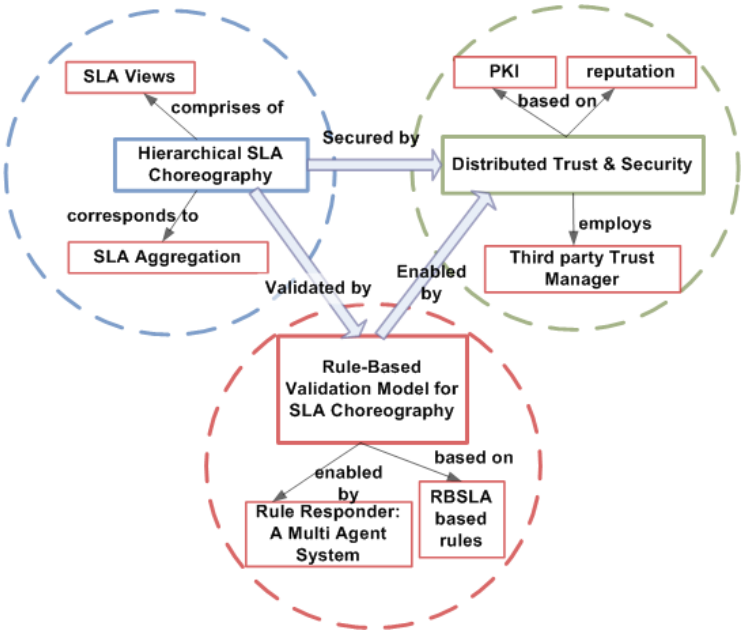


Figure 6.2.: Validation Framework for SLA Choreographies, where red boundaries indicate the directly contributing notions

6.3.1. Rule Responder Architecture

Rule Responder is much more than a multi agent system. Rule Responder is a rule-based enterprize service middleware for distributed rule inference services and intelligent rule-based (Complex) Event Processing on the Web. It utilizes modern enterprize service technologies and Semantic Web technologies with intelligent agent services that access external data sources and business vocabularies (ontologies), receive and detect events (complex event processing), and make rule-based inferences and autonomous pro-active decisions and reactions based on these representations (enterprize decision management) [109]. For a description of the syntax, semantics and implementation of the underlying logical formalisms and its usage in IT Service Management (ITMS) see [106]. Rule Responder adopts the approach of multi agent systems. There are three kinds of agents:

- Organizational Agents
- Personal Agents
- External Agents

A virtual organization is typically represented by an organizational agent and a set of associated individual or more specific organizational member agents. The organizational agent might act as a single agent towards other internal and external individual or organizational agents. In other words, a virtual organization's agent can be the single (or main) point of entry for communication with the "outer" world (external agents). Similar

to an organizational agent, each individual agent (personal and external) is described by its syntactic resources of personal information about the agent, the semantic descriptions that annotate the information resources with metadata and describe the meaning with precise business vocabularies (ontologies) and a pragmatic behavioral decision layer which defines the rules for using the information resources and vocabularies/ontologies to support human agents in their decisions or react autonomously as automated agents/services. The flow of information is from external to organizational to personal agent. Figure 6.3 shows the Rule Responder agents contributing to SLA validation. Two external agents outside of VO invoke the organizational agent by sending HTML and SOAP messages. Typical examples of external agents are web browser, client service or a workflow tool. It must be highlighted that the overall collaboration between VOs is based on choreography, while the internal collaboration model within a VO (one closed enterprize service network) can be either choreography with no central authority or an orchestration with orchestration workflows defined in the organizational agent as under control of a central authority within this particular VO. Rule Responder can span across several VOs and can support both of the collaboration models. In the current scenario, Rule Responder provides the rule-based enterprize service middleware for highly flexible and adaptive Web-based service supply chains.

Rule Responder utilizes RuleML [32] as Platform-Independent Rule Interchange Format. The Rule Markup Language (RuleML) is a modular, interchangeable rule specification standard to express both forward (bottom-up) and backward (top-down) rules for deduction, reaction, rewriting, and further inferential-transformational tasks [110, 104]. It is defined by the Rule Markup Initiative, an open network of individuals and groups from both industry and academia [109]. Figure 6.3 shows Enterprize Service Bus (ESB), the Mule open-source ESB [97], as Communication Middleware and Agent/Service Broker to seamlessly handle message-based interactions between the responder agents/services and with other applications and services using disparate complex event processing (CEP) technologies, transports and protocols. ESB is a framework for highly scalable and flexible application messaging to communicate synchronously and asynchronously with services and agents which are deployed on the bus. The Mule ESB supports more than 30 protocols. Rule Responder supports Platform dependent rule engines. Each agent service can run one or more rule engines to execute the queries, rules and events and derive answers. Currently the Prova [82], OO jDREW [28], and Euler [113] rule engines are implemented as three rule execution environments.

6.3.2. Rule Based Service Level Agreements (RBSLA)

The Rule Based Service Level Agreements (RBSLA) [107, 108, 103, 105, 106] project focuses on sophisticated knowledge representation concepts for service level management (SLM) of IT services. It formulates contracts such as service level agreements or policies in logical rules with the help of rule based languages . The research exploits knowledge representation concepts from Artificial Intelligence as well as in the area of web services and the semantic web. It employs logical formalisms such as defeasible logic, deontic

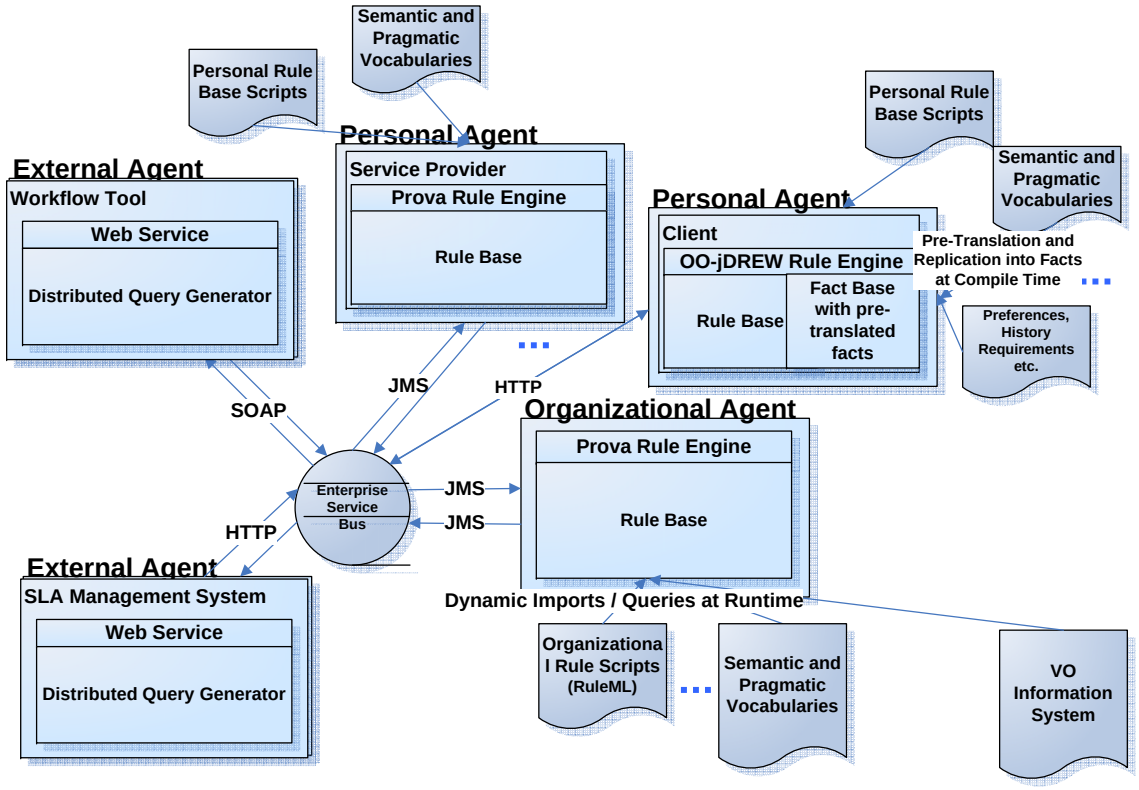


Figure 6.3.: Rule Responder Services for SLA Validation

logic, temporal event/action logics, transaction and update logics, description logics to express formal declarative contract specifications. The logical expression address contract norms such as permissions, obligations and prohibitions and their violations and exceptions. The RBSLA knowledge representation techniques in the present context are utilized for the automated validation and consistency checks of large distributed rule sets through automated chaining and scoped reasoning and local execution of rules.

6.3.3. Distributed Trust Model

Trust not only plays a crucial role in reducing SLA violations in workflow compositions but it has also been shown [134] that maximizing participants trust even helps runtime scheduling to survive in dynamic and open environment. There is need to choose a suitable trust model that integrates seamlessly with our aggregation and validation model.

Trust management has two broad categories as policy-based and reputation-based systems. These two types of techniques have been designed to target diverse nature of challenges in different environments. The policy-based trust management also promotes "strong security" through signed certificates and trusted Certification Authorities (CA). The reputation-based trust on the other hand promotes rather "fuzzy trust", where trust is usually computed from personal experiences as well as through the feedback by other friendly entities who have already used the service in question. However the basic challenge for both of the systems is the same i.e. to establish trust among interacting parties in distributed and decentralized systems. The policy based approach fits very well with structured organizational environments whereas for unstructured organizations, the reputation systems suit better.

The policy-based trust systems are very secure and hence are an essential requirement for the B2B and B2C relationships in virtual organizations and for this reason have been widely adopted in Grid Computing. On the other hand, the reputation-based trust is a lenient approach and are very suitable for self-emergent, automated, ad-hoc and dynamic business relationships across virtual enterprizes. In the line of this research, the best features of both approaches have been employed to propose a PKI coupled Reputation-based Trust Management System. Rule Responders' agents have been used to spawn trust across different stake-holders of a cross-enterprize business relationship.

During service choreography, services may form temporary composition with other services, scattered across different VOs. The question of whose parent VO acts as the root CA in this case is solved by including *third party trust manager* like the case for dynamic ad hoc networks. The distributed trust system should work hand-in-hand with the breach management of the SLA validation framework. In case of SLA violation, in addition to enforcing penalty, the affected party is likely to keep a note of the violating service in order to avoid it in future. Moreover, a fair business environment demands even more and the future consumers of the failing service also have a right to know about its past performance. Reputation-based trust systems are widely used to maintain the reputation of different business players and to ensure this kind of knowledge. A hybrid trust model based on PKI and reputation-based trust systems is proposed having the following salient

features.

- The PKI based trust model has a third party trust manager that will act as a root CA and authenticate member VOs. These VOs are themselves CAs as they can further authenticate their containing services.
- Selection of services at the the pre-SLA stage is done by using reputation to prevent SLA violation. Services reputation are updated after each SLA validation process.
- SLA views integrate very closely with the trust model to maintain a balance between trust and security. While the trust model promises trust and security, the SLA views protect privacy.

In the following sub-sections, it is elaborated how the best features of both PKI (policy-based approach) and reputation-based trust systems, along with Rule Responder architecture, are utilized to our advantage.

6.3.3.1. Single Sign-On and Delegation

In the proposed model, a third party acts as a root CA. Public Key Infrastructure (PKI) is a popular distributed trust model that offers certificate containing the name of the certificate holder and the holder's public key, as well as the digital signature of a Certification Authority (CA) for authentication. The public keys are distributed among all the trusted parties, packaged in digital certificates, building trust chains. A solution for dynamic ad hoc networks is the inclusion of a *Third Party Trust Manager* acting as a root CA. PKI based trust model with a third party trust manager acting as a root CA is proposed to authenticate member VOs.

This third party trust manager acts as a root Certification Authority (CA) and authenticates member VOs. These VOs are themselves CAs as they can further authenticate their containing services. Each member is given a certificate. Certificates contain the name of the certificate holder, the holder's public key, as well as the digital signature of a CA for authentication. The authentication layer in each VO middle-ware may be based on Grid Security Infrastructure (GSI) [144] where all resources need to install the trusted certificates of their CAs. GSI uses X.509 [86] proxy certificates to enable Single sign-on and Delegation. With Single Sign-On, the user does not have to bother to sign in again and again in order to traverse along the chain of trusted partners (VOs and services). This can be achieved by the Cross-CA Hierarchical [86] [144] Trust Model where the top most CA, called the root CA provides certificates to its subordinate CAs and these subordinates can further issue certificates to other CAs (subordinates), services or users.

6.3.3.2. Reputation Transfer using Trust Reputation Center

Alnemr [25] has presented a reputation-based model that facilitates reputation transfer. One of the main components of this model is Trust Reputation Centers (TRC). It acts as a trusted third party. The TRC is a pool of users' reputation gathered from different

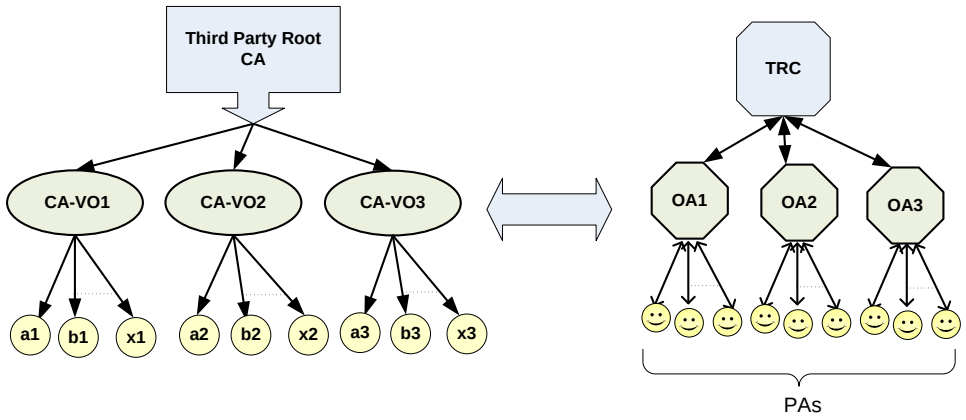


Figure 6.4.: The correspondence between the PKI and reputation based systems and to the Rule Responder architecture

platforms. Each user can have a context-based reputation object (RO). TRC, as a trust third party helps two users from two different organizations to establish an interaction. The proposed hybrid system is fully compliant with the Rule Responder architecture as shown in figure 6.4.

As depicted in figure 6.4, this reputation-based trust model has direct correspondence with Rule Responder's agents and their mutual communication. The PAs consult OAs and OAs in return consult the TRC which is equivalent to the third party CA in PKI based system. In the rest of the chapter, the channel direction flow from PA to OA to TRC, is referred simply as communication among agents.

The word agent in this context refers to a software representation or a smart service. Alnemr in [25] illustrates how Agents can exchange Acquaintance Agent Lists (AAL). An AAL is a list of all previously dealt with trusted agents. The questioner agent upon receiving the list from a friend, cross-references it with the names of its own trusted agents, may update trust values of its aquantances and may issues an inquiry about any agent. The answer is inform of a Reputation Object (RO) that expresses the reputation value given by each agent and the context related to this value. The questioner then takes a decision whether to carry out the transaction with this particular agent or not. There can be various ways to represent trust (e.g. in form of numerical values) and hence the concept of multiple corresponding interpretation or reference models [25] . So the name of the trust model can be used as a reference of what measures trust, and its degree is based upon. [25] has proposed the development of Reputation Reference Trust Models (RRTM) that is used as a parameter when mentioning trust. Therefore the reputation object utilizing all these concepts can be represented [25] as follows:

```
Object Reputation {
TrustMatrix [context][reputation value][RRTM];
Time ValidTime;
```

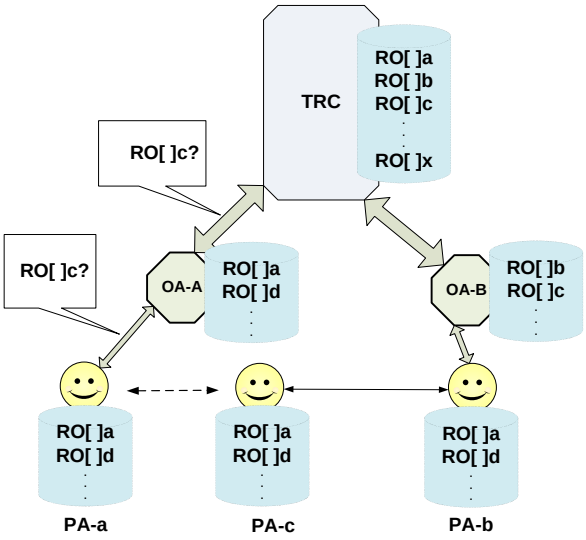


Figure 6.5.: Query of PA-a about reputation of PA-c to OA-A and then redirected to TRC

```
Credentials PresentedCredentials;}
```

In figure 6.5, PA-a that corresponds to *service a* that makes an SLA with an unknown *service c* by checking first its credentials. For this purpose, it consults its corresponding organistational agent, which is OA-A in this case. OA-A too, does not have any information about *service c*'s reputation so it redirects *a*'s query to the trust reputation center *TRC* which then transfers the required reputation object tracing back the same channel.

