

HLRS

Institut für
Hochleistungsrechnen

FORSCHUNGS - UND ENTWICKLUNGSBERICHTE

*ENHANCED SLA MANAGEMENT
IN THE HIGH PERFORMANCE
COMPUTING DOMAIN*

Bastian Koller

Höchstleistungsrechenzentrum
Universität Stuttgart
Prof. Dr.-Ing. Dr. h.c. M. Resch
Nobelstrasse 19 - 70569 Stuttgart
Institut für Höchstleistungsrechnen

ENHANCED SLA MANAGEMENT IN THE HIGH PERFORMANCE COMPUTING DOMAIN

von der Fakultät Energie-, Verfahrens- und
Biotechnik der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs
(Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Bastian Koller

aus Ansbach

Hauptberichter:	Prof. Dr.- Ing. Dr. h.c. Michael Resch
Mitberichter:	Prof. Dr.-Ing. Prof. E.h. Dr.-Ing. E.h. Dr. h.c. mult. Engelbert Westkämper
Tag der Einreichung:	19. April 2010
Tag der mündlichen Prüfung:	21. Januar 2011

D93

Danksagung

Nun, da ich die Druckversion meiner Dissertation vorbereite, ist es auch Zeit denjenigen Personen zu danken, die mir einen erfolgreichen Abschluss überhaupt erst ermöglicht haben.

Besonders möchte ich mich bei meinem Doktorvater Professor Michael Resch bedanken, der mich während vieler Gespräche immer wieder dazu angespornt hat diese Arbeit voran zu treiben und erfolgreich zu beenden. Herrn Professor Westkämper danke ich für seine Unterstützung als Mitberichterstatter.

Des Weiteren möchte ich Dr. Stefan Wesner danken, der mich seit meinem Arbeitsbeginn 2004 immer in meinem Anstreben der Promotion unterstützt hat und mit seinen konstruktiven Kommentaren zu meiner Arbeit eine sehr große Hilfe war. Philipp Wieder und Dr. Peer Hasselmeyer, meinen "Mitreitern" in Projekten wie NextGRID, dank ich für die vielen guten Diskussionen über Service Level Agreements und die damit verbundene Weiterentwicklung und Verfeinerung meines Themas.

Ebenso danke ich meinem Kollegen Lutz Schubert mit dem ich 2004 fast zeitgleich die Projektarbeit begonnen habe und der mich überhaupt erst mit der Thematik der Service Level Agreements bekannt gemacht hat.

Für das Korrekturlesen des englischen Textes bedanke ich mich bei Julia Wells, Daniel Field, Donal Fellows und Eva Windhäuser.

Freddie Klank danke ich für die Unterstützung beim Druck dieser Arbeit und die sehr gute Zusammenarbeit am HLRS.

Meiner Arbeitsgruppe "Service Management & Business Processes" (SANE) danke ich für die Unterstützung während der finalen Phase meiner Promotion und das gute Arbeitsklima, durch das wir auch in kritischen Situationen Bestarbeit leisten können. Mein besonderer Dank gilt meiner Familie. Meinen Eltern Inge und Otto Koller, die mich von Anfang an immer bestmöglich unterstützt haben und mir auch das notwendige Vertrauen geschenkt haben, damit ich immer meinen Weg gehen konnte. Meiner Frau Carolin danke ich für Ihre schier unendliche Geduld mit mir, dass sie mir jederzeit den Rücken freigehalten hat und nicht zuletzt dafür, dass sie mir unsere wunderbare Tochter Selina geschenkt hat.

Leinfelden-Echterdingen, im Januar 2011

Bastian Koller

Abstract

Historically, the Grid paradigm originates from the academic domain, assuming that there is the will of sharing resources in a collaborative manner and most likely for free [1]. The most prominent concept in handling this kind of collaborations are Virtual Organizations (VOs) [2] which represent cross-organizational shared resources to achieve a common goal.

However, with the growing influence of information technology on all areas of life, the way people and companies conduct business was (and still is) influenced and adaptation of new technologies and paradigms (such as Service Oriented Architecture - SOA) is of highest interest.

This evolution to VOs has strong implications on their realization as existing support mechanisms for rather academic oriented collaborations are not sufficient enough to be applied within the industrial domain. Business collaborations need stronger control mechanisms for the states of services and the environments in which they are executed. Additionally the necessity of stating and implementing contractual obligations and assurances according to business rules and laws by adequate tools is given.

The concept of Service Level Agreements (SLAs), originating as paper documents from the telecommunication domain, provides such a tool that allows the representing parts of electronic contracts. Especially terms of the agreed Quality of Service (QoS) or business-related contractual parameters (e.g. the involved business parties, penalties or success metrics) can be expressed by applying this technology. Several activities in research and development have already dealt with Service Level Agreements, however, up to today, there is no mature solution yet available.

The underlying assumption of this thesis is that with some enhancements and a merge of existing technologies, SLA can be made product ready for certain industrial domains. As example, the High Performance Computing (HPC) field is chosen, which faces due to the evolution of technology a diverse set of challenges based on the respective end user needs (e.g. in the field of Urgent Computing [3]). Currently all of the service lifecycle phases (design, transition and operation) are still handled manually. SLAs can be the technology which allows for (of course limited to the capabilities of the current resource management systems) semi-automation of processes. To develop a proper SLA Management for the HPC domain, first of all the SLA lifecycle is presented covering in total six different phases (development, creation,

provisioning, execution, assessment, termination). This lifecycle is comparable to the service lifecycle as defined by the IT Infrastructure Library [4]. In order to start from a realistic basis, a set of three use-cases from different domains is analysed and their requirements are collected. Furthermore additional requirements are added, based on examination of other reports or own experiences from research activities. The collection of requirements is classified into five different categories:

- Resourcing - Covering the requirements with respect to hardware and software set up
- Service Availability - Covering the requirements with respect to time constraints of the services availability
- Business Handling - Covering the business aspects in terms of monetary issues (such as penalties)
- Data Treatment - Covering the requirements with respect to data management and handling
- Contractual Terms - Covering the contractual aspects but also contractual issues which may influence the establishment of an SLA

Following this, the current state of the art in different areas is reviewed and evaluated with respect to the applicability of existing solutions to cover the requirements. A variety of solutions exists, but all of them address only parts of the above mentioned requirements. That makes further developments mandatory which are done in two different areas within this thesis:

- SLA Management - The development of an architectural blueprint for an SLA Management framework which takes into account existing technologies and their capabilities of Service Providers to ensure an easy uptake.
- SLA Schema - The respective representation of the desired contractual terms in an electronic format (in this case with XML).

The management architecture is built up based on the needed functionalities within the six phases of the SLA lifecycle. It enhances classical approaches with the needed business flexibility in terms of SLA creation by taking into account the context of the application and better decision support mechanisms during the runtime of the service.

Addressing the set of requirements, the to date most prominent (and most mature) SLA specification, WS-Agreement [5], is taken as basis for the schema development. In the first instance, the SLA structure as defined by WS-Agreement is extended to provide place holders for all necessary contents of an SLA. To enable a detailed

definition of terms from business to the plain infrastructure level, candidate specifications (e.g. the Job Submission Description Language, JSDL [6]) are embedded into this main structural frame. This allows later extraction of parts of the document which can then act as they are as input to the resource management system or other involved components.