6.3.4. Rule based Validation of SLA Choreographies

Service Level agreements are frequently validated throughout their life. Runtime Validation ensures that the service guarantees are in complete conformance with the expected levels. WS-Agreement [56] defines a detailed structure of Guarantee Terms with the most important constituents being: Service Level Objectives that express the desired quality of service, Qualifying Conditions that express assertions over service attributes, and Penalty and Reward expressions.

In the proposed rule based validation framework, these terms are represented as logical rules following the RBSLA specifications. These rules are composed together during the process of SLA aggregation. The process of validation is performed by using these rules as distributed queries. During the validation process, queries are decomposed making their premises as subgoals. This backward chaining propagates throughout the SLA Choreography. If all the subgoals are satisfied then the validation is successful.

Due to the consumer-oriented aggregation structure of SLA choreography, a top-down validation framework is proposed. A top-down validation approach has several advantages

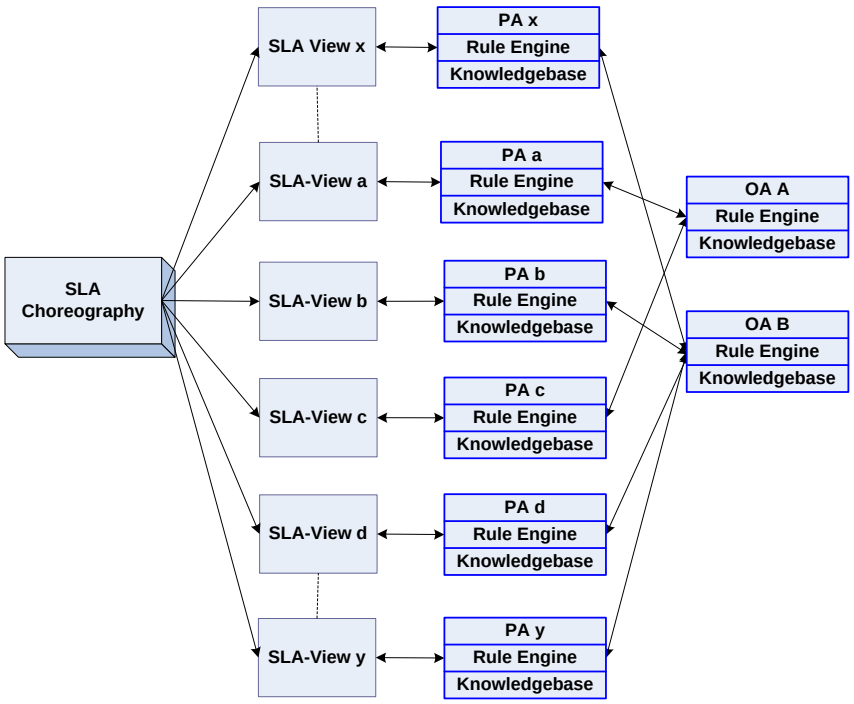


Figure 6.6.: Every SLA-View corresponds to a Personal Agent

in connection with its implementation.

- Interfaces can be validated before going into details of modules,
- In case of a problem on higher levels, one does not need to go into lower levels,
- Since in the view based SLA aggregation, the top level represents the client's perspective therefore this approach can better translate the on-demand validation queries initiated from the client.

Figure 6.6 depicts how the Rule Responder and SLA-Views work together to enable this scheme.

Each SLA-View that in fact represents a service provider in the SLA Choreography, is connected to a Personal Agent (PA). SLA choreography is composed of various SLA views. A PA receives queries from the Organizational Agent (OA) and having complete information of its consumer oriented SLAs in its knowledge-base, performs the local validation and delivers back the responses on behalf of the service providers.

The complete request pattern starting from the External Agent has been depicted in figure 6.7. OA intercepts the query at the boundary of a VO and redirects it towards the corresponding PA. Rule Responder architecture supports various multi-agent communication protocols including Agent Communication Language (ACL) [59]. The trust model facilitates the distributed query to travel across various domains through a single sign-on

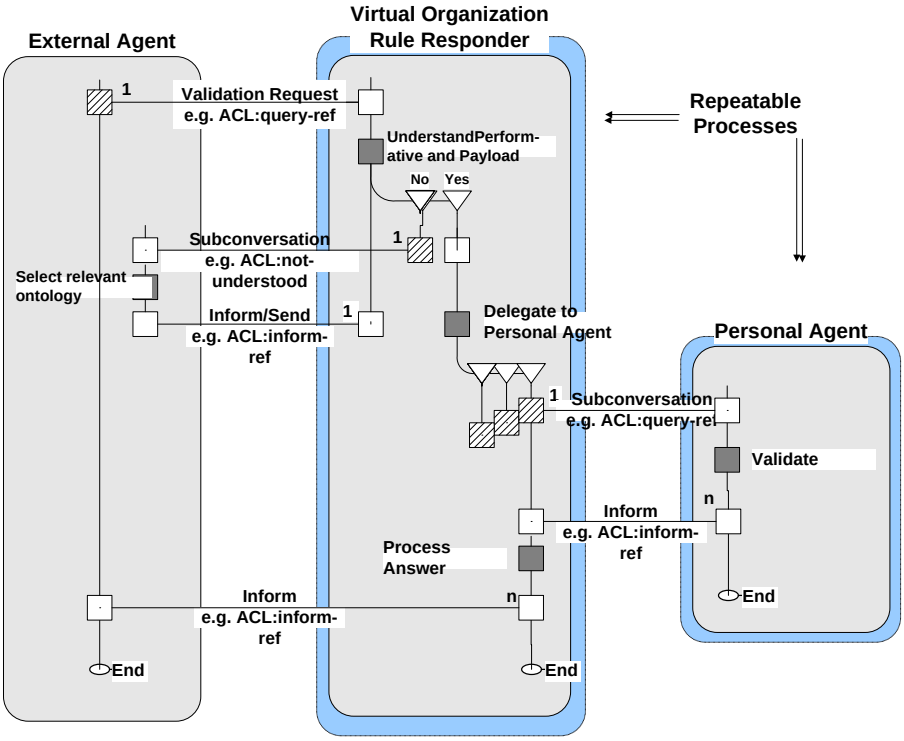


Figure 6.7.: Role Activity Diagram for a simple Query-Answer Conversation

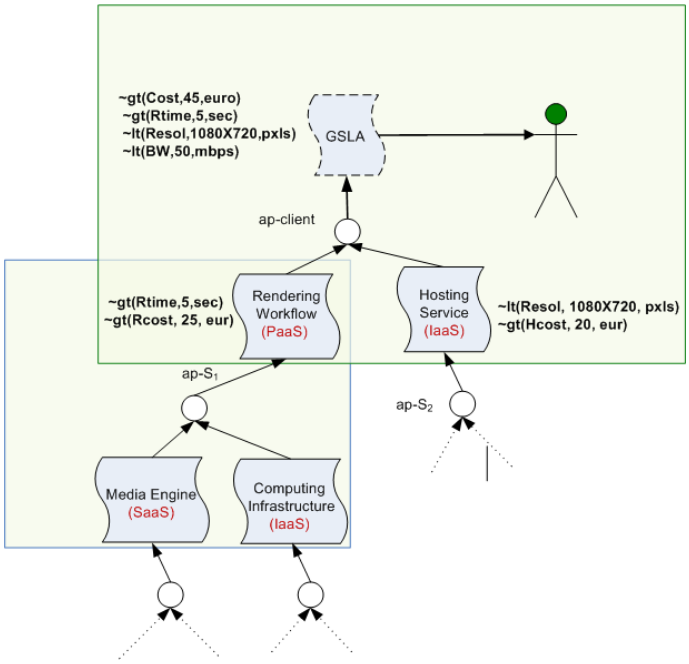


Figure 6.8.: Running Example - Hierarchical SLA Validation

and delegation mechanism. Referring to this multi-agent architecture coupled with the notion of SLA Views and the distributed trust, the validation process is termed as the *Delegation of Validation*.

6.3.5. Delegation of Validation

The aggregation of SLAs is a distributed mechanism and the aggregation information is scattered throughout the SLA choreography across various SLA views. To be able to validate the complete SLA aggregation, the validation query is required to traverse through all the SLA views lying across heterogeneous administrative domains and get validated locally at each SLA view. The multi-agent architecture of Rule Responder provides communication middle-ware to the distributed stakeholders namely the client, the VOs and various service providers. The *Delegation of Validation* process empowered by the *single sign-on and delegation* properties of the distributed trust model, helps the distribute query mechanism to operate seamlessly across different administrative domains.

Now its time to explain the Guarantee Terms of a WS-Agreement, expressed as rules, are transformed into distributed queries. Section 5.3.3 explains how the aggregation functions are applied on the basis of aggregation type of a service term, identified by $type_a$ attribute. SLOs can also be aggregated as conjunctive premises of derivation rules. It is also important to realize that the SLOs refer to an established SLA and their ranges are meant to be guarded in order to maintain desired levels of service.

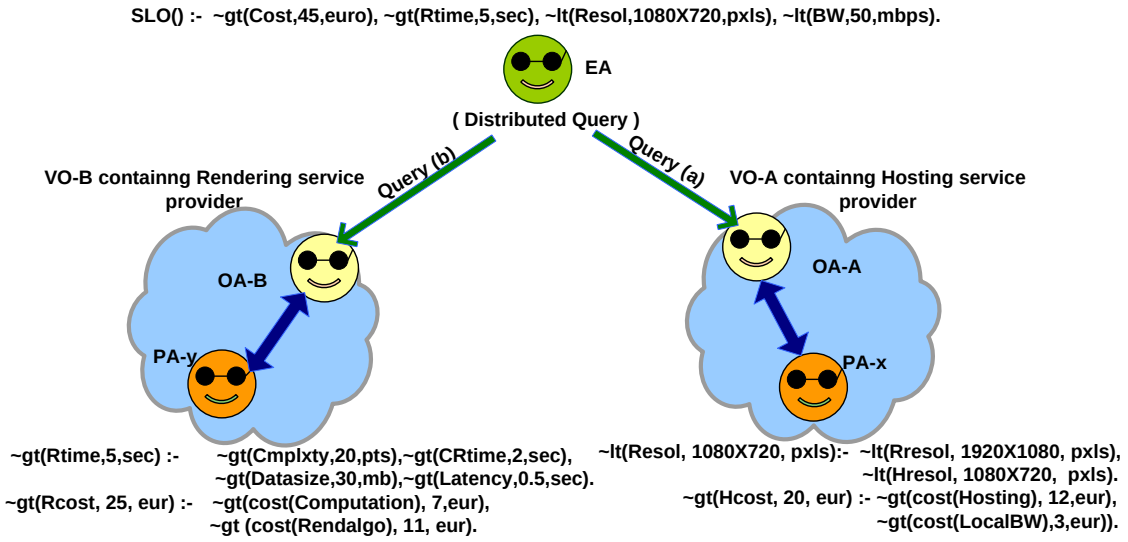


Figure 6.9.: Validation through distributed query decomposition

6.4. Running Example – Hierarchical SLA Validation

Coming back to the running example, the user needs to carry out hi-tech multi-media operations such as rendering and editing. She plans to utilize online services to accomplish these tasks. The SLA-Choreography resulting from this simple scenario is shown in Figure 6.8.

In the scenario, the user is interested to render her videos and then host them on the web. Her requirements include a maximum cost of 45 Euros, maximum response time of 5 seconds, minimum resolution of 1080X720 pixels and the minimum bandwidth (from hosting service) of 50 Mbps.

In Figure 6.9, this scenario has been depicted from validation point of view. The user requirements are shown on the top of the figure, expressed as a derivation rule composed of SLOs of the final aggregated SLA. It must be noted that in Figure 6.9, it has been intentionally chosen to represent these rules in a highly abstract format. This is only for the convenience of reading and comprehension. However later in this section it will be explained how to formally represent and implement these rules.

The agents OA and PA in the Figure 6.9 representing the Rule Responder architecture, are shown to automate the distributed query processing. For the sake of simplicity, the Rule Responder architecture has been skimmed just from agent-oriented perspective, and various essential details such as the Rule-bases, the knowledge resources and the role of Enterprize Service Bus (ESB) have been abstracted. The predicates *lt* and *gt* denote lesser-than and greater-than respectively. The user requirements are expressed as a set of premises in the following derivation rule:

```
SLO() :- ~gt(Cost,45,euro), ~gt(Rtime,5,sec),
         ~lt(BW,50,mbps),   ~lt(Resol,1080X720,pxls).
```

It should be noted that in accordance with the WS-Agreement standard, there are three arguments in each SLO, denoting: the SLO name, its value and its unit respectively. During the validation process, this rule will be decomposed such that each premise will become a subgoal. This subgoal will be sent as a message to the PA corresponding to the next SLA view in the hierarchy where it will emerge as a conclusion of one of the rules in the local rule set, thus forming a distributed rule chain. The initial steps of decomposition procedure are depicted at the bottom of the figure. In the figure, Organizational Agents (OA) have been shown to receive and track the distributed query whenever it enters a new VO. For each service provider, there is a Personal Agent (PA). A PA, after finishing its job, should report to the corresponding OA that will redirect the distributed query to the service provider's PA that comes next in the hierarchical chain. Alternatively, depending upon the organizational policies, the PA can communicate with the next PA directly. The process continues until the query has found all the goals expressed in terms of logical rules. Active rules tracking these goals or SLOs, are then invoked locally within the administrative domains of the corresponding SLA views. The true or false results are conveyed back following the same routes.

To validate all the guarantee terms of the final (client's) aggregated SLA, the aggregation chunks within all the SLA Views, scattered through the whole SLA Choreography, are required to be validated. In the scenario, OA-B receives a subgoal $\sim gt(Rtime, 5, sec)$ representing the requirement that the total response time of the system should not be more than 5 seconds. This SLO depends on several factors such as the complexity of the rendering algorithm, size of the data, latency and response time of the computational hardware which is expressed as the new subgoal:

```
~gt(Rtime,5,sec) :- ~gt(Cmplxty,20,pts), ~gt(CRtime,2,sec),
                    ~gt(Datasize,30,mb), ~gt(Latency,0.5,sec).
```

The SLO expressing the cost will be divided between the two service providers as shown in the Figure 6.9. The service cost at the level of OA-A should be less than 20 and is dependent on the sum of the cost for hosting and the cost for local bandwidth. The varying upper limit of cost at different levels reflect the profit margins of different providers e.g. the provider in OA-A has a profit margin of 5 Euros.

As it has been discussed earlier, the rules shown in the Figure 6.9 have been highly abstracted for reading convenience. In practice, one needs to take into consideration many additional details. To highlight these issues, lets begin with the formal representation of the SLO state that CRtime should be less then 2 seconds:

```
slo(Serv2, CRtime, <2, sec)
```

Serv2 is the name of the service with which this SLO is associated. Every SLO must have a reference point similar to Serv2. This particular SLO represents a state that is initiated if there is an event CRtime, which is a variable bound to a measurement value, which is greater then 2 seconds:

```
 initiates(CRtime, slo(Serv2,CRtime,<2,sec), T) :- CRtime < 2.
terminates(CRtime, slo(Serv2,CRtime,<2,sec), T) :- CRtime >= 2.
```

These two lines describe the initiation and termination of the SLO state. The SLO itself is associated with a specific service Serv2 and describes the user's requirement that the response time of the service should not exceed 2 seconds. In other words, if the response time is lower than 2 seconds, the SLO is fulfilled, if it is greater than 2 seconds, the SLO is violated. The event is the measurement of CRTime at a particular time point such as:

```
happens(CRtime,T):- sysTime(T), ping(Serv2,CRtime).
```

Since CRtime is an event, one needs to make it happen. In this case, in the happens rule one simply measure the response time in terms of systems time lapsed by pinging the service. It is now possible to ask queries if the SLO state holds at a particular point in time or not (i.e., violation of the SLO):

```
holdsAt(slo(Serv2,CRtime,<2,sec), 2001-10-26T21:32:52.12679)?
```

The result is true or false depending on the measurement result in the happens event rule. It is now possible to define SLO state processing rules such as SLO Rtime, which is the response time of CRtime.

```
holdsAt(slo(Serv1,Rtime,<5,sec),T) :-
                          holdsAt(slo(Serv2,Cmplxty,<20,pts),T),
                          holdsAt(slo(Serv2,Datasize,<30,mb),T),
                          holdsAt(slo(Serv2,Latency,<0.5,sec),T),
                          holdsAt(slo(Serv2,CRtime,<2,sec), T).
```

and ask if this derived SLO it violated at a certain point in time,

```
not(holdsAt(slo(Serv1,Rtime,<5,sec), 2001-10-26T21:32:52.12679))?
```

The delegation of validation, continuing across various levels, reaches the SLA views originating the corresponding SLOs, and the SLOs get validated there. At each level, the corresponding reward and penalty conditions are also checked and if required, appropriate action is taken. The distributed Rule Responder agent architecture acts as an enabling technology for the SLA Views concept. One of its important features is that we can implement principles of autonomy, information hiding and privacy with the agent approach. For instance, the details how a particular service level objective is measured and computed in a personal agent might be hidden (e.g. a third-party monitoring service) and only the result if the service level is met or not might be revealed to the public. Another important aspect is that the monitoring/validation might run in parallel, i.e. several service provider (PAs) might be queried by an OA in parallel via messaging. For instance, a complex SLOs might be decomposed by the OA into several subgoals which are then sent in parallel to the different services (PAs) which validate them.

Qualifying Conditions and penalty and reward expressions can be expressed through Event Condition Action (ECA) rules. For example, if one wants to express the statement "If the response time of the service named "Serv7" is larger than 60 seconds then there is a penalty of 5 Euros", one can write its equivalent in WS-Agreement as follows:

```
<wsag:Penalty>
  <wsag:AssesmentInterval>
    <wsag:TimeInterval> 60
    </wsag:TimeInteval>
    <wsag:Count> 1 </wsag:Count>
  </wsag:AssesmentInterval>
  <wsag:ValueUnit> Eur </wsag:ValueUnit>
  <wsag:ValueExpr> 5 </wsag:ValueExpr>
</wsag:Penalty>
```

This can also be represented by ECA rules:

```
timer(sec,T) :- Timer(T), interval(1,min).
event(Serv7,Violate) :- ping(Serv7,RT), RT>60.
action(Serv7,Penalty) :- penalty(Serv7,Obligation,5).
```

Now combining together and generalizing for any service x:

```
ECA(?x, Monitor) :- timer(sec,T),
                    event(?x,violate),
                    action(?x,penalty).
```

The above rule is activated according to the timer(sec, T) which is defined by the following rule, invoked after every minute:

```
timer(sec,T) :- Timer(T), interval(1,min).
```

In case of detected SLA violation, two actions are taken:

1. The penalty enforcement rules are activated and a fine is imposed on the violating service.

2. The reputation value of the violating service is decreased as an additional fine.

Similarly the reputation based trust can complement with the validation system for the reward conditions. For instance for consistently complying services, the reputation of the service can be increased as an implementation of reward points. In case of failing services, the alternate can be selected on the basis of highest reputation value among the competitors.

Similar approach can be used for the renegotiation, fault tolerance and breach management processes. During renegotiation, the distributed query traverses in the same way towards the service providers, offering those terms which are desired to be renegotiated. During fault tolerance and breach management, violations are localized through a similar invocation of the distributed query. The combination of ECA rules and using derivation rules to implement the different parts of an ECA rule provides high expressiveness and can be very easily transformed in a rule based markup language such as RuleML [32]. RuleML allows to declaratively implement the functionality of each part of a Reaction Rule (event, condition, action etc.) in terms of derivation rule sets (with rule chaining), thus making them processable in autonomic and autonomous way.

6.5. Role of Validation and Trust Model During Service Selection

Reputation transfer is required at two stages: at service selection stage and at penalty enforcement stage. In the process of service selection, the reputation transfer helps to select the least violation-prone services, taking into account proactive measures to avoid SLA violations. Out of all the available services, the client (which is also a service in this case) first filters the best services complying its "happiness criteria" as formalized in 4.3.1.1.8. Then the client compares the credentials from reputation objects of the services and selects the best service in accordance to its already devised criteria. Out of redundant services which fulfil client's requirements, the service with the highest reputation is selected.

6.6. Summary

This chapter elaborates a multi-agent, rule-based validation framework based on the *Delegation of Validation* approach and a third party hybrid trust management system that employs PKI and reputation based systems. The third party hybrid trust management system complements the validation framework as its PKI mechanism provides single sing-on and delegation to the distributed query making it interoperable across heterogeneous enterprize boundaries. The reputation based trust weaves together trusted services and become very important during the fault tolerance process as it may require service selection after a service failure within the choreography. Fault tolerance mechanism can then utilize the two-phase service selection algorithm and the negotiation/renegotiation protocol described in Chapter 4, to quickly come up with alternate services.

Chapter 7.

Implementation

Love is the essence of deeds.

(Hazrat Ghulam Rehman R.A)

This chapter provides the implementation details of two components:

- a simulation of a two-phase service selection algorithm based on branch and bound and heuristic approaches, and
- a prototype for rule based distributed management system that highlights various challenges of the aggregation and validation of SLA Choreography.

The results of the implementation are presented and analyzed in this chapter.

7.1. Optimization of SLA-Based Service Selection

7.1.1. Use Case Scenario

A generalized version of the scenario presented in the running example is taken as a use case here. As shown in Figure 3.3, the user specifies the steps of the operation as a workflow. For each step, the user must indicate the type of service needed to complete it, and some quality of service (QoS) parameters the service must or should satisfy. Service providers are providing services which can be categorized into the types/classes specified by the user and which have given QoS characteristics. The goal is to devise an algorithm which automatically picks specific services of the types requested by the user, satisfying all the "must" requirements and fulfilling the "should" requirements as well as possible. The algorithm should also be able to adapt to changes in user requirements and to service failures. A first step towards devising such an algorithm is to create a mathematical model for the problem specifying the service types, their QoS requirements in terms of "must" and "should" constaints, and the global constraints i.e. total cost K and total time t. A branch and bound algorithm is then applied to search for the service combinations with the maximum Happiness Measure. The heuristic-based algorithm is then tested on the basis of changing user requirements. The results of the branch and bound algorithm and the heuristic algorithm are then compared. Another test is made to check the behavior of the heuristic algorithm in case of failing services. An analysis of the results is then presented.

7.1.2. Simulation Setup and Tools

For branch and bound algorithm, initially it was planned to spawn threads/processes dynamically, one for each node in the branch and bound search graph. Then it was decided to make a parallel version of the algorithm to run on various nodes on the network. For these requirements several tools were used.

7.1.2.1. Kepler

In the implementation, to model the different components of the optimizer, Kepler (`http://www.kepler-project.org/`), a grid workflow tool written in Java was used.

The initial plan was to write everything in Kepler. However, after realizing that Kepler can only spawn a fixed amount of threads/processes (one for each component in the Kepler workflow), it was decided to use an external process for the optimizer core, leaving only I/O to the Kepler workflow. The structure of the workflow models the real-world data flow: there are subworkflows for the user interface, a service manager, the optimizer and a report module. In the simulation, simple dialogs or files were used for input. In the real world, one would have a user-friendly graphical user interface and the service manager would communicate with third-party services.

7.1.2.2. Optimizer in C++ with QtCore

As explained above, the core optimizer is implemented as an external process invoked from the Kepler workflow. It is written in C++ using `libQtCore`, the non-GUI portion of the Qt 4 (`http://www.qtsoftware.com/products/appdev`) library. Qt is best known as a GUI toolkit, however it also contains convenient utility classes, which were decided to be used in the implementation. The `QtCore` module also works without any graphical interface.

The communication with the Kepler workflow is done through the standard file descriptors `stdin`, `stdout` and `stderr` and through the exit code: 0 if a solution was found, 1 if the problem is infeasible.

Heuristics and feasibility tests can be enabled or disabled at compile time, through the use of the `#define` preprocessor directive.

The first implementation was based on threads. As threads have high overhead and as they mean effectively leaving the search strategy to the operating system (because the thread scheduler decides which node in the search graph to process next), which knows nothing about the structure of the problem, a deterministic sequential version was also implemented using breadth-first search. The optimal solution for a single machine would be to use one thread for each CPU core and a breadth-first search queue in each of those threads, however the efforts were focused on the distributed version.

7.1.2.3. Distributed version with TAO

As threads on a single machine do not show the full picture of parallelization, A parallel version of the algorithm using TAO (`http://www.cs.wustl.edu/~schmidt/TAO.html`), a distribution of CORBA was implemented.

In the distributed version, a master process talks to Kepler as if it were the whole optimizer. When starting up, that master process spawns child processes on different machines (or on the same machine – as there is only one computing thread in each process in the current implementation, it may be useful to run several instances on a multi-core machine) using SSH. The master process contains a CORBA object representing the optimizer as a whole, the child processes call back to that object to request the global input data and to announce their readiness.

A spawner, another CORBA object, is started in each process. Its role is to manage the queue of decision tree nodes for that process. (Each decision tree node is also represented as a CORBA object.) The spawner runs in its own thread, separately from computation. The computing thread processes the decision tree node as they're added to the queue. When it has to branch, each newly branched node is sent to the spawner of a random process and enqueued by that spawner. The resulting search strategy is a distributed generalization of breadth-first search.

The ideal strategy would also run multiple computation threads for each process on a multi-core machine, however it was decided to keep it simple. Thus, in the current implementation, multiple cores can only be used by running multiple processes on the same machine. Use of one thread per core could bring an additional performance boost.

7.1.3. Performance Analysis

7.1.3.1. Branch and Bound Speedup

A classical speedup analysis for the branch and bound implementations comparing the serial, single-machine version with the parallel, distributed version was performed.

7.1.3.1.1. Test Method Some feasible, but otherwise random test cases with a pseudo-random number generator were produced. Tests with 25, 28, 30 and 32 workflow nodes were performed, each of which was mapped to a different service class. The tests were performed with 5 services per service class and 10 attributes per service class (i.e. 10 attribute values per service).

The test was run with both the serial and the parallel version. For the latter one, the number of slave processes were varied from 0 to 4, i.e. the total number of processes from 1 to 5.

The tests were run on a dedicated simulation environment consisting of a heterogeneous network of four 3 GHz Xeon machines, two of which have two CPU cores each, the other two only one. All four machines have 1 GB of RAM each. Due to the limited memory of these machines, some problems were faced in the largest test case of 32 service classes and could run only up to 4 processes for that test case.

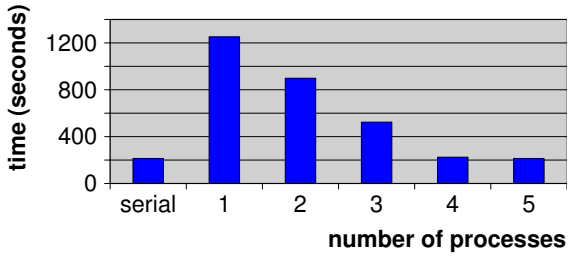


Figure 7.1.: Speedup analysis with 25 service classes

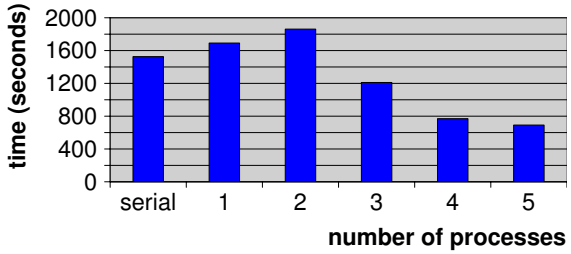


Figure 7.2.: Speedup analysis with 28 service classes

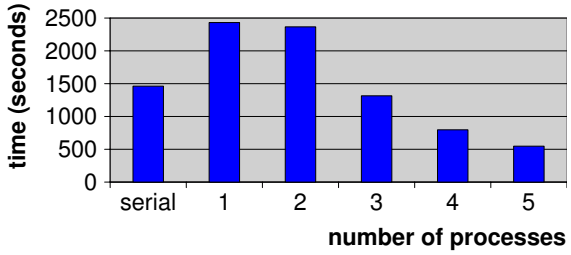


Figure 7.3.: Speedup analysis with 30 service classes

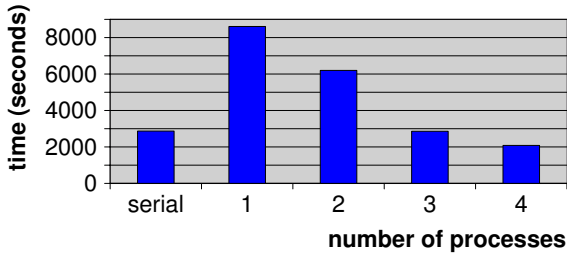


Figure 7.4.: Speedup analysis with 32 service classes

7.1.3.1.2. Results Figure 7.1 summarizes the results with 25 service classes. Unfortunately, in this case the overhead from CORBA made parallelization worthless. This effect is even stronger with smaller test cases. However, luckily, it disappears as one increases the dimension.

The results for the test cases with 28, 30 and 32 service classes are shown in figures 7.2, 7.3 and 7.4 respectively. The fact that the 30-node test case is solved faster than the 28-node one in some configurations is not a measurement glitch: the computation time strongly depends on the individual test case, including the bounds for K and T, and the distributed implementation is also not completely deterministic due to process scheduling and network latency effect. One can see that once we reach a sufficiently large computation time, the distributed implementation overtakes the serial one and a significant speedup can be measured. The trend shows that as the problem size grows, the performance behavior appears to converge to a practically linear speedup.

It shall be noted that once the exact optimum is computed, it is possible to react to dynamic changes producing near-optimal solutions within a few milliseconds by using the heuristic update we sketched.

7.1.3.2. Branch and Bound vs Heuristics

7.1.3.2.1. Test Method The branch and bound implementation was compared with the updating heuristics to see how much faster the heuristics are.

Feasible but otherwise random synthetic testcases were generated with a pseudorandom number generator. Tests with 10, 12, 15, 20, 25 and 30 workflow nodes were made, each of which was mapped to a different service class. The timings reported for the branch and bound algorithm are from a sequential breadth-first implementation, which proved the most efficient on a single CPU.

For each testcase, a series of tests was run with constant services and user requirements, changing only the values of K and T (one at a time), and the results of the heuristics were compared with repeated runs of the branch and bound, looking at both the execution time of the optimization process and the quality of the solution (i.e. how close to the optimum it is).

The tests were run on a single-core 2.6 GHz Pentium 4 with HyperThreading disabled.

The following pairs (K, T) were used for each testcase:

10 nodes: $(350, 500)$, $(400, 500)$, $(400, 550)$, $(400, 500)$, $(380, 500)$, $(350, 500)$,

12 nodes: $(400, 600)$, $(450, 600)$, $(450, 650)$, $(450, 600)$, $(430, 600)$, $(400, 600)$,

15 nodes: $(550, 700)$, $(600, 700)$, $(600, 750)$, $(600, 700)$, $(580, 700)$, $(550, 700)$,

20 nodes: $(800, 850)$, $(850, 850)$, $(850, 900)$, $(850, 850)$, $(830, 850)$, $(800, 850)$,

25 nodes: $(900, 1200)$, $(950, 1200)$, $(950, 1250)$, $(950, 1200)$, $(930, 1200)$, $(900, 1200)$,

30 nodes: $(950, 1500)$, $(1000, 1500)$, $(980, 1500)$, $(950, 1500)$.

7.1.3.2.2. Results Figure 7.5 shows the results of the performance measurements. Almost invisible bars in the figures mean the value is very small or zero. The results show

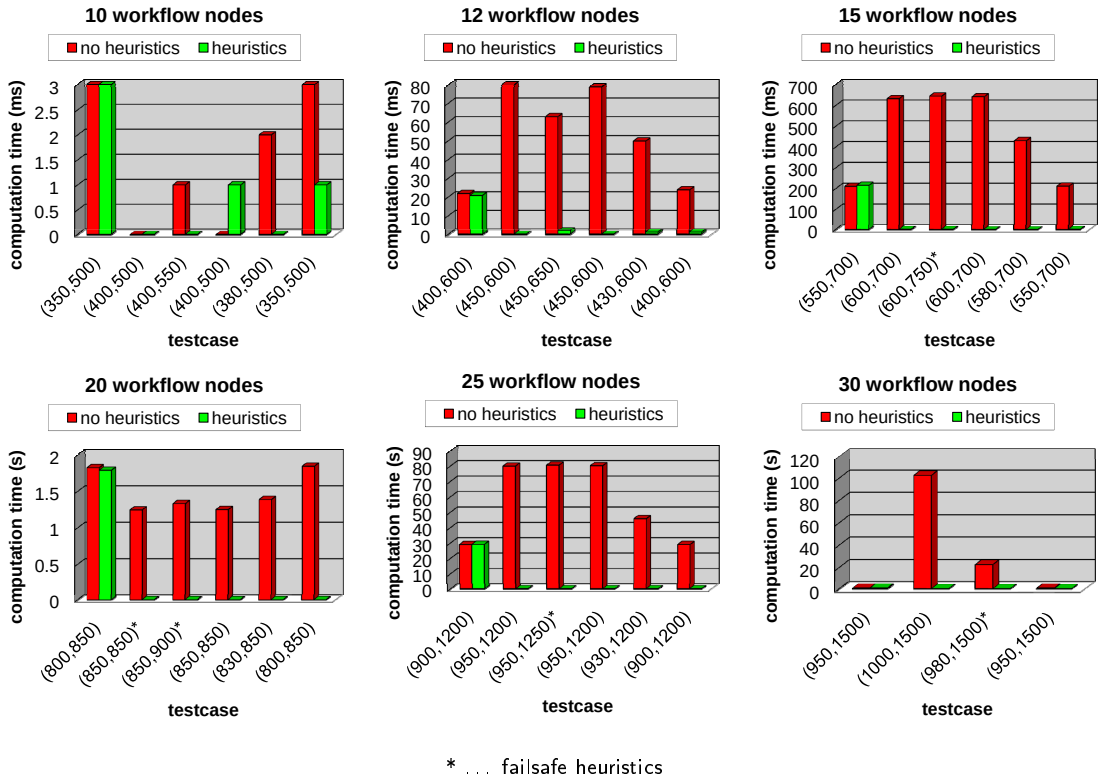


Figure 7.5.: Performance comparison branch and bound vs. heuristic update

that the branch and bound algorithm scales up to problem sizes in the order of 30 workflow nodes with acceptable performance, but that the heuristic update is several orders of magnitude faster. Note that the first testcase of each set is the initial solution, which is always computed using branch and bound.