Both solutions are checked for their feasibility, by applying and verifying them either with respect to the use cases or the SLA lifecycle. The schema itself is validated by instantiation of the the respective SLA documents for each case. This shows a clear benefit of the approach as e.g. domain specific issues such as the definition of creation constraints (i.e. the urgency to come to an agreement) are improved, which leads to a more effective SLA establishment process.

The realization of the architecture is discussed by going step by step through the lifecycle of a Service Level Agreement. For each phase it is elaborated which parts of the architecture could be covered re-using existing solutions and which would require additional effort to implement the functionalities. This leads to the result that even though not all components exist as fully functional implementations, a base framework could be already produced today, which may then be enhanced in future iterations with enhanced functionalities. This is also demonstrated by a deployment diagram showing the distribution of the architectural bits within the different firewall domains of an HPC Centre.

Nonetheless, not only SLA components will need evolution in future. As an outlook for future work, the limitations of current resource management systems are discussed, showing mandatory enhancements with respect to their capabilities (e.g. prediction mechanisms for system and job behaviour) to further improve the impact of SLA Management in an HPC environment.

Zusammenfassung

Historisch gesehen wurden der Begriff Grid und das zugehörige Paradigma innerhalb des akademischen Umfeldes definiert. Dabei nahm man an, dass bei den beteiligten Parteien jederzeit der Wille bestehe, eigene Ressourcen mit Anderen zu teilen oder minimalen Kosten zu teilen [1]. Das bekannteste Konzept in diesem Zusammenhang ist das der Virtuellen Organisationen [2], welches zur Darstellung und zum Management von organisationsübergreifend verteilten Ressourcen dient, die dazu benutzt werden, ein gemeinsames Ziel zu erreichen.

Nachdem die Informationstechnologie inzwischen in allen Lebensbereichen an Einfluß gewonnen hat, hat sich dementsprechend auch die Art und Weise wie Geschäfte auf elektronischem Wege geführt werden (oder geführt werden möchten) angepasst und neue Technologien und Paradigmen (z.B. Service-orientierten Architektur - SOA) sind für die Industrie von hohem Interesse.

Die Entwicklung zu geschäftlich orientierten VOs hat natürlich einigen Einfluß auf deren Umsetzung, da existierende Mechanismen für eher akademisch orientierte Zusammenarbeiten nicht alle notwendigen Eigenschaften besitzen, die für eine Anwendung im industriellen Umfeld von Nöten sind. Geschäftlich orientierte Zusammenarbeiten benötigen stärkere Kontrollmechanismen über die Dienste (ihre Zustände) selber aber auch über die jeweiligen Ausführungsumgebungen. Zusätzlich braucht man passende Mechanismen und Techniken, um vertragliche Verpflichtungen und Versicherungen (vor allem in rechtlicher Hinsicht) auf elektronischem Wege darzustellen.

Das Konzept der Dienstgütevereinbarungen oder Service Level Agreements (SLAs), das seinen Ursprung als Papierdokument im Bereich der Telekommunikationsindustrie hat, ist eine passende Technik mit deren Hilfe Teile elektronischer Verträge dargestellt werden können. Vor allem in Bezug auf Servicequalität und vertragliche Rahmenbedingungen (z.B. die Vertragsparteien, Vertragsstrafen oder Erfolgsmetriken) ist diese Technologie hilfreich. Bis jetzt gab es eine Vielzahl von Aktivitäten, die sich mit Dienstgütevereinbarungen auseinandergesetzt haben, jedoch gibt es bislang keine wirklich ausgereifte Lösung dafür.

Die dieser Arbeit zugrunde liegende Annahme ist, dass mit Hilfe von einigen Erweiterungen und durch das Zusammenführen existierender Technologien, Dienstgütevereinbarungen bereits jetzt schon in einigen industriellen Anwendungsfeldern nutzbar sein könnten. Als ein Beispiel wurde das der Höchstleistungsrechenzentren

(High Performance Computing Centres - HPCC) gewählt, welche sich aufgrund der sich verändernden technischen Möglichkeiten immer wieder neuen Herausforderungen (und damit Anforderungen) der Benutzer ausgesetzt sieht (z.B. aus dem Bereich des Urgent Computing [3]). Momentan werden alle Lebensphasen eines Dienstes noch manuell gehandhabt. Dienstgütevereinbarungen/SLAs könnten diejenige Technologie sein, die (im Rahmen der zur Verfügung stehenden Fähigkeiten der jeweiligen Resource Management Systeme) eine zumindest teilweise Automatisierung der Prozesse erlaubt.

Für die Entwicklung einer HPC spezifischen SLA-Management Lösung wird zuerst der Lebenszyklus von Dienstgütevereinbarungen/SLAs präsentiert, der sechs verschiedene Phasen beinhaltet (Entwicklung, Erstellung, Bereitstellung, Ausführung, Einschätzung, Beendigung). Dieser Lebenszyklus ist analog dem Lebenszyklus eines Dienstes wie von der IT Infrastructure Library [4] definiert. Um von einer realen Grundlage zu starten, werden drei verschiedene Anwendungsfälle diskutiert (jeweils aus verschiedenen Anwendungsgebieten kommend) und eine Anforderungsanalyse durchgeführt. Dazu werden weitere Anforderungen hinzugefügt, die aus Berichten anderer oder Erfahrungen eigener Forschungsaktivitäten bekannt sind. Diese Sammlung von Anforderungen wird in fünf verschiedene Kategorien eingegliedert:

- Resourcing - Anforderungen an die Hardware und Software Bereitstellung
- Verfügbarkeit des Dienstes - Beschränkungen, Anforderungen an die Verfügbarkeit des Dienstes
- Geschäftsabwicklung - Anforderungen in Bezug auf Geschäftsaspekte. Zum Beispiel die Darstellungen von Strafen (finanzielle Aspekte).
- Datenverarbeitungen - Anforderungen an das Datenmanagement
- Vertragliche Inhalte - Vertragliche Rahmenbedingungen, die auch die Vereinbarung der Dienstgüte beeinflussen können

Danach wird der aktuelle Stand der Technik betrachtet, vor allem in Bezug auf dessen Abdeckung der Anforderungen. Nachdem einige Lösungen existieren, diese aber jeweils nur einen Satz verschiedener Anforderungen adressieren, sind weitere Entwicklungen nötig. Diese werden auf zwei verschiedenen Schichten durchgeführt:

- SLA Management - Die Entwicklung einer architektonischen Blaupause für ein SLA Management System, das zum Ziel hat existierende Technologien von Diensteanbietern anzubinden, so dass das System relativ einfach integriert werden kann.
- SLA Schema - Eine passende Darstellung der Dienstgütevereinbarung (Schema) in elektronischem Format (in diesem Fall basierend auf XML).

Die Management Architektur ist anhand der sechs Phasen des SLA Lebenszyklus aufgebaut. Sie erweitert klassische Ansätze mit der benötigten (geschäftlichen) Flexibilität, vor allem in Bezug auf die Erstellung von Dienstgütevereinbarungen. Dies findet durch die Einbindung des Kontext von Anwendungen und Diensten in die Vereinbarungsfindung statt. Ferner werden auch verbesserte Unterstützungsmechanismen zur Entscheidungsfindung angeboten.