It was also found that in the testcases with 10 and 12 workflow nodes, the heuristic updates always found the optimum solution. This is not always guaranteed, because the update is only a heuristic. Indeed, for the testcases with 15 or more workflow nodes, the solutions found by the heuristic approach were not always optimal, but they came very close (within 98% of the happiness) to the optimum. Figure 7.6 shows the ratios between the happiness values for the solutions found by the heuristics and the optimum happiness values as found by branch and bound.

An experiment was also tried using only the heuristics instead of the branch and bound process, using the solution without constraints for K and T as the starting solution. This turned out to be much faster than branch and bound, which matches the expectations, as the heuristics are polynomial, whereas the branch and bound is exponential. The initial heuristic updates are as fast as the subsequent ones. Unfortunately, this is only preliminary due to the problem that the heuristic update can fail if the update to reduce the value of

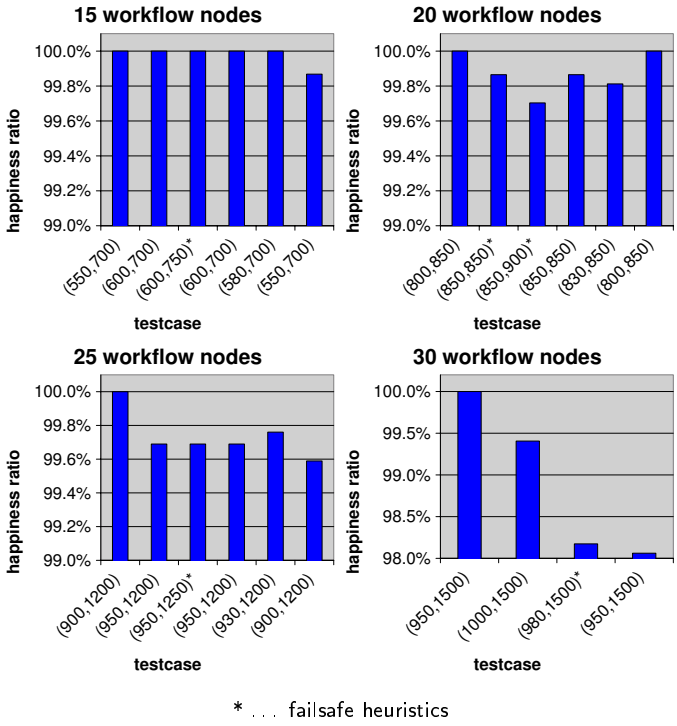


Figure 7.6.: Happiness ratio between heuristic solution and optimum

one constraint makes it exceed the bound for the other one, the "failsafe" version can fail if a more expensive service needs to be picked to reduce the computation time or vice-versa, and there is no guarantee of optimality. However, these problems are inherent to heuristics.

A solution was also presented to the well-known problem of user-centric optimization of service composition and our approach shows its qualities in the second phase by demonstrating very promising results for real-time response to changing user requirements. Even more, in practical use the two-phase algorithm as a whole showed an acceptable runtime behavior, justifying it to be a working solution to the workflow optimization problem.

7.1.3.3. Runtime Service Failures

7.1.3.3.1. Test Method The testcase with 10 service classes was reused with 5 services per service class and 10 attributes per service class. The services and user requirements were kept constant, including the bounds $K = 350$ and $T = 500$ and the design-time optimum was computed using the sequential branch and bound algorithm. In the testcase, the node number n is assumed to require the service class number n (without loss of generality, as one can number the service classes in any arbitrary way – also please note that this is *not* a requirement of the algorithm, in fact it is also possible to use the same service class for more than one node, or any other arbitrary assignment).

The successive failure of several services were then simulated. It was also assumed in

this testcase that the workflow is linear, which means that the services before (i.e. with a lower node number than) the failed service have already completed and cannot change anymore, whereas the nodes with a higher number can still be changed to use a different service of the requested class. (This is *not* a requirement of the algorithm, it is possible to define an arbitrary set of immutable workflow nodes at the point of the service failure.) The first number is for 0 failures, i.e. all services work. Then the failures of the services were simulated in the following order:

1. service 1 from class 1, i.e. for node 1 (the second node, as the counting starts at 0)
2. service 3 also from class 1 (thus there are only 3 choices left for node 1)
3. service 3 from class 3, i.e. for node 3
4. service 4 from class 5, i.e. for node 5
5. service 2 from class 8, i.e. for node 8

The first 4 failures were chosen such that the service to be dropped was used in the optimum solution, for the last one, an arbitrary one was picked to check that the heuristic will not try to change the solution if there is no need. So this leads to the following lists of fixed nodes (i.e. nodes one cannot touch anymore because the services have already completed):
0 failures:
1 failure: 0
2 failures: 0
3 failures: $0, 1, 2$
4 failures: $0, 1, 2, 3, 4$
5 failures: $0, 1, 2, 3, 4, 5, 6, 7$
and the following lists of failed services (class,service pairs): 0 failures:
1 failure: $1, 1$
2 failures: $1, 1, 1, 3$
3 failures: $1, 1, 1, 3, 3, 3$
4 failures: $1, 1, 1, 3, 3, 3, 5, 4$
5 failures: $1, 1, 1, 3, 3, 3, 5, 4, 8, 2$.

The updating heuristic was used for the updates.

7.1.3.3.2. Results The graph shown in Figure 7.1.3.3.1 presents the results obtained.

In the graph, the dashed horizontal lines are the K and t constraints, the continuous lines are the total cost, the total time and the happiness for the solution. The happiness must be read on the right axis, everything else on the left one. (Separate axes are used because otherwise the happiness curve would be too flat to be able to see anything.) The computation time is less than 2 ms for each of the updates.

It must be noted that, as the updates were done using a heuristic approach, optimality is not guaranteed, only approximated. That explains why the happiness actually goes slightly up with the fourth failure: This means the solution which was computed after the third failure was suboptimal.

failures	happiness	cost	time	K	t
0	23.453121	349.08014	457.269835	350	500
1	22.787904	340.240413	467.319084	350	500
2	21.889761	331.958092	482.282088	350	500
3	19.630979	343.473266	488.363885	350	500
4	20.073241	349.925362	472.993888	350	500
5	20.073241	349.925362	472.993888	350	500

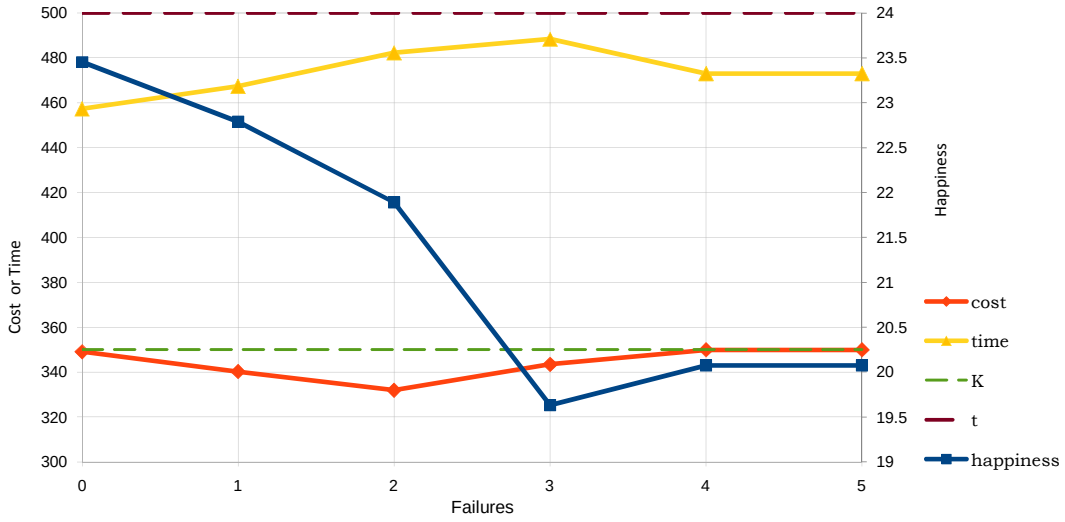


Figure 7.7.: Effects of Service Failures

7.2. Validation of SLA Choreographies

7.2.1. Use Case Scenario

Moving forward with the running example, the user needs to carry out hi-tech multi-media operations such as rendering and hosting videos. She plans to utilize online services to accomplish these tasks. The SLA-Choreography resulting from this simple scenario has already been described in Figure 6.8. Typical SLOs relevant to the use case concern the availability, bandwidth, resolution and response time.

7.2.2. Assumptions

The prototype for the validation of SLA Choreographies has been designed on the basis of following assumptions.

- The aggregation of the SLA Choreography is already complete and the aggregated SLAs are represented as distributed set of rules, whose different parts are scattered across respective partners.

- A third party trust manager is assumed to foster trust among interacting partners. In this context, the third party root trust manager has been assumed to construct a

Virtual Enterprize Organization (VEO), which will be considered as the root of the distributed system.

- In the prototype implementation of the scenario, only some of the SLA parameters have been considered, out of which, selected SLOs have been picked up for the simulation.
- There are multiple layers or agents with their rule-bases. These rule-bases consist of rules representing different SLOs to provide a proof of concept that heterogeneous knowledge-bases can work together. The final layer actually implements an elaborate version of the rules expressed in Prova.
- The overall system is a prototype for proof of concept of the proposed framework. A detailed implementation can be carried out, moving forward on the same lines.
- The SLA information including exact facts and rules are hidden within the knowledge-bases of their respective agents. The actual validation and penalty enforcement is done locally within agent's premises but to highlight the hierarchical nature of query processing, a unified interface has been developed for this prototype. This unified interface is just for the sake of demonstration and does not imply the actual realization of the validation framework.

7.2.3. Simulation Setup and Tools

The prototype implementation is based on Rule Responder architecture. Rule Responder employs a suit of technologies to coordinate among various components within its architecture. In addition to Java Servlets, the technologies used in Rule Responder are described as follows.

7.2.3.1. Mule ESB

Mule Enterprize Service Bus (ESB) [97] supports various protocols and facilitates many business topologies for component organization.

7.2.3.2. RuleML

RuleML [32] provides an XML based rule exchange format to exchange rules among different (sub)systems especially on Internet. RuleML can conveniently represent logical statements in XML format. It is a very extensive language and uses a variety of tags. A few tags which will be used in the prototype are introduced below.

- For representing relations or predicates it uses <Rel>.
- For implication it uses <Implies>.
- To declare a variable it uses <Var>.

- Constants are represented using <Ind>
- A combination of <Expr> and <Fun> tags, is used to build complex terms, for instance for building a data structure for the hierarchical arrangement of various SLOs. <Fun> denotes a function and <Expr> is interpreted as an expression in this regard.
- A logical sentence is wrapped between <Implies> tags and has two parts i.e., head of the logical statement and the body of the logical rule represented by <head> and <body> tags.
- A predicate is enclosed between <atom> tags

As an example with a body 'and-ing' two atoms, consider the English sentence:
"The discount for a client renting a service is 20 percent if the client is also a business partner and the service is basic."
It can be marked up as the following RuleML (implication) rule:

```
<Implies>
  <head>
    <Atom>
      <Rel>discount</Rel>
      <Var>client</Var>
      <Var>service</Var>
      <Ind>20 percent</Ind>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <Rel>partner</Rel>
        <Var>client</Var>
      </Atom>
      <Atom>
        <Rel>basic</Rel>
        <Var>service</Var>
      </Atom>
    </And>
  </body>
</Implies>
```

RuleML is used a standard rule exchange format in the prototype. The queries are exchanged among the client and various components in RuleML format. Even if the rules are represented in some other format, for exchange they must be transformed in RuleML. For example the functions for Prova-to-RuleML and RuleML-to-Prova are used inside OA.

7.2.3.3. POSL

POSL [13] integrates Prolog's positional and F-logic's slotted syntaxes for representing knowledge (facts and rules) in the Semantic Web. The same logical statement, which was chosen as an example represented in RulML format can be written in POSL as follows:

```
discount(?client,?service,percent20) :- partner(?client), basic(?service).
```

In Rule Responder, POSL uses OOJDrew rule engine [28] to execute its rules.

7.2.3.4. Prova

Prova [82] is an open source rule language for reactive agents and event processing in Java and is designed to work with Enterprize Service Bus (ESB). Prova can integrate with Java. This has made it a very attractive choice to design agents in Rule Responder. In the presented prototype several PAs have been implemented as Java servlets with their knowledge-bases programmed in Prova and POSL.

A sample Prova rule stored in the knowledge-base of OA for sorting out the right PA, sending it message and receiving its answer in order to validate an SLO is shown below.

```
getBandWidth(XID,Topic,Request,Contact):-
   % Retrieve the responsible PA (Agent) for the Topic
   assigned(XID,Agent,Topic,veo_responsible),
   % Send the query to the PA
   sendMsg(XID,esb,Agent, "query", slo(BandWidth)),
   % Receive the answer(s)
   rcvMult(XID,esb,Agent,"answer",Contact).
```

7.2.4. Architecture

The architecture of the system has been depicted in Figure 7.8.

In Figure 7.8, three types of agents i.e., the External Agent (EA), the Organizational Agent (OA) and (four) Personal Agents (PAs) have been depicted to be connected together. Mule Enterprize Bus (ESB) perceives different components as end points of the system and keeps their information in mule-all-config.xml file. The EA is a thin client implemented as an html file, which can post a RuleML query whose answer is also wrapped in the RuleML format. All the four PAs shown in Figure 7.8 are implemented as Java Servlets equipped with their respective rule-bases. The messaging among the PAs utilizes HTTP protocol and the RuleML is used as the standard rule exchange format. The PAs Rendering Workflow and Hosting form the first layer of PAs and have their knowledge-bases implemented in POSL that uses the OOJDrew rule engine. The second layer of PAs constitute of the Media Engine and the Computing Infrastructure and their knowledge-bases are implemented in Prova.

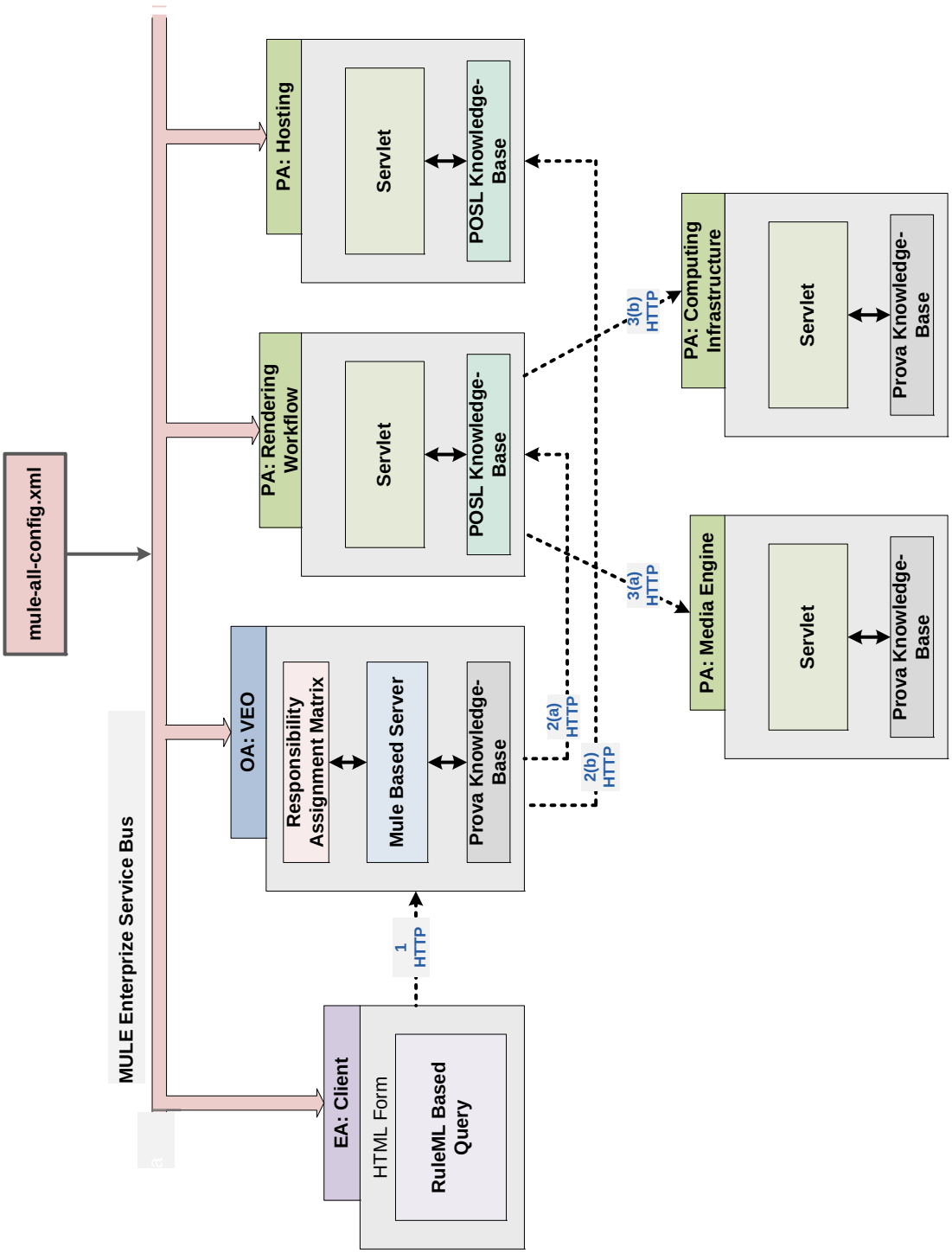


Figure 7.8.: Prototype Architecture)