Um die Anforderungen zu adressieren, wird die momentan akzeptierteste SLA Spezifikation, WS-Agreement [5], als Grundlage für die Schemaentwicklung benutzt. Dadurch dass erweiterte Funktionalität erreicht werden soll, muss allerdings schon diese Grundlage angepasst werden. Um eine detaillierte Definition der einzelnen Terme sowohl auf der Geschäftsebene, als auch auf der Infrastrukturebene zu ermöglichen, werden ausgewählte andere Spezifikationen in diese Basis integriert (z.B. die Job Submission Description Language, JSDL [6]). Dies erlaubt gleichzeitig dann ein Extrahieren von Teilen des Dokumentes, die in Folge direkt als Eingabe für das Resource Management System oder andere Komponenten dienen.

Beide Lösungen werden auf Basis der Anwendungsszenarien und dem SLA Lebenszyklus auf ihre Realisierbarkeit getestet. Das Schema wird durch die Instantiierung der jeweiligen SLA Dokumente für jeden Anwendungsfall validiert. Dadurch werden Vorteile des Ansatzes herausgearbeitet, wie die domänenspezifische Definition der Grenzwerte innerhalb derer die Dienstgütevereinbarung erreicht werden muss (z.B. basierend auf der Dringlichkeit eine Vereinbarung zu finden). Dies ermöglicht ein effektiveres Erstellen einer Vereinbarung.

Die Realisierung der Architektur wird anhand der Lebenszyklusphasen verifiziert, indem für jede Phase gezeigt wird, welche Komponenten/Funktionalitäten von bereits existierenden Lösungen übernommen werden können und was mit einem gewissen Aufwand noch zu realisieren ist. Dabei wird deutlich, dass obwohl nicht alle Komponenten als voll-funktionsfähige Implementierungen existieren, eine Art Grundimplementierung schon jetzt zusammengeführt werden kann, um dann Stück für Stück funktional erweitert werden zu werden. Zusätzlich wird anhand eines Deployment Diagramms die Verteilung der Einzelkomponenten der Architekturen innerhalb der verschiedenen Firewalls eines Höchstleistungsrechenzentrums demonstriert.

Nichtsdestotrotz werden nicht nur die existierenden SLA Komponenten weiterentwickelt werden müssen. Um dies deutlich zu machen, werden die Einschränkungen momentaner Ressource Management Systeme aufgezeigt. Darauf basierend wird ein Ausblick gegeben, welche Weiterentwicklungen im Bereich der Ressource Management Systeme notwendig sind (z.B. Vorhersagemechanismen für das Verhalten von Diensten und deren Verhalten), damit SLA Management im Höchstleistungsrechnen etabliert und bestmöglichst verwendet werden kann.

Contents

List of Figures	xix
List of Tables	xxi
Abbreviations	xxiii
1. Introduction	1
1.1. The Aim of this work	3
1.2. Methodology	4
1.3. SLAs	5
1.3.1. The different phases of the SLA lifecycle	6
2. The Requirements For This Thesis	9
2.1. From Use Cases Towards Requirements	9
2.1.1. Medical support through simulation - The MS Use Case . . .	10
2.1.2. Improved car design through simulation - The CCS Use Case	12
2.1.3. An visualization Scenario - The VIS Use Case	15
2.2. Non scenario specific requirements	17
2.3. Categorization of Requirements	21
2.3.1. Resourcing	21
2.3.2. Service Availability	21
2.3.3. Business Handling	22
2.3.4. Data Treatment	22
2.3.5. Contractual Terms	23
3. Related Work	25
3.1. Middlewares	25
3.1.1. Globus Toolkit 4	25
3.1.2. Unicore 6	26
3.1.3. GRIA	26
3.1.4. gLite	27
3.1.5. Summary	28

3.2.	Service Level Agreement and Term Description Languages	28
3.2.1.	Web Service Level Agreements - WSLA	28
3.2.2.	WS-Agreement	31
3.2.3.	Semantic WS-Agreement Partner Selection (SWAPS)	32
3.2.4.	SLAng	33
3.2.5.	Web Service Offerings Language - WSOL	34
3.2.6.	Job Submission Description Language - JSDL	35
3.2.7.	Summary	35
3.3.	Resource Manager/Scheduler	37
3.3.1.	PBSPRO	37
3.3.2.	OpenPBS/Torque	37
3.3.3.	LoadLeveler	38
3.3.4.	Load Sharing Facility - LSF	38
3.3.5.	Sun Grid Engine - SGE	39
3.3.6.	Maui	39
3.3.7.	Moab	39
3.3.8.	Summary	40
3.4.	SLA Frameworks	41
3.4.1.	The NextGRID SLA Framework	41
3.4.2.	The Akogrimo SLA Framework	42
3.4.3.	The BREIN SLA Framework	42
3.4.4.	The BEinGRID SLA Framework	43
3.4.5.	SLA@SOI	43
3.4.6.	Summary	43
3.5.	Discovery/Negotiation Protocols	44
3.5.1.	Discovery	44
3.5.2.	Negotiation	47
3.5.3.	Auctions	50
3.5.4.	Summary	51
4.	The Architecture of the SLA Management System	53
4.1.	Terminology and concepts	53
4.1.1.	SLA Template	54
4.1.2.	(Invitation to) Tender	55
4.1.3.	SLA Request	55
4.1.4.	SLA Offer	55
4.1.5.	SLA Counter-Offer	55
4.1.6.	SLA Agreement	56
4.1.7.	Preventive SLA	56

4.2.	Six phases in the life of an SLA	56
4.2.1.	Development of the Service, the associated Templates and Publication	56
4.2.2.	Creation - Discovery and Negotiation of an SLA	58
4.2.3.	Service Provisioning and Deployment	60
4.2.4.	Execution of the Service	61
4.2.5.	Assessment and corrective actions during execution	62
4.2.6.	Termination and Decommission of the Service	64
4.3.	The high level architecture	66
4.3.1.	Development	66
4.3.2.	Creation	70
4.3.3.	Provisioning	78
4.3.4.	Execution and Assessment	82
4.3.5.	Termination	86
4.3.6.	Message flow between the different lifecycle phases	89
4.4.	Enhanced Creation of Service Level Agreements	90
5.	An SLA Schema for the HPC domain	93
5.1.	Scope for the Schema definition	93
5.1.1.	Usage of the Schema	94
5.1.2.	Contents of the Schema	94
5.2.	WS-Agreement as starting point	95
5.2.1.	WS-Agreement: The Agreement Structure	95
5.2.2.	WS-Agreement: The Template Structure	96
5.3.	The Schema	96
5.4.	Name	98
5.5.	Context	99
5.5.1.	hpcwsag:Customer	100
5.5.2.	hpcwsag:ServiceProvider	100
5.5.3.	hpcwsag:ThirdParty	101
5.5.4.	hpcwsag:DocumentBase	101
5.5.5.	hpcwsag:Validity	103
5.5.6.	Additional elements in the PreSLA Context	105
5.6.	Terms	107
5.6.1.	hpcwsag:ServiceTerms	108
5.6.2.	hpcwsag:GuaranteeTerms - The PreSLA-Document	109
5.6.3.	hpcwsag:GuaranteeTerms - The Agreement-Document	124
5.6.4.	hpcwsag:BusinessTerms	124
5.7.	Exclusion Terms	131
5.8.	Creation Constraints	131