7.2.5. The Validation Process

The query is intercepted by the Mule server and is directed to the OA to be handled there. The OA works like a post office. It keeps the information about all its directly subordinate PAs in a file called Responsibility Assignment Matrix (RAM) implemented in OWL-lite. RAM maintains the hierarchy of PAs, their responsibilities, accountabilities etc. Upon receiving any query, OA's knowledge-base (implemented in Prova), determines by looking in its RAM, which PA this query should be directed to and then sends an appropriate Prova message to that particular PA. The PA interprets the message with the help of its knowledge-base and upon finding out that it can not furnish all the required information, redirects it to the next layer of PAs. The second layer of PAs consisting of Media Engine and Computing Infrastructure simulates the validation process through its Prova based rules. A text file populated with values plays the role of the monitoring system. Depending upon these values, penalty enforcement conditions can be invoked and the penalty can be calculated. All this information is transformed in RuleML and returned, which goes all the way back to the html based thin client and constitutes the answer of the query. The sequence of execution has been numbered in Figure 7.8. The step-wise detail of the validation process is elaborated as follows:

7.2.5.1. Step 1– Distributed Query and its Response

The EA representing the client interface is shown in Figure 7.9. The client can choose from the given SLO elements namely: Bandwidth, Availability, Resolution, Response Time. The client can further choose to direct his RuleML based query toward one or both of the service providers i.e., hosting engine and rendering workflow. The RuleML query for availability sent to the rendering workflow service is given below.

```
<RuleML xmlns="http://www.ruleml.org/0.91/xsd">

<Message mode="outbound"
 directive="query-sync">
  <oid>
     <Ind>VEO</Ind>
  </oid>
  <protocol>
     <Ind>esb</Ind>
  </protocol>
  <sender>
     <Ind>User</Ind>
  </sender>
  <content>
     <Atom>
        <Rel>getAvaibility</Rel>
        <Ind>veo_RenderingWorkflow</Ind>
```

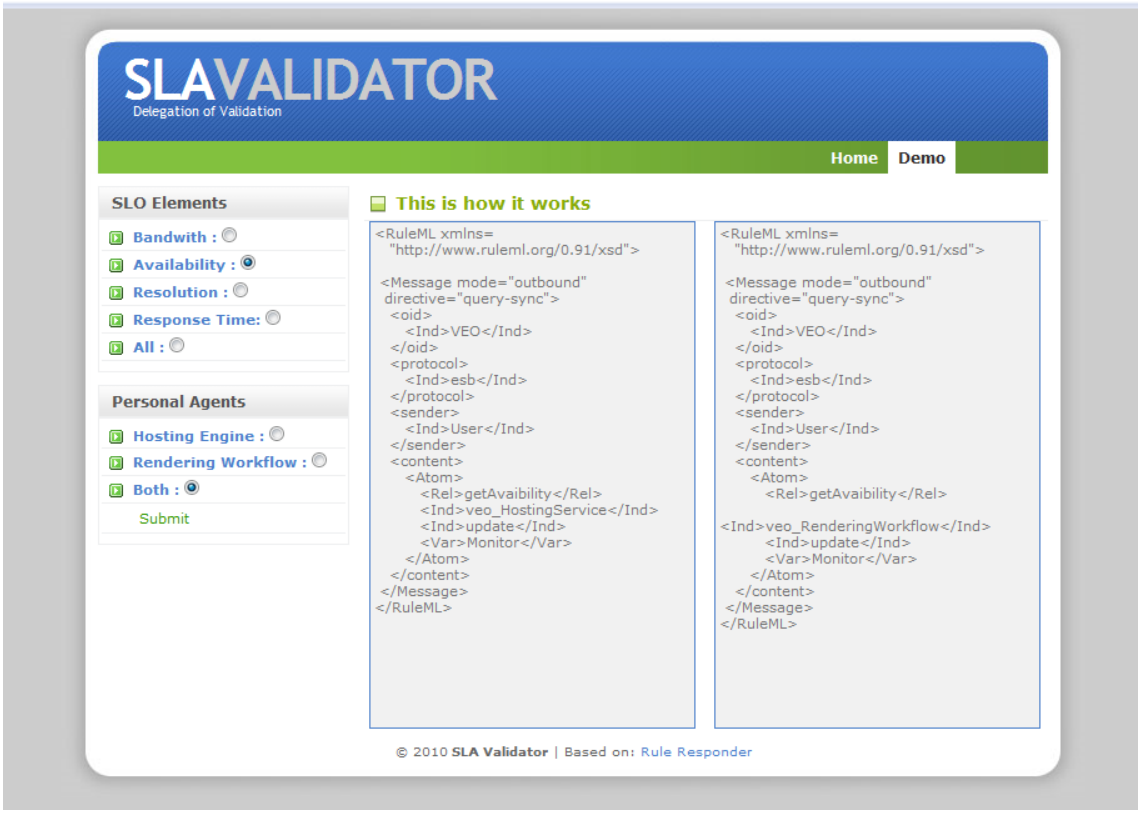


Figure 7.9.: External Agent with RuleML-Based Query)

```
        <Ind>update</Ind>
        <Var>Monitor</Var>
      </Atom>
   </content>
 </Message>
</RuleML>
```

The query travels the whole branch of the rendering workflow and brings back the availabilities of all the PAs involved the service value chain of the rendering workflow. It must be noted that steps 2 and 3 must happen before the answer is retrieved. The answer is shown below:

```
 <?xml version="1.0" encoding="UTF-8"?>
<RuleML xmlns="http://www.ruleml.org/0.91/xsd"
...

     <Atom>
```

```
      <Rel>getAvaibility</Rel>
        <Ind>veo_RenderingWorkflow</Ind>
        <Ind>update</Ind>
        <Expr>
          <Fun>SLO</Fun>
            <Expr>
              <Fun>chainAvaibility</Fun>
                <Expr>
                  <Fun>avaibility</Fun>
                    <Ind>98</Ind>
                    <Expr>
                      <Fun>computing_infrastructure</Fun>
                        <Ind>98.4%</Ind>
                        <Ind>penalty_is_15</Ind>
                        <Ind>Available at 2010-1-3,14:10:0?</Ind>
                    </Expr>
                    <Expr>
                     <Fun>media_engine</Fun>
                        <Ind>98.9%</Ind>
                        <Ind>penalty_is_10</Ind>
                        <Ind>Available at 2010-1-3,14:10:0?</Ind>
                    </Expr>
                </Expr>
            </Expr>
        </Expr>
   ...
</RuleML>
```

The construct </Expr> signifies the hierarchical nature of the query processing in the above chunk of RuleML based response.

7.2.5.2. Step 2(a) & 2(b)– Distributed Query Processing and Redirection in OA

In these steps the OA has to redirect the query toward the PA addressed in the query. The OA and the PAs are defined as endpoints on the Mule Enterprize Service Bus (ESB). The IP addresses and port numbers of these end points is given in the mule-all-config.xml file. A chunk of the file is shown below.

```
 ...
    <endpoint-identifiers>
    <endpoint-identifier name="SLAValidation" value="jms://topic:slaValidation" />
<!-- service endpoints of the SLAValidation use case -->
        <endpoint-identifier name="VEO" value="jms://topic:veo" />
```

```
        <endpoint-identifier name="veo_HostingService"
                             value="http://127.1.1.0:8080/HostingService/" />
        <endpoint-identifier name="veo_RenderingWorkflow"
                             value="http://127.1.1.0:8080/RenderingWorkflow/" />

   </endpoint-identifiers>
 ...
```

From the RuleML-based query received by the OA, the receiver of the query is determined through an OWL-Lite based ontology called Responsibility Assignment Matrix (RAM). RAM is used to describe the structure of various divisions of an organization and the responsibilities assigned to them. A chunk of RAM is shown below.

```
 <?xml version="1.0"?>

 <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
...
<owl:Ontology rdf:about="./Information.owl">
  <owl:versionInfo>v 0.01</owl:versionInfo>
  <rdfs:comment>Describes the RAM of VEO</rdfs:comment>
</owl:Ontology>

<owl:Class rdf:ID="VEO" />
<!-- Model of the VEO SLOs -->

<owl:Class rdf:ID="SLOs">
<rdfs:subClassOf rdf:resource="#VEO" />
</owl:Class>

    <owl:Class rdf:ID="Rendering_Workflow">
     <rdfs:subClassOf rdf:resource="#SLOs" />
    </owl:Class>

    <owl:Class rdf:ID="Hosting_Service">
        <rdfs:subClassOf rdf:resource="#SLOs" />
</owl:Class>

<!-- Responsibility Domains -->

<owl:Class rdf:ID="Responsibility">
<rdfs:subClassOf rdf:resource="#VEO" />
```

```
</owl:Class>

<!-- TODO: extend responsibilities -->

<Responsibility rdf:ID="Validation" />
<Responsibility rdf:ID="PenaltyEnforcement" />
<Responsibility rdf:ID="Monitoring" />
    <Responsibility rdf:ID="Policies" />
    <Responsibility rdf:ID="Billing" />

<!-- Meta Topics -->

<Responsibility rdf:ID="HostingService" />
<Responsibility rdf:ID="RenderingWorkflow" />
    ...

    <Rendering_Workflow rdf:ID="RenderingWorkflow">
 <responsible rdf:resource="#Validation" />
                 <responsible rdf:ressource="#Monitoring"/>
                 <responsible rdf:resource="#PenaltyEnforcement" />
                 <accountable rdf:resource="#Billing" />
         <informed rdf:resource="#Policies" />
 <responsible rdf:resource="#RenderingWorkflow" />
</Rendering_Workflow>

</rdf:RDF>
```

After sorting out the appropriate PA, the OA uses the following Prova function to send the query:

```
sendMsg(XID,esb,Agent, "query", slo(Avaibility))
```

then the OA goes into the listening mode and waits for the answer:

```
rcvMult(XID,esb,Agent, "answer", Contact)
```

7.2.5.3. Step 3(a) & 3(b)– Distributed Query Processing in PAs

The PAs corresponding to Rendering Workflow and Hosting are implemented as Java Servlets having POSL based knowledge-bases. A PA after receiving the answer from OA makes use of its POSL based rules to come up with the response to the query. A typical rule invoked in this condition looks like this:

```
contactRenderingAvaibility(?Avaibility) :-
slo(chainAvaibility[avaibility[?Avaibility]]).
```

The variable ?Availability can find a value from a local fact. But in the present scenario, availability and other SLOs depend on the services below in the chain. The rendering workflow is a composite service that uses two other services in the chain namely Media Engine and the Computing Infrastructure. To find out about the availability statistics of these two services, http messages are sent to the servlets of both the services.

The PAs representing Media Engine and the Computing Infrastructure are implemented as Java Servlets. Upon receiving queries about a specific SLO they consult their Prova based knowledge-base to come up with the answers.

The overall availability of a service is determined by defining initiating and terminating events for unavailability, such as an outage or a restart of the service. These events can be defined as:

```
initiates ( outage (S),unavailable (S),T).
terminates ( restart (S),unavailable (S),T).
```

which just defines that an outage event initiates the unavailable state and a restart event terminates the unavailable state. This detour over unavailability is due to the assumption that the service is initially running and available. Therefore, the initiating event should be something that terminates this initial state. The following rule simulates these events:

```
happens ( outage (S), datetime (Y,M,D,H,M,S)).
happens ( restart (S), datetime (Y,M,D,H,M,S)).
```

These rules describe that the outage or restart events happens at a certain point in time, defined by datetime(Y,M,D,H,M,S), thus initiating or terminating the unavailable state respectively. With this set of rules, one can finally set up a query, if the service is available at a given point in time:

```
:- solve (not( holdsAt ( unavailable (S),datetime (2010 ,8 ,30 ,10 ,30 ,0) ).
```

This query tests if the service was unavailable on 30-August-2010 at 10:30:00, negates it and returns yes, if it was available or no if it was not available.

For the penalty-reward system, the service can be assigned a rank depending on the average monthly availability:

- good: for an average monthly availability of 98% or more.
- ok: for an average monthly availability of 95% or more but less than 98%.
- bad: for an average monthly availability of less than 95%.

```
getRank(A,Rank,Penalty) :- less(A,95), Rank = bad,
                           Penalty = penalty-15.
```

The SLA validation and penalty enforcement is local decision and the PAs take execute these functions locally. But for the sake of demo, these statistics are inserted in the RuleML based stream and are directed back through the same channel to the client.

7.2.6. Conclusion

7.2.6.1. SLA Oriented Service Selection

It was found that the branch and bound algorithm scales very badly. It may be worth optimizing further. More results from the distributed implementation could be interesting as well. The heuristics proved themselves worthy of further investigation, as even a very simple heuristic update gives very good results, it is incomparably faster than branch and bound. It may be useful to pump research into using heuristics for the initial solution. The updating heuristics can also be extended to other kinds of changes in user requirements and to changes in service offerings. Finally, more complex heuristics could be looked into.

7.2.6.2. SLA Choreography Validation

This prototype implementation serves as a basic platform for further exploration of much more rigorous performance focused simulations of highly distributed large-scale SLA Choreographies. There are a few things needed to be understood regarding the scalability and fault-tolerance of the system.

- SLA Choreography is an emergent and self-organizing system. The service selection and negotiation process is independently employed at various levels of the SLA choreography whose result emerges into the formation of SLA connections in a hierarchical manner and thus an SLA Choreography is created. This emergent behavior allows the system to be highly scalable. The time taken by validation query to validate a huge SLA Choreography can be very hard to predict due to the diverse performance of various local rule-bases. This may not be an issue for most of applications but would be highly undesirable for time-critical applications. However, a worst case time can be calculated in advance and can be conveyed to the stake-holders as part of SLA parameters.

- Organizational Agent (AO) is the biggest hotspot in the system, whose failure can bring the whole system down therefore there must be some standby mechanism to keep the system going on even in case of the failure of OA. One strategy is based on replica management of the information contained by OA so that a new OA can be re-instantiated, reconfigured and reconnected as soon as such a failure is detected. A Replica generation of the whole SLA Choreography is neither feasible from space viewpoint nor sensible from a business perspective. Therefore, it is recommended to replicate only VOs. A VO contains all the end points of PAs connecting it. The system remains self-stable even in case of SLA Choreography scattered across multiple VOs. If a PA X is making a connection to another PA Y located in a VO other than its own then this X will have to go through the parent VO of Y to establish a connection. In this way X appears to be an EA to the parent VO of Y and its end point is maintained as a EA there. Thus replicating all VOs replicates all the end points of the SLA Choreography. Dual Modular Redundancy (DMR) [132] replication models can be employed to keep the system self-stable. In case a

VO adheres to some error-prone functions, depending on the specific requirements of the VO, a Triple Modular Redundancy (TMR) [143] or a pair-and-spare replication strategy can employed.

- In case of a PA failure, the system can self-organize itself. If a PA located somewhere in the SLA Choreography just fails then the all the links above in the chain are affected. If the PA can not be recovered then a new service provider is required to be added in the system. The service consumer of the failing PA can select a new service provider by using the heuristic updating algorithm. However, as a worst scenario, a renegotiation of service will be required across all the affected links.

7.3. Summary

For implementation, two components of the SLA-centric framework have been simulated to present a proof of concept of how these parts can be applied to a service-based Utility Computing infrastructure. The service selection algorithms have been parallelized, implemented and tested using the Kepler workflow tool and CORBA. The updating heuristic based service selection shows tremendous efficiency as compared to the branch and bound algorithm. The resilience of the algorithm has been tested against failing services and the approach has been found to be quite stable. For rule-based hierarchical validation of SLA Choreographies, a Rule Responder based implementation is demonstrated, which enables a distributed query to traverse across a set of distributed rules representing SLA aggregations across heterogeneous boundaries and get validated locally. For the sake of demonstration, the results are aggregated and brought back to client interface reflecting a hierarchical nature of the service value chains. This prototype can serve as a platform for rigorous performance tests of globally scattered large SLA Choreographies.

Chapter 8.

Extensions and Applications of the SLA-centric Framework

Its easy to see, hard to foresee.

(Benajmin Franklin)

The proposed SLA-centric framework for service-based Utility Computing is a group of generic models, which can be applied wherever they are needed in the Utility Computing based infrastructures. The author of the thesis has already proposed some extensions [122, 128] of the framework. One of such extension has been proposed for the unification of the rule based validation framework, SLA monitoring model LoM2HiS [55] and the cloud infrastructure LAYSI [33]. A summary of these proposed extensions is provided in the following sections.