6. Application And Verification Of The Concepts	135
6.1. Realizing the scenarios	135
6.1.1. Instantiating the SLA documents for the Medical Support Scenario	136
6.1.2. Instantiating the SLA documents for the Improved Car Design Scenario	145
6.1.3. Instantiating the SLA documents for the Visualization Scenario	153
6.2. Processing the Guarantee Terms	156
6.2.1. Resource Terms	156
6.2.2. Application Terms	156
6.2.3. Availability Terms	157
6.2.4. Data Treatment Terms	157
6.3. Realization of the SLA Lifecycle with today's State of the Art Technologies	157
6.3.1. Development	158
6.3.2. Creation	159
6.3.3. Provisioning	162
6.3.4. Execution and Assessment	164
6.3.5. Termination	166
6.4. Enabling enhanced offering of services	166
6.5. Deploying the SLA Management in a real world environment	168
6.6. Towards future resource management systems	168
6.7. Enhanced Business Models	171
7. Conclusions	173
7.1. Future Work	174
7.1.1. Enhanced Prediction of the Service Boundaries	174
7.1.2. Intelligent Context analysis and automated choice of protocols	175
7.1.3. Enhanced Terminology Mapping	176
A. The XSD	179
Bibliography	191

List of Figures

1.1. SLAs as bipartite relationships	3
1.2. The service lifecycle according to ITIL	6
1.3. The different phases of the SLA lifecycle	7
2.1. Simulation of an aneurysm	10
2.2. Visualization in a CAVE	16
3.1. The WSDL - WSLA Relationship	29
3.2. The WSLA Structure	30
3.3. The WS-Agreement Conceptual Layer Service Model]	32
3.4. The WS-Agreement Guarantee Terms	33
3.5. The WS-Negotiation Structure	47
3.6. Discrete Offer and Multiphase Negotiation	49
4.1. The different States of the SLA related Documents	54
4.2. Actions of Customers and Service Providers in the Development Phase	57
4.3. Actions of Customers and Providers in the Creation Phase	59
4.4. Actions of Customers and Providers in the Provisioning and Deploy- ment Phase	61
4.5. Actions of Customers and Providers in the Execution Phase	62
4.6. Actions of Customers and Providers in the Assessment Phase	63
4.7. Actions of Customers and Providers within the Termination Phase	65
4.8. Dependencies of the SLA Lifecycle Phases	65
4.9. The SLA Lifecycle - Focus on Development	66
4.10. Architecture of the Development Phase	67
4.11. The SLA Lifecycle - Focus on Creation	71
4.12. The Discovery Architecture	72
4.13. The Negotiation Architecture	74
4.14. The SLA Lifecycle - Focus on Provisioning	79
4.15. Architecture of the Provisioning Phase	80
4.16. The SLA Lifecycle - Focus on Execution and Assessment	82
4.17. Architecture of the Execution and Assessment Phase	83
4.18. The SLA Lifecycle - Focus on Termination	86

4.19. Architecture of the Termination Phase	87
4.20. Business Entities sharing a Common Context	90
5.1. The High Level Structure of WS-Agreement (Templates)	96
5.2. The High Level Structure of the Agreement Schema	97
5.3. The High Level Structure of the PreSLA-Document Schema	98
5.4. Structure of the Context Part of an Agreement	99
5.5. Structure of the Party Description	100
5.6. Structure of the Third Party Description	102
5.7. Structure of the DocumentBase Element	102
5.8. Structure of the Validity Element	104
5.9. Structure of the Context Part of an PreSLA-Document	106
5.10. Structure of the AgreementInitiator Element	106
5.11. Structure of the SLA Terms	107
5.12. Structure of the Service Terms of the SLA	108
5.13. The Guarantee Terms Structure of the PreSLA-Document	110
5.14. The Non-Negotiable Guarantee Terms Structure	111
5.15. JSDL representing the Resource Terms	113
5.16. JSDL representing the Application Terms	116
5.17. Structure of the Availability Terms	117
5.18. Structure of the AvailabilityRate Element	118
5.19. Structure of the ResponseTime Element	119
5.20. Structure of the Data Treatment Terms	120
5.21. Structure of the DataBackUp Element	121
5.22. Structure of the DataStaging Element	123
5.23. Structure of the Agreement Guarantees	124
5.24. Structure of the Business Terms	125
5.25. Structure of the CompensationType	126
5.26. Structure of the Price Terms	128
5.27. Structure of the Fixed Price Terms	128
5.28. Structure of the Variable Price Terms	130
5.29. Structure of the CreationConstraints	132
5.30. Structure of the CreationContext Element	133
6.1. Deploying the SLA Framework in a Real World HPC Environment	169
7.1. Probable extension with a plug-in concept	175
7.2. Variation of the terminology according to the interaction level	176

List of Tables

1.1.	Mapping of the Service to the SLA Lifecycle Phases	8
1.2.	The different roles within the SLA lifecycle	8
2.1.	Requirements of the End User in the Medical Use Case	13
2.2.	Requirements of the End User in the Car Crash Simulation Use Case	15
2.3.	Requirements of the End User in the Visualization Use Case	17
2.4.	Non Scenario specific Requirement for Contact Details	18
2.5.	Non scenario specific requirement for Traceability of the Origin of an SLA	18
2.6.	Non Scenario specific Requirement for the SLA Validity	19
2.7.	Non Scenario specific Requirement for the Limitation of Service Invocations	19
2.8.	Non Scenario specific Requirement for the Handling of Data	20
2.9.	Non Scenario specific Requirement for Business Definitions	20
3.1.	Frameworks addressing SLA needs	36
3.2.	RMS and Schedulers addressing Advanced Reservations, Queue Variety and JSDL Compliance	41
3.3.	Frameworks addressing SLA Needs	44
4.1.	Components involved in the Development Phase	70
4.3.	Description of the SLA Discovery Components	73
4.4.	Description of the SLA Negotiation Components	78
4.5.	Components involved in the Provisioning Phase	82
4.6.	Components involved in the Execution and Assessment Phase	86
4.7.	Components involved in the Termination Phase	89
4.8.	Mapping of the Service to the SLA Lifecycle Phases	89
5.1.	Mapping the SLA Documents to the Lifecycle Phases	94
5.2.	Elements of the Party Description	101
5.3.	Elements of the Third Party Description	101
5.4.	Elements of the DocumentBase	103
5.5.	Elements of the Validity Structure	105
5.6.	Elements of the AgreementInitiator Structure	107

5.7. Elements of the ServiceDescription	109
5.8. Elements of the ServiceReference	109
5.9. Attributes of the GuaranteeTerms	110
5.10. JSDL Resource Types addressing the Resource Requirements of the HPC Schema	112
5.11. Elements of the ResourceTerms	115
5.12. Elements of the ApplicationTerms Structure	116
5.13. Elements of the AvailabilityTerms Structure	118
5.14. Elements of the AvailabilityRate Structure	119
5.15. Elements of the ResponseTime Structure	119
5.16. Addressing the Availability Requirements with the Schema Elements	120
5.17. Elements of the DataTreatmentTerms Structure	121
5.18. Elements of the DataBackUp Structure	122
5.19. Elements of the DataStaging Structure	122
5.20. Elements of the Business Terms Structure	126
5.21. Elements of the CompensationType Structure	127
5.22. Elements of the AssessmentInterval Structure	127
5.23. Elements of the Penalty Structure	127
5.24. Elements of the Price Structure	128
5.25. Elements of the FixedPrice Structure	129
5.26. Elements of the Complete Structure for a Fixed Price	129
5.27. Elements of the Per Unit Structure for a Fixed Price	129
5.28. Elements of the Variable Price Structure	130
5.29. Elements of the Complete Structure for a Variable Price	130
5.30. Elements of the Per Unit Structure for a Variable Price	131
5.31. Elements of the Creation Constraints Structure	132
5.32. Elements of the CreationContext Structure	133

Abbreviations

Akogrimo	Access to Knowledge through Grid in a mobile World, page 40
BEInGRID	Business Experiments in Grids, page 17
BREIN	Business objective driven reliable and intelligent Grids for real business, page 17
BSD	Berkely Software Distribution, page 25
BSL	Binding Service Level Agreement, page 48
CAD	Computer-Aided Design, page 13
CAE	Computer-Aided Engineering, page 13
CAVE	Cave Automatic Virtual Environment, page 15
CCS	Car Crash Simulation, page 12
CET	Central European Time, page 19
CFD	Computational Fluid Dynamics, page 13
CIM	Common Information Model, page 155
COVISE	Collaborative Visualization and Simulation Environment, page 15
CPU	Central Processor Unit, page 26
CVD	Cardiovascular Diseases, page 10
DB	Database, page 26
DEISA	Distributed European Infrastructure for Supercomputer Applications, page 25
DO	Discrete Offer, page 42
DRMAA	Distributed Resource Management Application API, page 36
DUNS	Data Universal Numbering System, page 44
EC	Elastic Cloud, page 93
EGEE	Enabling Grids for E-sciencE, page 2
EMOF	Essential Meta Objective Facility, page 32
FCFS	First-Come-First-Serve, page 36
FIPA	Foundation for Intelligent Physical Agents, page 49
GENIUS	Grid Enabled Neurosurgical Imaging Using Simulation, page 11

GRAAP	Grid Resource Allocation Agreement Protocol, page 35
GRIA	Grid Resources for Industrial Applications, page 25
GT4	Globus Toolkit 4, page 24
GTPL	Globus Toolkit Public License, page 25
GUI	Graphical User Interface, page 156
HPC	High Performance Computing, page 1
hpcwsag	HPC WS-Agreement, page 93
ITIL	IT Infrastructure Library, page 2
JSDL	Job Submission Description Language, page 25
LCG	Large Hadron Collider Computing Grid, page 26
LGPL	Lesser General Public License, page 26
LSF	Load Sharing Facility, page 3
MDS	Monitoring and Discovery System, page 44
MRA	Magnetic Resonance Angiography, page 10
MRI	Magnetic Resonance Imaging, page 10
MS	Medical Simulation, page 10
N1GE	N1 Grid Engine, page 38
NextGRID	The Next Generation Grid, page 17
OASIS	Organization for the Advancement of Structured Information Standards, page 43
OCL	Object Constraint Language, page 33
OGF	Open Grid Forum, page 30
OGSA	Open Grid Services Architecture, page 24
OS	Operating System, page 14
PBS	Portable Batch System, page 26
QoE	Quality of Experience, page 81
QoS	Quality of Service, page 2
RMS	Resource Management System, page 39
RSLA	Resource Service Level Agreement, page 48
SA-SLA	Semantic Annotated Service Level Agreements, page 41
SDT	Service Description Term, page 31
SGE	Sun Grid Engine, page 37
SJF	Shortest-Jobs-First, page 36
SLA	Service Level Agreement, page 2
SLO	Service Level Objective, page 29

SNAP	Service Negotiation and Acquisition Protocol, page 48
SOA	Service-Oriented Architecture, page i
SODR	Service Offering Dynamic Relationship, page 33
SOI	Service Oriented Infrastructure, page 42
SP	Service Provider, page 66
SWAPS	Semantic WS-Agreement Partner Selection, page 31
TAPAS	Trusted and QoS Aware Provision of Application Services, page 32
TMF	TeleManagement Forum, page 6
TORQUE	Terascale Open-Source Resources and Queue Manager, page 36
TrustCoM	Trust and Contract Management Framework, page 30
TSLA	Tasks Service Level Agreement, page 48
TTP	Trusted Third Party, page 52
UDDI	Universal Description, Discovery and Integration, page 43
UNSPSC	United Nations Standard Products and Services Code, page 44
VIS	Visualization, page 15
VO	Virtual Organization, page 1
WHO	World Health Organization, page 10
WSAG	Web Service Agreement, page 42
WSDL	Web Service Description Language, page 33
WSIL	Web Service Inspection Language, page 44
WSLA	Web Service Level Agreement, page 3
WSOL	Web Service Offering Language, page 33
WSRF	Web Services Resource Framework, page 24
XML	Extensible Markup Language, page ii

Chapter 1.

Introduction

With electronic business (eBusiness) becoming ubiquitous, the traditional ways of doing commerce need to be changed or completely replaced to support business entities (e.g. companies) effectively in performing their business. One target domain of these eBusiness activities is the High Performance Computing (HPC) domain, in which many different types of electronic services (e.g. simulation services, visualization services...) are offered and which may benefit from adequate eBusiness solutions. Therefore these solutions aim to support different kinds of service provisioning, from the simple case of offering single resources (e.g. Data Storage) up to complex aggregated services provided across organizations (e.g. conceiving and realizing of a new air plane) to satisfy the needs of a variety of end users. By that, the movement away from the classical offering of one service by Provider towards the potential of offering composed services which are then fulfilled by a set of Service Providers is on its way. In other words, an easy extensibility of the service portfolio needs to be possible, especially for small players in business.

A conglomerate of all entities working together to achieve a common (business) goal is called Virtual Organization [2] (VO) . Virtual Organizations can be described as a set of individuals, companies or organisations that agree to work together to provide a comprehensive service to another organization. Thereby VOs allow for managed and dynamic operation of different services to enable transactions between the different companies in a coordinated manner whilst taking into account the business objectives/goals of the individual VO members. By that they ensure that the needs of Customers and Service Providers are met as good as possible. Within the scope of this thesis, Virtual Organizations in the Grid are addressed which target the HPC domain.

As with the possibility of composing services to new bigger services the the number of different Providers working together may increase, the relationships of business entities within an VO can become numerous and dynamic proportional to the complexity of the services. Consequently also the risk of failure of a composed services

increases with the number of involved business entities and the degree of dependencies that exists amongst them.

In that scope, the topic of business relationship management and monitored service execution becomes a preconditions for satisfaction of all parties. In turn, that implies the need for enhanced technologies and mechanisms.

Originating from a more academic background (for example the EGEE¹), the foundations of Virtual Organizations have been relatively simplistic, with a focus on access control for execution. Applying the VO concept to the real business world implies the need for ensuring that legal obligations are met (e.g. in terms of licensing) and that correct payment is remitted.

With the change towards the business community, these foundations have to be extended to ensure that the expectations of the quality of the services (QoS) of the involved business parties are met. Therefore the means for establishing and maintaining the business relationships are becoming imperative and need to provide the necessary capabilities to ensure that assurances are given, in case one party does not fulfil its tasks.

This implies the need for an instrument which can be used for (a) representation of the tasks of each party within a business relationship, (b) representation of contractual terms and (c) allowance for monitoring and evaluation of performance. A valuable concept covering all the three needs are Service Level Agreements (SLAs) which have already been presented as an important concept for operating Virtual Organizations [7] [8].