8.1. Extension of the Validation Framework

In [122], the author of this thesis brings together three systems i.e., LAYSI [33], LoM2HiS [55] and the rule-based validation system and weaves a holistic validation framework for agile Cloud infrastructures. LAYSI - A Layered Approach for Prevention of SLA-Violations in Self-manageable Cloud Infrastructures, is embedded into the FoSII project (Foundations of Self-governing ICT Infrastructures) [5], which aims at developing self-adaptable Cloud services. LAYSI, as shown in Figure 8.1 provides an agile component based architecture for layered Cloud infrastructure and facilitates SLA-based service discovery, deployment, orchestration, maintenance and fault tolerance. The layered Cloud architecture utilizes loosely coupled and replaceable components like negotiator, broker or automatic service deployer, which are hierarchically glued together through SLAs.

LoM2HiS (Low Level Metrics to High Level SLAs) system provides a means to map resource metrics to high level service parameters. The service provider in this way uses this system to maintain the contracted QoS. It is an integral component of the *Foundations of Self-governing ICT Infrastructures (FoSII)* project [5]. The rule-based validation framework has been described in detail in Chapter 6.

Although the validation mechanisms of the three systems were already available but their blended version yields a holistic approach which targets the SLA validation problems within their specific scopes.

The holistic validation framework addresses the issue of validation at three levels:

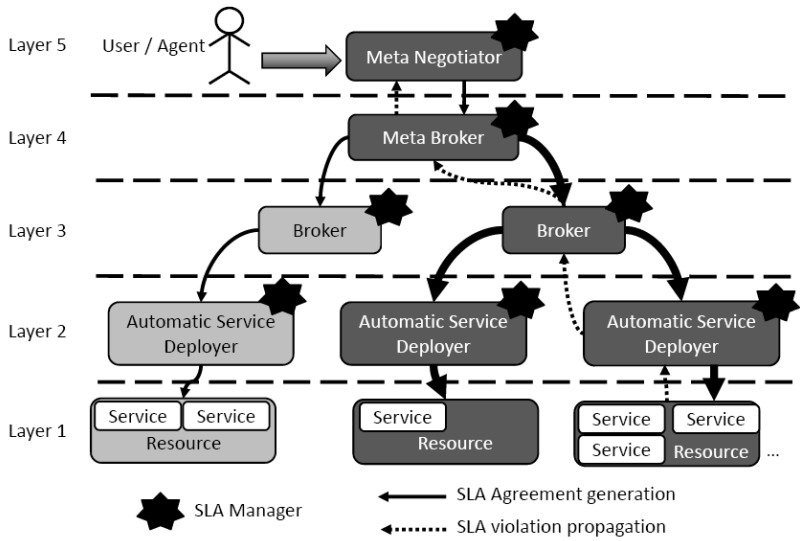


Figure 8.1.: Architecture of LAYSI

1. Infrastructure Level: Adjusting the agile Cloud infrastructure by fine tuning its building blocks to prevent the SLA violation threats. This type of validation is a specialization of LAYSI [33].

2. Resource Level: Taking proactive actions within the domain of service provider to prevent the possible SLA violations. For this, all the participating Cloud services need to implement certain interfaces provided by the LoM2His validation model. This is an intrinsic functionality of LoM2HiS model.

3. Business Level: Taking reactive measures when the SLA violation has already occurred and has been detected by the client. The violation needs to be localized and addressed within the service provider's SLA View due to the distributed constraints of the system.

The validation mechanism at the first two levels is proactive in nature i.e., preventive actions are taken before the violation has taken place. the third level requires a reactive validation approach i.e., a validation strategy after the SLA violation has occurred.

The proposed holistic validation system has been depicted in Figure 8.2

In the proposed holistic validation framework, different agents from the rule-based validation framework have been merged with various components of LAYSI framework in such a manner that the components of LAYSI framework has been divided into several domains, each taken care of by one agent of the rule-based validation framework. In the extended framework, it is assumed that each VO will be represented by an OA and will have its own broker. The Meta-broker will be part of the EA and will search services within various VOs by interacting with their respective brokers. Each services within a VO will be represented by its corresponding PA and will be required to implement the necessary

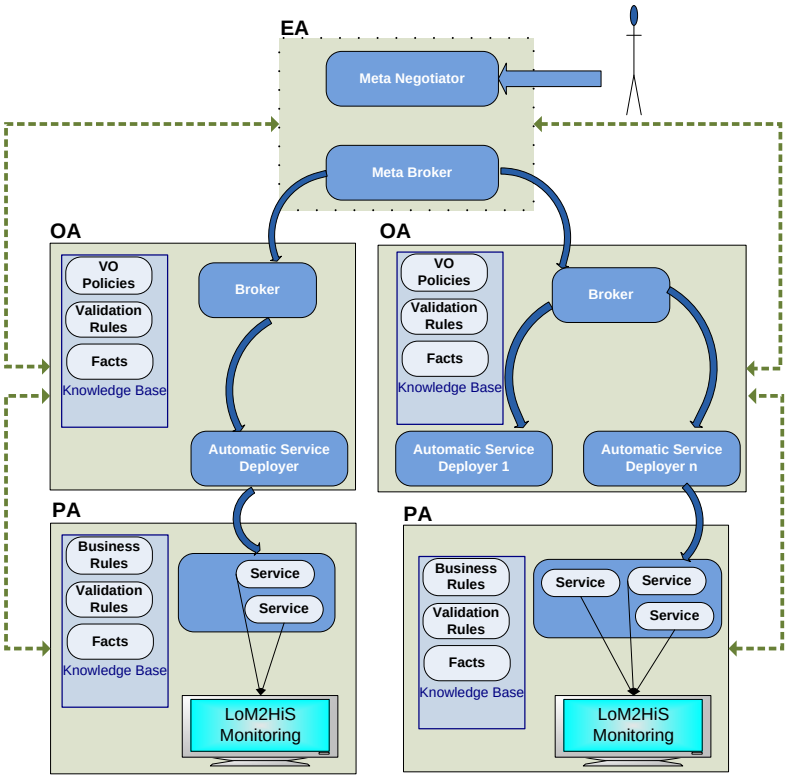


Figure 8.2.: A holistic SLA validation framework

LoM2HiS interfaces to attain the self-validation functionality for proactive SLA validation. The details of the proposed extended framework are available in [122]

8.2. Extension of the Negotiation Model

8.2.1. Formal Model

8.2.1.1. Parameter Vector

Both the set F of feasible configurations (and thus the renegotiation function g) and the price function f may depend on additional outside parameters known to the service provider, such as the amount of idle CPU power currently available on the server infrastructure, or such as the number of services from the same provider being purchased by the client, to be considered for mass purchase rebates. This can be modeled by introducing an additional *parameter vector* θ which is added to the definition of F, f and g, turning the set F into a set-valued function $F(\theta)$ and adding an additional parameter to $f(\theta, q_0)$ and $g(\theta, q_0, w)$. (In the definition of g, one only needs to replace F by $F(\theta)$, all the other quantities do not depend on θ.)

If the vector θ is assumed constant throughout the negotiation process, one can ignore

it during computation and just consider F, f and g for a given fixed value of θ.

8.2.1.2. Asymmetric Weights

Due to the symmetricity of the distance relation d_w used in the renegotiation function g, the client has no means to specify it for a given attribute, e.g. the resolution of the video, getting a higher quality than requested is not a big problem, but getting a lower one is. Instead, a violation by the same amount in either direction will always be the same.

This limitation can be addressed by introducing *asymmetric weights* $w^+ \in \mathbb{R}_+^m$ and $w^- \in \mathbb{R}_+^m$ and redefining d_w as the *asymmetric distance*

$$d_w(\hat{q}_0, q_0) = \sqrt{\sum_{i=1}^{m} \begin{cases} {w_i^+}^2(\hat{q}_i - q_i)^2, \text{ if } \hat{q}_i \geq q_i \\ {w_i^-}^2(\hat{q}_i - q_i)^2, \text{ if } \hat{q}_i < q_i. \end{cases}}$$

It shall be noted that this asymmetric distance is no longer a distance relation in the classical sense, which would require symmetricity, i.e. $d(u, v) = d(v, u)\ \forall u, v$.

8.2.2. Negotiation Protocol with Payment

The author of the thesis has proposed a payment model for mobile Grids [125, 77] based on gSET [135] payment mechanism. The gSET mechanism is based on Secure Electronic Transaction (SET) protocol [93, 94, 95] initially introduced for credit cards but could not gain popularity due to an intrinsic requirement of exchange of PKI related certificates. As shown in Figure 8.3, gSET requires an interaction among four parties:

1. Client
2. Service Provider
3. Trust Manager
4. Account Provider

There are two basic requirements of gSET:

- The client and the service provider should have a secure PKI based communication channel.
- The client and the service provider must trust a trust manager with whom they share there financial credentials.

The proposed framework for SLA-centric service-based Utility Computing has a strong affinity to gSET as it fulfills all the requirements of gSET protocol. The third party hybrid trust manager fosters PKI based security among the interacting partners, which can be utilized by the gSET based protocol. The third party trust manager acts as the root of the system and can very well mediate between service consumer and service consumer during

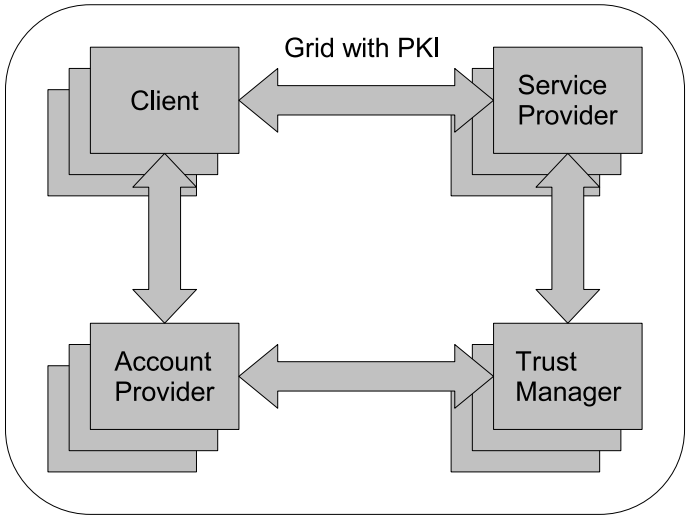


Figure 8.3.: gSET Architecture

gSET based transactions. The fourth element i.e., the Account Provider can be smoothly integrated with the rest of the system.

The proposed negotiation protocol can be extended by integrating it with a gSET based payment protocal. The gSET steps from [135] can be inserted to ensure payment right before the SLA is signed by both parties.

8.3. SLA Oriented Service Selection for Reverse Auction based Systems

The author has proposed [129] a blackboard [42] based system for service selection that employs branch and bound service selection algorithms and has contributed [115] in a blackboard system for service selection using A* algorithm. The proposed blackboard based systems can be extended for reverse auction. In the reverse auction based systems, service providers bid to sell their services against an advertisement posted by a client. The winning bid is not necessarily the lowest but usually the one coming from the service provider with the highest reputation. The blackboard system in this regard is a perfect place for publishing advertisements. The hybrid trust model can play a crucial role of reputation management. The proposed blackboard system as shown in Figure 8.4 consists of several regions where different parts of a workflow can be mapped. Service providers shown here as knowledge sources, will be invited to solve the advertised workflow activities by providing services to carry out these tasks. A control system will control the access of the blackboard.

The reverse auction system has huge potential for time efficiency and cost for the service consumer due to several reasons:

- It is an absolutely user-directed selection methodology as the consumer itself ad-

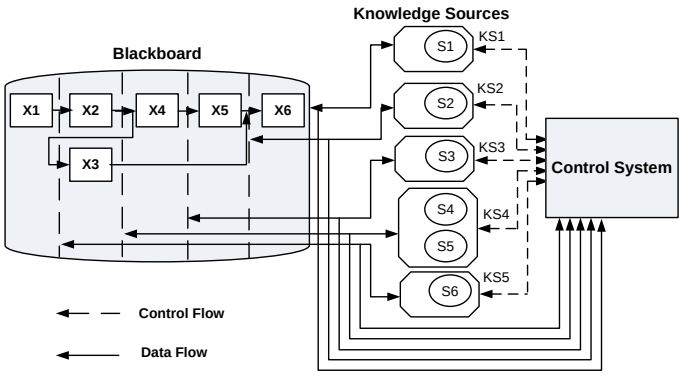


Figure 8.4.: A blackboard architecture for a reverse auction based service selection system

vertises the specifications of the required products and has the complete control throughout the process.

- As it is a user-directed approach, the service providers follow the user requirements from the very beginning thus saving the consumer useful time.
- The process is advantageous to both the service provider and the service consumer as it results in reducing marketing costs thus reducing product cost.

A blackboard based reverse auction system for service selection smoothly integrates with the proposed framework with service selection algorithms, reputation based trust and the negotiation protocol directly contributing to the system design.

8.4. SLA Aggregation Patterns in Business Processes

The SLA aggregation patterns presented in this thesis are available for application in Business Process Management, which expects to generate interesting developments especially in the sub-domains of architectural and connection patterns. Different types of business value networks are also an interesting area where the application of SLA aggregation patterns is sought. It must be highlighted that one of the final objectives of SLA@SOI [22] project is to develop SLA aggregation patterns. The contribution through this thesis is expected to directly complement SLA@SOI's research in this regard.

8.5. Applications in Enterprize 2.0

Enterprize 2.0 [67] is a rapidly evolving concept, which proposes to design new business models by integrating the technologies of web2.0 such as social networking, blogging, wikis etc. directly into the business enterprize. This is a very promising vision also for the development of IT based micro-economy in terms of new business models providing micro-players an opportunity to actively contribute into web based business initiatives. The

SLA-centric framework proposed in this thesis can play a very important role in realizing enterprize 2.0 based business models. One scenario can be a social community of mashup services having composite mashup services of varying granularity scattered across the virtual space of various members of the community. The social community, also seen as a Virtual Enterprize Organization, also defines a reputation based trust mechanism among its members. A service consumer with an objective to build a service based application initiates a search in this social community for highly reputed services having a specific combination of service attributes, rentable within a maximum affordable cost etc. As a result the potential service consumer is suggested a set of services owned by different members of this VEO. The service consumer intends to buy the service instances and initiates the negotiation mechanism. The Personal Agents (PA) of the service provider participate in the negotiation mechanism on behalf of their respective stakeholders. After a successful round of negotiation, services are connected together emerging into a business value network underpinned by SLA Choreography. From this scenario it is apparent how various components of the proposed SLA-centric framework can play a crucial role in order to realize the vision of Enterprize 2.0.

8.6. Summary

This chapter has presented various possibilities for the extension and the applications of the proposed SLA-centric framework for service based Utility Computing. The validation framework has been integrated with a Cloud based infrastructure for service provision. Several possibilities have been shown to extend and apply the proposed negotiation model. The trust model has been found equally useful in various roles such as a mediating body during payments, a unification authority in case of a VEO connecting various mashup service providers and a reputation manager in case of reverse auction blackboard systems. Several examples and scenarios have been presented to emphasize the high applicability of the proposed framework in various domains of service based Utility Computing.

Chapter 9.

Conclusion

If you want to know the end, look at the beginning.

(An African proverb)

This body of work elaborates the significance of Service Level Agreements as an enabling technology for the realization of service-based Utility Computing. The proposed framework covers the service life cycle and it discusses SLA oriented selection, negotiation, aggregation and validation and their particular roles in the formation of service value chains. The results of this research are expected to directly contribute in the materialization of the notion of service markets and their underlying service value chains. SLA oriented service selection algorithms provide a generic means to optimize QoS-based selection of service.

The negotiation algorithm for configurable services allows service providers and clients to dynamically tweak the fine details of their interests through a flexible negotiation algorithm. SLA Aggregation methodology is especially useful to micro-economy players such as composite service providers and resellers. SLA Choreography and its aggregation formulate a SLA-based mechanism to realize cross-enterprise business relationships. Several aggregation patterns have been extracted which can be applied in Business Process Modeling and cooperative workflows.

The hierarchical validation mechanism introduces a technique based on distributed query processing to validate SLA Choreographies scattered across multiple heterogeneous domains. Rule-based Systems present the possibility of implementing such a distributed validation mechanism, keeping the privacy issues of stakeholders intact at the same time. This research is likely to contribute to the research community by complementing projects such as SLA@SOI, SLA4DGrid etc. as well as to the industry by providing a basic framework for enabling SLA-based Utility Computing infrastructures.

Personal Reflections

- The end-user of a Cloud concerned with neither software nor hardware but only dataware.
- The aggregation of two Clouds is still a Cloud.
- Cross-enterprize business automation inevitably needs of third party trust management.
- The notion of SLA Choreography is indispensable for service value chain automation.
- Mashup technologies along with social networks offer an important role in the progress of IT-based micro-economy.
- A SLA View can perfectly be realized by an agent equipped with its local knowledge-base.
- Autonomic Computing, Human-Centered Computing and SOA are the driving philosophies of evolving ICT infrastructures.
- A Thousand unlit lamps are not worthy to illuminate even one whereas one enlightened lamp can illuminate thousands.
- Reality is a function of man's comprehension.
- One of the biggest blunders of science is the assumption of isolated systems.

Appendix A.