The IT Infrastructure Library (ITIL) [9] describes Service Level Agreements as follows:

An Agreement between an IT Service Provider and a Customer. The SLA describes the IT Service, documents Service Level Targets, and specifies the responsibilities of the IT Service Provider and the Customer.

Furthermore they argue that

A single SLA may cover multiple IT Services or multiple Customers.

which can easily be done when having a paper SLA, but which is not foreseen in this work due to its high complexity with respect to mapping and legal validity. In this thesis it is assumed that a Service Level Agreements is always a bipartite relationship. Figure 1.1 shows as example three users from which always two are connected by respective Service Level Agreements. Even though, there is no relationship between A and C as C only has an SLA with B. By that Virtual Organizations become much more dynamic as in the case of a misbehaviour Service Provider X can be easier replaced with Provider Y to fulfil an SLA without having to take into account

¹EGEE - Enabling Grids for E-scienceE, Website: <http://www.eu-egee.org/>

potential dependencies to other participants in the VO. However, this replacement is only acceptable, if the guarantees of the original SLAs are met. In all other cases, re-negotiations or other corrective actions have to be taken.

Recent research activities (e.g. NextGRID [10] and TrustCoM [11] projects) started

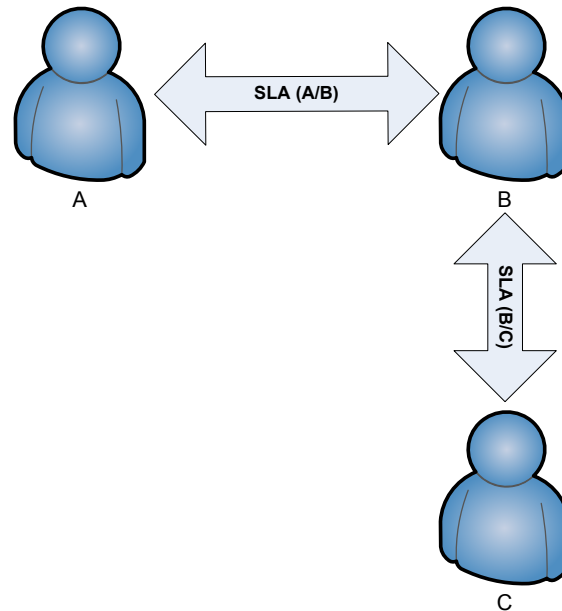


Figure 1.1.: SLAs as bipartite relationships

addressing the business market by providing Service Level Agreement management solutions. In parallel to that standardization activities by industry (such as the specification of the Web Service Level Agreement Language - WSLA [12]) have been performed to provide a common basis for the design of Service Level Agreements. However, all these attempts have not found the aimed uptake yet as the results are often immature or designed in a generic way which requires additional investment to ensure usability in concrete application areas.

Therefore, another step forward is needed to make Service Level Agreements usable for business. This thesis focuses on the taking of that step within the HPC domain.

1.1. The Aim of this work

The development of current solutions for resource management of HPC environments were not mainly driven by the business needs and goals of service providers. Only particular cases (such as LSF - Load Sharing Facility) deliver SLA Management

capabilities, but they are extremely limited to the infrastructure domain and, as mentioned before, deliver no means to manage business relationships. The objective of this thesis is to develop and present how Service Level Agreements can enhance the overall business of High Performance Computing Centres in terms of relationship, service and resource management.

The work presented will be based on experiences in and results of academic and industrial research activities, as well as national and international projects. Thereby it is not aimed to re-invent everything from scratch but rather to see how existing technologies can be used to build up an SLA Management framework together with an according SLA Schema for the HPC domain which could be implemented and used already today. Baseline for the developments will be three use cases selected to cover a wide range of requirements of HPC and by that SLA Management.

Within these developments a dedicated focus will also lie on increasing the flexibility of users of the framework with respect to the choice of protocols to create an agreement. When in existing approaches the users are obliged to use exactly and only the fix implemented solution, independently of whether it fits their needs or not, a solution for SLA creation (consisting of discovery of Service Providers and negotiation of Agreements with them) will only be taken up if it provides the means for choosing the best fitting protocol to derive an SLA. If not, end users of such a framework would be forced to install (and buy) other implementations which support alternative protocols. This would limit the overall benefit of SLA usage in a Providers domain.

It is important to mention here that this work will not address the intelligence behind such a system in terms of the decision making process, e.g., whether to accept a contract proposal or not. Furthermore, SLAs as such are seen within the context of an existing contract (a frame contract) but not representing all aspects of a legal valid contract.

1.2. Methodology

As stated in the previous sections, the starting points for this thesis are three use cases. These are analysed to get a list of requirements they have on an SLA Management Framework and Schema for the HPC domain. Supplementary to these extracted requirements, additional non-scenario specific requirements were worked out, derived from previous experiences with business oriented SLA Management.

These requirements together with the analysis of the current State of the Art in related technologies will show the gaps that have to be overcome to deliver a solution for embedded SLA Management in an HPC Environment. This leads to an architecture sketch which inherits concepts from existing work and enhances and combines them

to form the proposed framework for SLA Management in the HPC domain.

As second important step, an SLA Schema will be presented in this thesis which is based on existing SLA and term description languages. On basis of the three use cases, this schema is instantiated and validated. Consequently also the architecture of the framework is validated by analysing in how far the needed capabilities of resource management systems are realizable with the current state of the art and what benefits can be gained.

Finally the thesis concludes with a summary of the results and provides potential domains of future work, which will be needed to realize the full functionalities of the presented solution.

1.3. SLAs

From the early 1980s on, Service Level Agreements were realized as paper documents in the Telecommunication Industry to record a common understanding about services, priorities, responsibilities, guarantees and warranties of Customers and Providers. Once used mainly by Telecom Operators, the usage of SLAs (as paper documents) is now relatively common in most business areas. However, most of these SLAs are still represented as formal paper documents and by that have to be handled manually. Independent of the format they exist in, they specify certain parts of the contractual relationship between business entities, commonly established as a bipartite relationship between one Customer and one Service Provider.

In addition to the ITIL definition cited before, the TeleManagementForum² definition of a Service Level Agreement (as can be read in their SLA Management Handbook [13]) supports this statement. They see an SLA as

a formal negotiated agreement between two parties, sometimes called a service level guarantee. It is a contract (or part of one) that exists between the service provider and the customer, designed to create a common understanding about services, priorities, responsibilities, etc.

Ergo it can be stated that the concept of SLAs (Service Level Agreements) foresees the definition of electronic contracts that define specific quality of service parameters (QoS). But it does not state *how* the provider does it.

Given the right system, SLAs would enable automatic observation of a specific service's performance and thus allow for timely reactions to critical deviations. With the respective set-up, the information thus provided can also be used to support self-management [14] functionality. By using SLAs, Customers as well as Service

²TMF - Tele Management Forum Homepage, <http://www.tmforum.org>

Providers have a document which states certain quality properties, in most cases bound to penalties for the service provider failing to deliver this quality but probably also putting obligations on the Customer.

Within the frame of this thesis, SLAs are assumed to be part of a framework agreement, which may contain further information about legal or security aspects. However, SLAs are not complete contracts.

To understand the implications of using Service Level Agreements in a service oriented environment, and also to select the appropriate schema, it is of high importance to know the lifecycle of a Service Level Agreement. This includes the different actions where it is used as well as the different operations made on it.

Even though this thesis will mainly concentrate on the so-called discovery and negotiation phases (also labelled as creation phase), awareness of the whole lifecycle should be given as the creation of Service Level Agreements has implications on all other phases. Thus, the SLA lifecycle is shortly presented in the following part of this document.

1.3.1. The different phases of the SLA lifecycle

It is obvious that the lifecycle of a Service Level Agreement is closely related to the service lifecycle itself. Figure 1.2 is based on the most recent version of a service lifecycle as defined by the IT Infrastructure Library . The service (business) strategy of a Service Provider influences the design, transition and operation of the service. With the cyclic approach, the basis is set for continuous improvement of the service. Analogue to the lifecycle of a service, we can also distinguish several sub-phases



Figure 1.2.: The service lifecycle according to ITIL

of an Service Level Agreement. Many research activities [15] have dealt with this

lifecycle, mainly basing it the lifecycle definition from the TeleManagement Forum (TMF). Figure 1.3 shows an overview of the different phases in the SLA lifecycle, which is based on the TMF definition.

- **Development** of the Service and the according SLA Templates, Publication
- **Discovery** of Service Providers and **Negotiation** of an SLA (**Creation**)
- **Service Provisioning** and **Deployment**
- **Execution** of the Service
- **Assessment** and **Corrective Actions** during Execution
- **Termination** and **Decommission** of the Service

There is still ongoing discussion as to whether the creation of a service and SLA template which identifies and defines the non-functional quality of service attributes and the price (having strong impact on all later stages of the SLA lifecycle including negotiation, monitoring and service provisioning) should be part of the lifecycle or not. In some research activities it is seen as a predecessor of the lifecycle, however this thesis makes the case that it is part of the lifecycle. This phase is represented in 1.3 as the Development Phase.



Figure 1.3.: The different phases of the SLA lifecycle

The SLA lifecycle phases can be mapped to the service lifecycle phases (cf. Table 1.1).

Service Lifecycle Phase	SLA Lifecycle Phase
Service Design	Development, Creation
Service Transition	Creation, Service Provisioning and Deployment
Service Operation	Execution, Assessment, Decommission

Table 1.1.: Mapping of the Service to the SLA Lifecycle Phases

What can be noticed here is that the creation phase is double-assigned. This is due to the fact, that it has impact on the Service Design as well as on the Service Transition phase.

The following subsections will provide further detail of the different SLA lifecycle phases. Within the description, some roles will appear, also made visible in use case diagrams.

They are defined as follows:

Role	Description
(Service) Provider	The (legal) entity providing a service. Usually the service is provided in return for money. Providers usually have business relationships with Customers. SLAs are negotiated autonomously between Customers and Service Providers to enable the automatic execution of a Service. If the Provider is not a legal entity, suitable governance needs to be in place to permit independent binding resolution of disputes between the Provider and the Customer
(Service) Customer	The Customer of a service. The entity defining the scope of the service (his demands) and acting as direct business partner with the Service Provider
(Trusted) Third Party	The Trusted Third Party is a entity that provides several services on an outsourcing level, if either Service Provider or Customer don't want or are not able to use existing services in their own domain. This party only needs to be trusted within their official competence (e.g. to provide an SLA Template Discovery Service) and this degree of trust may be capturable with a contract.

Table 1.2.: The different roles within the SLA lifecycle

Chapter 2.

The Requirements For This Thesis

2.1. From Use Cases Towards Requirements

This chapter will provide the foundation for the research and development results presented in later chapters, namely the SLA Schema and the according SLA Management Architecture. In the first section, three scenarios will be outlined and analysed on their requirements on an SLA Schema and its creation. At the end of this chapter, an overview of requirement topics will be provided which will act as basis for the further developments. To gain the requirements for the developments within this thesis, a set of use cases is needed, which represents different aspects of Service Level Agreement usage in the HPC domain by implying their contents. For that purpose, three scenarios were chosen:

- From the eHealth domain for time critical medical support for treating cardiovascular diseases, which delivers requirements from the domain of Urgent Computing
- From the automotive domain focusing on car crash simulations which implies the need for a stable environment for a dedicated time span
- From the engineering domain focusing on aerodynamic drag simulations, demanding advance reservations for certain resources

After a short overview of those use cases, details will be provided which lead to a in-depth analysis of the scenarios with respect to their requirements/demands on a HPC SLA Schema. At the end, a list of requirements will be retrieved, which will consist of *scenario specific* demands.

To allow for a better classification of these requirements, Section 2.3 will group them in different main topics to provide the necessary abstraction for the evolution of the SLA Schema in chapter 5.

2.1.1. Medical support through simulation - The MS Use Case

2.1.1.1. Background

With the on-going advent of developments in the HPC area, especially in the provisioning of these resource to its customers, the research domains and possibilities have evolved constantly. One of these domains is the eHealth area, in which specialists already make use of the capabilities provided by High Performance Computing. With positive experiences in the past and on-going research in other areas, especially in the field of real-time simulations, the research foci in eHealth have also adapted now towards "on-the-fly" usage of resources to ensure a better and more reliable patient treatment. This "immediate" usage is part of the new upcoming domain of Urgent Computing which covers an orientation towards usage of HPC resources to perform tasks which provide the basis of decision making systems. As decisions are usually needed quickly, Urgent Computing will provide the means to get prioritized, immediate and reliable access to High Performance Computing systems.

Intentionally, the use case from the eHealth domain for this thesis is chosen as

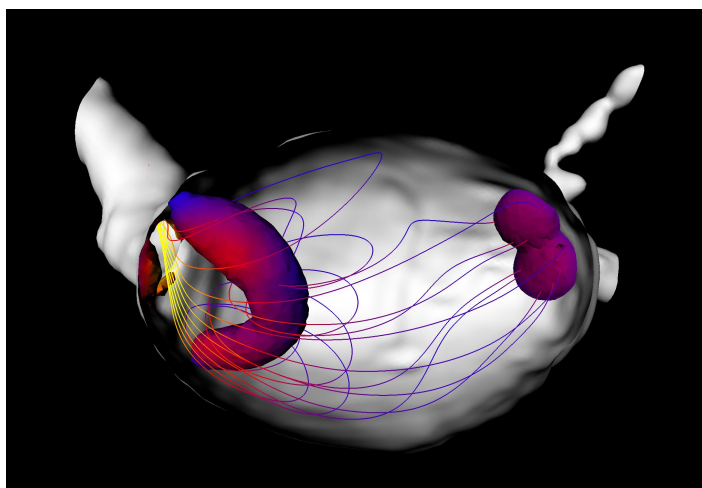


Figure 2.1.: Simulation of an aneurysm

an example where Urgent Computing is definitely needed and by that will deliver partially different requirements than the other scenario. This scenario is dealing with the treatment of cardiovascular diseases (CVD) as addressed e.g. by Haines et al. [16] or Manos et al. [17].

According to the World Health Organisation, WHO¹, CVDs are responsible for a high number of deaths in the developed world (cf. also the World Health Report

¹WHO - The World Health Organization, <http://www.who.int>

2008 [18]). As these diseases are often caused by anomalous blood flows in the surrounding areas of bifurcations as well as by brain aneurysms, a very supportive mechanism is the simulation of exactly the blood flows and to see what the effects of (mainly surgical) treatment could be. By that, there is no danger for the patient due to invasive "testing" activities, but a non-invasive experiment which can increase the rate of success of his treatment.