Installation Guide for the Simulation Environment for Optimized Service Selection

A.1. Compiling the Single-Machine Optimization Cores

To compile the non-distributed optimization cores, you need:

- GNU/Linux or a compatible operating system,
- a current version of g++ (the C++ compiler of the GCC suite, `http://gcc.gnu.org/`),
- a current version of Qt 4 (`http://qt.nokia.com/`), at least the `QtCore` module.

1. To build the sequential optimization core, symlink or copy `bnb-optimizer-sequential.cpp` to `bnb-optimizer.cpp` and run the `./build-optimizer.sh` script.
2. To build the threaded optimization core, symlink or copy `bnb-optimizer-threaded.cpp` to `bnb-optimizer.cpp` and run the `./build-optimizer.sh` script. (Note that only one of the sequential or threaded cores can be built at one time.)
3. To build the runtime optimization core (for runtime reaction to failed services), run the `./build-runtime-optimizer.sh` script.

A.2. Compiling and Installing the CORBA Optimization Cores

To compile the distributed optimization core using CORBA, you need all dependencies of the sequential version plus:

- TAO (The ACE ORB, `http://www.cs.wustl.edu/~schmidt/TAO.html`).

1. Change the hardcoded `#define NUM_SLAVES 4` near the beginning of `bnb-optimizer-corba.cpp`.
2. Change the hardcoded IP addresses and hostnames near the end of `bnb-optimizer-corba.cpp`.

3. On the master machine, configure the TAO event service, using the provided `tao-cosevent.conf` and `tao-cosevent.opt` files.

4. On the master machine, make sure the TAO name and event services are running. The script `corba-restart.sh` can be used to start or restart those services.

5. To build the master process, run the `./build-optimizer-corba-master.sh` script.

6. To build the slave process, run the `./build-optimizer-corba-slave.sh` script. The `scp-bin.sh` script (in which you will have to change the hardcoded hostnames) may be used to easily upload the built slave program to all slave machines.

A.3. Unsupported Variants of the Optimization Core

During development, various variants have been experimented with and discarded for various reasons. Those experimental variants are included in the `unsupported` directory. Since they did not end up working, we do not recommend attempting to build or use them and cannot provide any support for them. None of the published results were obtained with those unsupported versions of the code.

A.4. Executing the Optimizer through Kepler

The most straightforward way to run the optimizer is through the provided Kepler (`https://kepler-project.org/`) workflows (which expect the optimizer core in `~/Praktikum/bnb-optimizer`):

1. `kepler-workflow.xml` allows running any of the optimization cores (except the runtime optimizer) interactively through a basic dialog interface.

2. `kepler-workflow-auto.xml` allows running any of the optimization cores (except the runtime optimizer) automatically, with input coming from the `~/Praktikum/services.txt` and `~/Praktikum/workflows.txt` files.

3. `kepler-workflow-runtime.xml` does the same as `kepler-workflow-auto.xml`, but with the runtime optimizer (`~/Praktikum/runtime-optimizer`) enabled. It takes the additional input files `~/Praktikum/fixed.txt` (services which cannot be changed anymore at the given time) and `~/Praktikum/failed.txt` (services which failed at the given time and must be replaced).

All output is logged to `~/Praktikum/output.log`.

All hardcoded file names can be changed from the Kepler workflow editor.

A.5. Executing the Optimizer Directly

In some setups, it may not be practical to run Kepler on the machine on which the non-distributed core or the master process of the distributed core will run. For these cases, the `fake-optimizer` script is provided: if it is used as the `bnb-optimizer` executable, all the input passed by Kepler is logged to the `keplertest.txt` file, which can be easily transported to another machine and fed to the optimizer using a simple `./bnb-optimizer <keplertest.txt` input redirection.

A.6. Testcases

The `testcases` directory contains testcases to pass to:

- `bnb-optimizer` (directly) for `optimizer-input*.txt` and `testcase_*.txt`
- `kepler-workflow-auto.xml` for `services*.txt` and `workflows*.txt` (pairs with the same numbering must be used together, copied/symlinked/renamed to just `services.txt` and `workflows.txt`, respectively)
- `kepler-workflow-runtime.xml` for the single testcase composed of `services9.txt` (to rename to `services.txt`), `workflows9.txt` (to rename to `workflows.txt`), `fixed.txt` and `failed.txt`)

Appendix B.

Installation Guide for the Simulation Environment for Hierarchical SLA Validation

B.1. Setting Up Eclipse Environment

B.1.1. Prerequisites

- Installation of subclipse: http://subclipse.tigris.org/servlets/ProjectProcess?pageID=p4wYuA
- Installation of Mule IDE: http://www.mulesoft.org/documentation/display/MULEIDE/Home
- Download Mule: http://www.mulesoft.org/download-mule-esb-community-edition
- Add the JAVA_HOME environment variable to point to java install directory and Mule_Home to point to Mule installation directory.

B.1.2. Obtaining Source Code

- Windows->Show View->Other->SVN->SVN Repositories
- Right Click->New->Repository Location...
- Add the Rule Repsonder SVN repository:
  https://slavalidator.svn.sourceforge.net/svnroot/slavalidator
- Navigate to the root -> Right click on PragmaticAgentWeb -> Checkout

B.1.3. Configuring Mule

- Open the mule-all-config.xml file
- Change all end-point IP's (which point to PA's) to the IP of your machine.
- Save, and compile the JAR: Select the packages rules, src, repository, src/test/java, src/main/java, /src/main/resources, conf -> Right click -> Export -> Java -> JAR File
- Name the jar file pragmatic-agent-web-1.0-SNAPSHOT.jar and place in lib directory

B.1.4. Ports Required by SLAValidator

- Apache-HTTP-Server: 80
- Apache-Tomcat: 8080
- Mule: 8888, 60504
- RR OA's: 9995,9996,9997,9998,9999

B.2. Setting Up Apache HTTP Server

The apache HTTP server starts its file directory under htdocs. This is where you need to place any websites you wish to be installed. Remember that an HTML file labeled index.html located under root will be defaulted to when your IP is navigated to.

- Copy the EA webpage i.e. demo.html, to htdocs. If you wish to have it as default, rename as index.html
- The page must now be told to direct its message to your PC and not the previous server. Edit the page and find the old IP address (approximately halfway down), replace it with /local-ip/:8888 with the same port.

B.3. Setting Up TomCat Server

All servlets are located under the webapps folder. All source code for the PA's is located under /personalAgents. Class files for the PA's are located under "/target/classes". It should also be mentioned that there are two levels of PA-s. On the first level there are HostingService and RenderingWorkflow while on the second level there are ComputingInfrastructure and MediaEngine.

B.3.1. Configuring the first level PA Source

- Navigate to the PA you wish to Implement (e.g. /personalAgents/HostingService)
- Change the "address" global variable to local-ip
- Save, and now the class file is located in the directory mentioned above.

B.3.1.1. Creating the Tomcat Directory

- Create a new directory under ".../webapps" which has the same name as the PA which you are implementing (e.g. "HostingService").
- Create two more directories under the new one: "META-INF" and "WEB-INF"

- Navigate to META-INF and create a new file "MANIFEST.MF" with the following contents:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.6.5
Created-By: 1.5.0_06-b05 (Sun Microsystems Inc.)
Built-By: ...
Main-class:name of the main class
main class name = the name of the PA class (e.g. HostingService.class)
```

- Navigate to the WEB-INF folder and create the file "confix.xml" with the following contents:

```
<sign>
<url>local ip</url>
<context>class name</context>
<delay>5000</delay>
<log>yes</log>
</sign>
class-name = The main class name (e.g. HostingService)
```

- Navigate to the WEB-INF folder and create the file "web.xml" with the following contents:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<display-name>name of the class</display-name>
<description>Takes incoming messages and excutes queries</description>
<servlet>
<servlet-name>name of the class</servlet-name>
<servlet-class>name of the class</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>name of the class</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

- Navigate to the WEB-INF folder and create the folders "classes" and "lib"

- Under the lib folder, copy all contents from "/personalAgents/lib" to it.
- Under the classes folder, copy the PA class you want to use to it (e.g. "/target/classes/ HostingService.class")
- Under the classes folder, create a new folder called "resources-PA"
- Under the resources-PA folder, copy all contents from "/target/classes/resources-PA to it

B.3.2. Configuring the second level PA Source

It is pretty much the same procedure, the difference is just that the PA-s on the second level need some more files.

- Navigate to the WEB-INF folder and create the following folder structure ws->prova->esb.
- Now you need prova-2.0-SNAPSHOT.jar (or just prova.jar) which you can find in the lib folder of the project
- Open it (with Winrar or Winzip) and navigate to ws->prova.
- There you should look for the files : reagent.class, RMessageQueue.class and TaskQueue.class. Copy them in the prova folder which you have created inside of WEB-INF.
- Open again the jar file, navigate to ws->prova->esb and copy the file ProvaUmo.class to the esb folder in WEB-INF.
- Now take the .prova files (sla-availability, sla-bandwith, sla-resolution, sla-responsetime) and place them inside of "apache-tomcat-dir/ (the files with the values (bw.txt, rtime.txt should also be placed there)
- he last step is to open every prova file and change the paths in the header to : "your-project-dir/ rules/ContractLog/math.prova" Repeat with list.prove and list-math.prova.

B.4. Starting Up

- Start Apache-HTTP-Server: .../bin/httpd.exe
- Start Tomcat Server: .../bin/startUp.bat
- Start Mule Server: .../startUp.bat

Bibliography

[1] ASKALON Project, available at http://www.dps.uibk.ac.at/projects/agwl/, last accessed Nov 2010.

[2] AssessGrid Project, available at http://www.assessgrid.eu/, last accessed Nov 2010.

[3] BREIN Project, available at http://www.eu-brein.com/, last accessed Nov 2010.

[4] Community Scheduler Framework, available at http://www.globus.org/ grid_software/ computation/csf.php, last accessed Nov 2010.

[5] FOSII Project, available at http://www.infosys.tuwien.ac.at/linksites/FOSII/index.html, last accessed Nov 2010.

[6] Libra Project, CloudLab, University of Melbourn, available at: http://www.cloudbus.org/libra/, last accessed Nov 2010.

[7] MASCHINE Project, available at http://ai.eecs.umich.edu/people/wellman/maschine/, last accessed Nov, 2010.

[8] MASTER Project, available at http://www.master-fp7.eu/, last accessed Nov 2010.

[9] mOSAIC Project, available at http://www.mosaic-cloud.eu/, last accessed Nov 2010.

[10] NESSI-Grid SRA 3.0, available at http://www.soi-nwg.org/doku.php?id=sra:description, last accessed Nov 2010.

[11] NEXOF-RA Project, available at http://www.nexof-ra.eu/, last accessed Nov 2010.

[12] Open Grid Forum (OGF), available at http://www.ogf.org/, last accessed Nov 2010.

[13] POSL rule-based language, available at, http://ruleml.org/submission/ruleml-shortation.html, last accessed Nov 2010.

[14] RBSLA Project, available at http://ibis.in.tum.de/projects/rbsla/, last accessed Nov 2010.

[15] Romulus Project, available at http://www.ict-romulus.eu/web/romulus, last accessed Nov 2010.

[16] SLA4D-Grid Project, available at http://www.sla4d-grid.de/, last accessed Nov 2010.

[17] SOA4ALL Project, available at http://www.soa4all.eu/, last accessed Nov 2010.

[18] The Telemanagement Forum, available at: http://www.tmforum.org/pages/2016/default.aspx, last accessed: Nov 16, 2010.

[19] TrustCom Project, available at www.eu-trustcom.com, last accessed, Nov 2010.

[20] VIOLA Project.

[21] Virtual Organization Membership Service (VOMS). \url{http://hep-project-grid-scg.web.cern.ch/hep-project-grid-scg/voms.html}, 2003.

[22] SLA@SOI Project, available at http://www.sla-at-soi.org/index.html, last accessed Nov 2010., 2008.

[23] Adrian Paschke et al. Rule responder: RuleML-based agents for distributed collaboration on the pragmatic web. 2nd int. conference on Pragmatic Web, The Netherlands, 2007.

[24] M. Aiello, G. Frankova, and D. Malfatti. What's in an Agreement? A formal analysis and an extension of WS-Agreement. Lecture Notes in Computer Science, catagory Security and SLA, Springer Berlin Germany, 2005.

[25] R. AlNemr and C. Meinel. Getting More from Reputation Systems, a Context aware Reputation Framework Based on Trust Centers and Agent Lists. *The Third International Multi-Conference on Computing in the Global Information Technology (iccgi 2008)*, pages 137–142, July 2008.

[26] I. Altintas, A. Birnbaum, K. K. Baldridge, W. Sudholt, M. Miller, C. Amoreira, Y. Potier, and B. Ludaescher. A Framework for the Design and Reuse of Grid WorkFlows. In *International Workshop on Scientific Aspects of Grid Computing*, pages 120–133. Springer-Verlag, 2005.

[27] R. B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente and F. G. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. B.-Y., W. Emmerich. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Journal of Research and Development,*, 53(4), 2009.

[28] M. Ball, H. Boley, D. Hirtle, J. Mei, and B. Spencer. The OO jDrew Reference Implementation of RuleML. In *RuleML 2005*, Galway, 2005.

[29] D. Barry. *Web Services and Service-Oriented Architectures: The Savvy Manager's Guide*. Morgan Kaufmann, 2003.

[30] W. Binder, I. Constantinescu, B. Faltings, and N. Heterd. Optimal Workflow Execution in Grid Environments. In *NODe/GSEM*, 2005.

[31] M. B. Blake and D. J. Cunnings. Workflow Composition of Service Level Agreements. International Conference on Services Computing (SCC2007), 2007.

[32] H. Boley. The Rule-ML Family of Web Rule Languages. In *4th Int. Workshop on Principles and Practice of Semantic Web Reasoning*, Budva, Montenegro, 2006.

[33] I. Brandic, V. C. Emeakaroha, M. Maurer, S. Dustdar, S. Acs, A. Kertesz, and G. Kecskemeti. LAYSI: A Layered Approach for SLA-Violation Propagation in Self-Manageable Cloud Infrastructures. *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*, (i):365–370, July 2010.

[34] I. Brandic, D. Music, P. Leitner, and S. Dustdar. VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management. In *Proceedings of the 6th International Workshop on Grid Economics and Business Models*, GECON '09, pages 60–73, Berlin, Heidelberg, 2009. Springer-Verlag.

[35] J. Y. Buyya and Rajkumar. A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms. In *Workshop on Workflows in Support of Large-Scale Science, Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC 2006), Paris*, 2006.

[36] R. Buyyaa, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility . *Future Generation Computer Systems, Volume*, 25:599–616, 2010.

[37] I. Chebbi, S. Dustdar, and S. Tata. The view based approach to dynamic inter-organizational workflow cooperation. *Data and Knowledge Engineering*, 56:139–173, 2006.

[38] J. Chen and Y. Yang. Activity Completion Duration based Checkpoint Selection for Dynamic Verification of Temporal Constraints in Grid Workflow. *International Journal of High Performance Computing Applications,*, 319-329:22(3), 2008.

[39] J. Chen and Y. Yang. Temporal Dependency based Checkpoint Selection for Dynamic Verification of Temporal Constraints in Scientific Workflow Systems. In *accepted in ACM Transactions on Software Engineering and Methodology*, 2009.

[40] D. Chiu, S. Cheung, S. Till, K. Karalapalem, Q. Li, and E. Kafeza. Workflow view driven cross-organisational interoperability in a web service environment. *Information Technology and Management*, 5:221–250, 2004.

[41] D. Chiu, K. K. Q. Li, and E. Kafeza. Workflow view based e-contracts in a cross-organisational e-services environment. *Distributed and Parallel Databases*, 12:193–216, 2002.

[42] D. D. Corkill. Blackboard systems Craig, I D AI Review Vol 2 No 2 (1988) pp 103. *Knowledge-Based Systems*, 2(3):197, Sept. 1989.

[43] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP : A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. *LNCS Springer*, 2537/2002:153–183, 2002.

[44] D. D. Nurmi, R. Wolski, Ch. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and Zagorodnov. No TitleThe Eucalyptus Open-source Cloud-computing System. In *Proceedings of Cloud Computing and Its Applications 2008, Chicago, Illinois,*, 2008.

[45] A. Dan and et Al. Web services on demand: WSLA-driven automated management. *IBM Systems Journal Volume 43, Pages: 136 - 158, Issue 1*, Jan. 2004.

[46] G. Decker, O. Kopp, and A. Barros. An Introduction to Service Choreographies. *Information Technology*, 50(2):122–127, 2008.

[47] E. Deelman, J. Blythe, A. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Kor. Mapping Abstract Complex Workflows onto Grid Environments, 2003.

[48] M. S. Deun Ren Liu. Workflow Modeling for Virtual Processes: an order-preserving process-view approach. *Information Systems*, 28:505–532, 2002.

[49] D.G.A.Mobach B.J. Overeinder and F. M. T. Brazier. A ws-agreement based resource negotiation framework for mobile agents. Scalable Computing: Practice and Experience, 7 (1):23 ï¿½ 36, 2006.

[50] T. Dierks and E. Rescorla. The TLS Protocol Version 1.1. RFC 2246. Technical report, 2004.

[51] R. Duan, R. Prodan, and T. Fahringer. Run-time Optimisation of Grid Workflow Applications. *2006 7th IEEE/ACM International Conference on Grid Computing*, pages 33–40, Sept. 2006.

[52] M. S. Duen-Ren Liu. Business-to-business workflow interoperation based on process-views. *Decision Support Systems*, 38:399–419, 2004.

[53] J. Eder and A. Tahamatan. Temporal Consistency of View based Interorganizational workflows. 2nd International United Information Systems Conference, Austria, 2008.

[54] A. Edgardo, , B. Laura, , and F. A. Tiziana. Grid Workflow Optimization with Inferential Reasoning. *in Proceedings of CoreGRID Workshop, Poznan, Poland,*, 2005.

[55] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar. Low Level Metrics to High Level SLAs - LoM2HiS Framework : Bridging the Gap Between Monitored Metrics and SLA Parameters in Cloud Environments. *In Proc. International Conference on High Performance Computing and Simulation (HPCS), 2010*, pages 48–54, 2010.

[56] L. et al. Web Service Agreement (WS-Agreement). GFD.107 proposed recommendation.

[57] S. et al. Automated SLA monitoring for web services. Technical report, HP research report, HPL-2002-191, 2002.

[58] C. et. al. Cappiello. On Automated Generation of Web Service Level Agreements. *LNCS: Advanced Information Systems Engineering*, pages 264–278, 2007.

[59] FIPA. FIPA Agent Communication Language, http://www.fipa.org/, accessed Dec. 2001, 2000.

[60] I. Foster. What is the Grid? A Three Point Checklist. *Grid Today*, 2002.

[61] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications, 15(3)*, 2001.

[62] G. Frankova. {Service Level Agreements}: Web Services and Security. *Springer Verlag, Berlin Heidelberg*, pages 556–562, 2007.

[63] A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 Protocol. Technical report, Netscape Communications Corp., 1996.

[64] The Globus Security Team. *Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective*, Dec. 2004.

[65] I. U. Haq, R. Alnemr, A. Paschke, E. Schikuta, H. Boley, and C. Meinel. Distributed Trust Management for Validating SLA Choreographies. In *SLAs in Grids workshop, CoreGRID Springer series*, 2009.

[66] I. U. Haq, A. A. Huqqani, and E. Schikuta. Aggregating Hierarchical Service Level Agreements in Business Value Networks. In *Lecture Notes in Computer Science*, volume Volume 5701/2009, pages 176–192. Springer Berlin-Heidelberg, 2009.

[67] D. Hinchcliffe and H. Consulting. Enterprise 2 . 0 : How Business is transforming in the 21st Century. *Springer Enterprise 2.0, unternehmen zwichen hierarchie und selbstorganisation*, (2):21–45.

[68] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure: Certificate and CRL Profile. Technical report, RFC 3280.

[69] L. Huang, D. W. Walker, Y. Huang, and O. F. Rana. Dynamic Web Service Selection for Workflow Optimisation. *in Proceedings of 4th UK e-Science Programme All Hands Meeting (AHM), Nottingham, UK,*, 2005.

[70] M. Humphrey, M. R. Thompson, and K. R. Jackson. Security for Grids. *Proceeding of the IEEE*, VOL. 93, NO. 3.

[71] W. E. James Skene D. Davide Lamanna. Precise Service Level Agreements. 26th International Conference on Software Engineering (ICSE'04), 2004.

[72] W. Z. P. W. Jan Seidel Oliver Waldrich and R. Yahyapour. Using SLA for resource management and scheduling - a survey. Technical report, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, Aug. 2007.

[73] K. Jeffery and B. Neidecker. The Future of Cloud Computing, Opportunities for European Cloud Computing Beyond 2010, 2010.

[74] R. B. Jia Yu and C.-K. Tham. Cost-based Scheduling of Workflow Applications on Utility Grids. In *1st IEEE International Conference on e-Science and Grid Computing , Melbourne, Australia.*

[75] L.-j. Jin, V. Machiraju, and A. Sahai. Analysis on Service Level Agreement of Web Services, HP Technical Report: HPL-2002-180, available at :http://www.hpl.hp.com/techreports/2002/HPL-2002-180.html.

[76] S. Jones. TRUST-EC: Requirements for Trust and Confidence in E-Commerce. Technical report, European Commission Joint Research Centre, 1999.

[77] M. Juergen, W. Christoph, J. Oliver, S. Erich, W. Helmut, and H. I. Ul. Mobile gSET - secure business workflows for Mobile-Grid clients. *Concurrency and Computation: Practice and Experience*, 22(14), 2010.

[78] M. E. O. Karsten. A. Schulz. Facilitating cross-organisational workflows with a workflow view approache. *Data and Knowledge Engineering*, 51:109–147, 2004.

[79] L. H. Keller A. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, Vol. 11, No.1,.

[80] K. Kofler, I. Ul Haq, and E. Schikuta. A Parallel Branch and Bound Algorithm for Workflow QoS Optimization. *2009 International Conference on Parallel Processing*, pages 478–485, Sept. 2009.

[81] K. Kofler, I. Ul Haq, and E. Schikuta. User-Centric , Heuristic Optimization of Service Composition in Clouds. *Lecture Notes in Computer Science*, 6271/2010:405–417, 2010.

[82] A. Kozlenkov, A. Paschke, and M. Schroeder. Prova, http://prova.ws, accessed Jan. 2006, 2006.

[83] S. J. E. Lamanna D.D. SLAng: A Language for Defining Service Level Agreements. In Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems - FTDCS 2003 (Puerto Rico, May 2003). IEEE-CS Press, 2003.

[84] A. Leff, J. T. Rayfield, and D. M. Dias. Service-Level Agreements and Commercial Grids. *IEEE Internet Computing*, 7 (4):44–50.

[85] Q. Li, D. Chiu, Z. Shan, P. Hung, and S. C. Cheung. Flows and Views for scalable scientific process integration. In *First International Conference on Scalable Information Systems, Hong Kong*, 2006.

[86] A. Lioy, M.Marian, N.Moltchanova, and M. Pala. PKI past, present and future. *International Journal of Information Security, Springer Berlin*, page 2006.

[87] B. V. Looy, P. Gemmel, and V. Dierdonck. *Services Management: An Integrated Approach*. Financial Times, Prentice Hall, Harlow, England, 2003.

[88] A. Ludwig. COSMA -An Approach for Managing SLAs in Composite Services. In *Lecture Notes in Computer Science*. Springer Berlin-Heidelberg, 2008.

[89] H. Ludwig, T. Nakata, O. Waldrich, P. Wieder, and W. Ziegler. Reliable Orchestration of Resources using WS-Agreement. In *LNCS Springer, Munich, Vol. 4208:753 - 762*, Sept. 2006.

[90] Marilly, E. Martinot, O. Betge-Brezetz, and S. Delegue. Requirements for service level agreement management. In *IEEE Workshop on IP Operations and Management, ALCATEL CIT, Marcoussis, France*, 2002.

[91] D. Martin, M. Paolucci, S. Mcilraith, M. Burstein, D. Mcdermott, D. Mcguinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing Semantics to Web Services : The OWL-S Approach. *in Proceedings of First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, CA*.

[92] P. Masche, P. Mckee, and B. Mitchell. THE INCREASING ROLE OF SERVICE LEVEL AGREEMENTS IN B2B SYSTEMS. *Proceedings of the International Conference on Web Information Systems, Setuba*, pages 473–478, 2006.

[93] MasterCard, VISA. *SET Secure Electronic Transaction Specification, Book 1: Business Description*, May 1997.

[94] MasterCard, VISA. *SET Secure Electronic Transaction Specification, Book 2: Programmer's Guide*, May 1997.

[95] MasterCard, VISA. *SET Secure Electronic Transaction Specification, Book 3: Formal Protocol Definition*, May 1997.

[96] P. McKee, S. Taylor, M. Surridge, R. Lowe, and C. Ragusa. Strategies for the service market place. In *Proceedings of the 4th international conference on Grid economics and business models*, GECON'07, pages 58–70, Berlin, Heidelberg, 2007. Springer-Verlag.

[97] Mule. Mule Enterprise Service Bus, available at http://www.mulesoft.org/, last accessed Nov 2010., 2006.

[98] B. Neumann and Et.al. Kerberos: An authentication service for computer networks. *IEEE Commun. Mag*, vol. 32, n:pp. 33–38.

[99] T. Nurmela and K. Lea. Service level agreement management in federated Virtual Organizations. In *LNCS, Springer Berlin pp. 62-75*, 2007.

[100] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic {WS-Agreement} Partner Selection. Proceedings of the 15th international conference on World Wide Web, Edinburgh, Scotland, 2006.

[101] J. Padgett, K. Djemame, and P. Dew. Grid Service Level Agreements Combining Resource Reservation and Predictive Run-time Adaptation. In *In proceeding of School of Computing, University of Leeds, LS2 9JT, United Kingdom*, 2005.

[102] M. Parkin, P. Hasselmeyer, B. Koller, and P. Wieder. An SLA Re-Negotiation Protocol. In *2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC '08), Dublin, Ireland*, 2008.

[103] A. Paschke. RBSLA: Rule Based Service Level Agreement, http://ibis.in.tum.de/projects/rbsla/index.php, accessed Jan. 2006, 2004.

[104] A. Paschke. Reaction RuleML, http://ibis.in.tum.de/research/ReactionRuleML/events/ReactionRuleMLEvent06.htm, accessed, Nov. 2006. In *Special Event on Reaction RuleML at ISWC'06/RuleML'06*, Athens, Georgia, USA, 2006.

[105] A. Paschke. Verification, Validation and Integrity of Distributed and Interchanged Rule Based Policies and Contracts in the Semantic Web. In *Int. Semantic Web and Policy Workshop (SWPW' 06)*, Athens, Georgia, USA, 2006.

[106] A. Paschke. *Rule-Based Service Level Agreements - Knowledge Representation for Automated e-Contract, SLA and Policy Management.* Idea Verlag GmbH, Munich, 2007.

[107] A. Paschke and M. Bichler. Knowledge Representation Concepts for Automated {SLA} Management. *Int. Journal of Decision Support Systems (DSS).*

[108] A. Paschke and M. Bichler. {SLA} Representation Management and Enforcement. The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005.

[109] A. Paschke, B. Harold, A. Kozlenkov, and B. Craig. Rule Responder: A RuleML-Based Pragmatic Agent Web for Collaborative Teams and Virtual Organizations, http://ibis.in.tum.de/projects/paw/, 2007.

[110] A. Paschke, A. Kozlenkov, and B. Harold. Reaction RuleML Consensual Presentation, http://ibis.in.tum.de/research/ReactionRuleML/docs/RRCP.pdf, accessed Nov. 2006. White paper, 2006.

[111] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A Community Authorization Service for Group Collaboration. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.

[112] M. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance*. Free Press Publishers, 1985.

[113] J. D. Roo. Euler Proof Mechanism.

[114] V. A. Savva. Business Grid Computing Project Activities. *Fujitsu Scientific and Technical Journal*, 260(December):252–260, 2004.

[115] E. Schikuta, H. Wanek, and I. Ul Haq. Grid workflow optimization regarding dynamically changing resources and conditions. *Concurr. Comput. : Pract. Exper.*, 20(15):1837–1849, Oct. 2008.

[116] M. Shen and D. R. Liu. Discovering role-relevant process-views for disseminating process knowledge. *Expert Systems with Applications*, 26:301–310, 2004.

[117] G. Singh, C. Kesselman, and E. Deelman. Optimizing Grid-Based Workflow Execution. *Journal of Grid Computing*, 3(3-4):201–219, Jan. 2006.

[118] L. Skital, M. Janusz, R. Slota, and J. Kitowski. Service Level Agreement Metrics for Real-Time State of the Art. *LNCS*, 4967/2008:798–806, 2008.

[119] W. Sun, Y. Xu, and F. Liu. The role of XML in service level agreements management. International Conference on Services Systems and Service Management, 2005.

[120] D. I. Taylor, M. Shields, and D. I. Wang. Chapter 1 RESOURCE MANAGEMENT OF TRIANA P2P SERVICES.

[121] T. Tlhong and J. S. Reeve. Modeling and Management of Service Level Agreements for Digital Video Broadcasting(DVB) Services. *Lecture Notes in Computer Science Springer*, 4725/2007:288–294, 2007.

[122] I. Ul Haq, I. Brandic, and E. Schikuta. SLA Validation in Layered Cloud Infrastructures. *LNCS 6296*, pages 153–164, 2010.

[123] I. Ul haq, A. A. Huqqani, and E. Schikuta. A conceptual Model for Aggregation and Validation of SLAs in Business Value Networks. In *The 3rd International Conference on Adaptive Business Information Systems (ABIS 2009)*, 2009.

[124] I. Ul Haq, K. Kofler, and E. Schikuta. Dynamic Service Configurations for SLA Negotiation. In *In Proc. CoreGrid 2010, Europar 2010*, Ischia, Italy, 2010. Springer Verlag.

[125] I. Ul Haq, J. Mangler, H. Wanek, O. Jorns, and E. Schikuta. A Gridified, Secure, Mobile Business Workflow Using gSET. In *Workshop on Economic Models and Algorithms for Grid Systems in conjunction with the 8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, Austin, Texas, 2007.

[126] I. Ul Haq, A. Paschke, H. Boley, and E. Schikuta. Rule-Based Workflow Validation of Hierarchical Service Level Agreements. In *4th International Workshop on Workflow Management (ICWM2009) in conjunction with the The 4th International Conference on Grid and Pervasive Computing (GPC 2009) - Geneva, Switzerland*, 2009.

[127] I. Ul Haq and E. Schikuta. Aggregation Patterns of Service Level Agreements. In *Proc. ACM International Conference on Frontiers of Information Technology (FIT2010)*, Islamabad, 2010.

[128] I. Ul Haq, E. Schikuta, I. Brandic, A. Paschke, and H. Boley. SLA Validation of Service Value Chains. In *The 9th International Conference on Grid and Cloud Computing (GCC 2010)*, Nanjing, China, 2010.

[129] I. Ul Haq, E. Schikuta, and K. Kofler. Using Blackboard System to Automate and Optimize Workflow Orchestrations. In *The 5th IEEE Conference on Emerging Technologies (ICET 2009)*, Islamabad, Pakistan, 2009.

[130] I. Ul Haq, E. Schikuta, A. Paschke, and H. Boley. Rule-Based Validation of SLA Choreographies. *to appear in Journal of Super Computing*, 55(99), 2011.

[131] T. Unger, F. Leyman, S. Mauchart, and T. Scheibler. Aggregation of Service Level Agreement in the context of business processes. Enterprise Distributed Object Computing Conference (EDOC '08) Munich, Germany, 2008.

[132] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier. Multicore Soft Error Rate Stabilization Using Adaptive Dual Modular Redundancy. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, 2010.

[133] H. Wanek and E. Schikuta. Using Blackboards to Optimize Grid Workflows with Respect to Quality Constraints. *Grid and Cooperative Computing Workshops, International Conference on*, 0:290–295, 2006.

[134] M. Wang, R. Kotagiri, and J. Chen. Trust-based Robust Scheduling and Runtime Adaptation of Scientific Workflow. *Concurrency and Computation: Practice and Experience*, 21(16):1982–1998, 2009.

[135] T. Weishaeupl, C. Witzany, and E. Schikuta. gSET: Trust Management and Secure Accounting for Business in the Grid. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*, Singapore, May 2006.

[136] J. . I. B. M. C. WSLA Language Specification Version 1.0. No Title, 2003.

[137] J. Yan, R. Kowalczyk, J. Lin, M. B. Chhetri, S. K. Goh, and J. Zhang. Autonomous service level agreement negotiation for service composition provision. *Future Gener. Comput. Syst.*, 23(6):748–759, July 2007.

[138] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3(3-4):171–200, Jan. 2006.

[139] J. Yu, R. Buyya, and C. K. Tham. QoS-based Scheduling of Workflow Applications on Service Grids. *Technical Report, GRIDS-TR-2005-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia*, 2005.

[140] T. Yu and K.-j. Lin. K.: Service selection algorithms for composing complex services with multiple qos constraints. In *In: ICSOC05: 3rd Int. Conf. on Service Oriented Computing*, pages 130–143, 2005.

[141] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for Web services selection with end-to-end QoS constraints. *ACM Trans. Web*, 1(1), May 2007.

[142] R. Yu, J And Buyya. A Novel Architecture for Realizing Grid Workflow using Tuple Spaces. In *5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, 2004.

[143] Z. Zhang, D. Liu, Z. Wei, and C. Sun. Research on Triple Modular Redundancy Dynamic Fault-Tolerant System Model. *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)*, pages 572–576, June 2006.

[144] S. Zhao, A. Aggarwal, and R. D. Kent. PKI-Based Authentication Mechanisms in Grid Systems. International Conference on Networking, Architecture, and Storage, 2007.

[145] W. Ziegler, S. Birlinghoven, S. Augustin, D. Battr, and W. Ziegler. Extending WS-Agreement for dynamic negotiation of Service Level Agreements CoreGRID Technical Report Number TR-0172 Extending WS-Agreement for dynamic negotiation of Service Level Agreements, 2008.