With HemeLB 2D MRI (Magnetic Resonance Imaging)/MRA (Magnetic Resonance Angiography) pictures can be evolved to a 3D map of vasculature. In this map parameters for velocity and pressure are set which leads then to the simulation of blood pressure. Finally a detailed, interactive visualization of blood velocity is available which supports surgeons to predict the impact of their activities. With this, the surgical planning can be improved to a higher and more patient-protective degree.

2.1.1.2. The concrete scenario

The "Goodwill" Hospital in Castle Rock gets a new patient with a committal from his family doctor. The doctor's suspicion is a cerebral bleeding. Immediate examinations with a computed tomography confirm this suspicion. Based on the first results, decision is taken that a neurosurgery has to be performed within the next 12 hours.

To ensure the best treatment, the surgeon wants to plan the surgery as best as possible in advance but additionally wants to have the possibility to align his strategy during the surgery with additional simulations. As the surgery itself will take approximately 5 hours, there are in a maximum 7 hours left for preparation, the faster, the better.

Since some months, the hospital uses the HemeLB [19] application, especially in the neurosurgical area which was developed and implemented within the scope of the GENIUS² [20], [21] project and focuses on fluid flow in sparse geometries. Basis for this simulation is the Lattice-Boltzmann Method [22], [23] which provides a good technique for computational modelling of complex fluid flow problems. As input a magnetic resonance angiogram dataset is needed, which is immediately produced. In the meantime, the hospital needs to find a HPC centre, providing the capabilities and having the free capacities to perform the simulation immediately. If possible, for the best price available. Additionally updates of the simulation need to be performed during the surgery. From experience the surgeon knows that with the traditional approach of getting a 3D dataset and the following evaluation he usually needs 20 minutes until he can perform the next embolisation. This means that every method which shortens this time of the "hpcless" method is an improvement, but as he does not know how many embolisations he will have to perform, he will need guaranteed access to the systems, to

²GENIUS - Grid Enabled Neurosurgical Imaging Using Simulation, <http://wiki.realitygrid.org/wiki/GENIUS>

perform the simulation for each bit.

By intention this is a visionary scenario, as this is not yet used in practice anywhere, however the main technological innovations for such a scenario have been reached already. At the same time, this scenario delivers the "raison d'être" for Urgent Computing in this field. Imagining current procedures such as putting the simulation request in a job queue would be simply not possible as this kind of simulation has to be performed extremely quickly or it will be of no help at all. For the doctor, as well as the treated patient, it is of no help, if the simulation is done "soon", they need it within a certain time frame to be able to adapt their plans.

2.1.1.3. Requirements of the Medical Support Use Case

Looking at this scenario, it's obvious that there are two main demands here on: **reliability** and **timing**. If the surgeon gets the data later than approximately 4 of the 7 hours of the preparation he won't be able to use it adequately any more and it would provide no benefit for the treatment. Additionally during the surgery, if the time until the necessary simulation has been performed exceeds the 20 minutes, it's the same way useless (even though maybe more concrete, but there are natural limitations by health which dictate the timing). On the other hand, the surgeon also wants to insure himself against possible actions for damages resulting from a misbehaviour of the HPC provider (non-availability of the resource).

Additionally there is already a requirement on the establishment of an SLA in terms of timing, as there is not much time for long negotiation and exchange of offers and counter offers due to the urgency of the case.

The requirements from the end user perspective with respect to the contents of a Service Level Agreement are listed within Table 2.1.

2.1.2. Improved car design through simulation - The CCS Use Case

2.1.2.1. Background

Safety, reliability and performance. These parameters are indicators for the quality and the success of a car type but not all of them are easy to test and to increase. A valuable possibility for supporting the evolution of car designs is the usage of High Performance Computing and by that the integration of Computer-aided engineering

Req. Nr.	Definition	Example
R1	Setting a maximum run time until when the service results are needed (independent of their quality)	Maximal 4 hours after providing the MRA data
R2	Guarantee of permanent access to the service	Guarantee is given by the provider
R3	Setting the timespan in which the guaranteed access is possible (as described in R2)	1 hour before the surgery and the whole surgery (5 hours) + 1 hour buffer after the surgery
R4	Guarantee of availability rate	>99,9%
R5	Definition of the requested application	HemeLB
R6	Guaranteed Compensation if the service fails	Money
R7	Limited Time for SLA establishment has to be taken into account	Limited validity of a request for 20 minutes

Table 2.1.: Requirements of the End User in the Medical Use Case

(CAE) [24] and Computer-aided design (CAD) [25] in the design and realization cycles. By that, the development of new prototypes is now split in two cycles, first a virtual prototyping phase and second then a real-world prototype. Amongst others, the fields of component fatigue analysis [26], car and passenger safety and computational fluid dynamics (CFD)³ are applications giving the highest benefit to this adapted procedures. This applies for the common car types as family cars as well as for racing cars and allows the manufacturers to save time and costs for performing their business. In the automotive sector, CFD is widely used, e.g. for the purpose of gaining knowledge about the aerodynamics of a car, the climate control, engine cooling etc. By that, the use of CFD for designing certain car components (e.g. front wings) can allow for pre-prototyping a candidate to be tested later as real component in a wind tunnel. Without CFD, a huge set of candidate front wing designs would have to be produced and tested (and most of them would be thrown away).

One of the fields today that is most highly dependent on computer simulation is car crash testing, as performing these tests with real cars would be first of all very costly and secondly not give a complete insight in what happens during a crash. By performing car crash simulations, the car makers can simulate the crash behaviour of individual car body components, their reciprocity with other components as well as the impacts on the passengers. Additionally the crash simulation is not for single-use

³Computational Fluid Dynamics - CFD, http://en.wikipedia.org/wiki/Computational_fluid_dynamics

purpose (as a prepared car would be) but can be repeated with adapted parameters again and again whilst not changing the other bits. Example software packages for performing these collision simulations are LS-DYNA [27] and PAM-CRASH [28].

2.1.2.2. The concrete scenario

In this scenario, a car manufacturer is planning the development of a new family car. For that purpose, the designing team is working on adapting a predecessor of this new car to address more requirements which families may have. After a planning phase and modelling of a first prototype, they want to see the behaviour of this car in case of a crash. Therefore, they want to run several simulations sequentially whilst intermediary adapting the design of safety relevant components like air bag or roll-over bar based on the results of the single runs. To ensure that they have a stable basis for the simulations, they need a stable environment which consists, amongst others, of Solaris v10 as operating system, Java v1.6 update 7 and LS-DYNA V971 R3.1 (R3.43919). It has to be clear that as long as they are running their jobs, a change to the system is not allowed by any of the administrators to avoid making the result incomparable. In addition, there is a huge demand on security of the commercially sensitive data here, as this data would be of high interest for competitors and therefore must not be accessible for outsiders by any means.

2.1.2.3. Requirements of the Car Crash Simulation Use Case

This scenario is a stakeholder for those cases, where end users have demands on **exclusive access to resources** and **the set-up of the resources**. If the system properties in terms of Operating System (OS) (here Solaris v10) or installed software (amongst others Java 1.6 Update 7) would change during their simulation executions, this would have a bad implication on the gained data, which is not acceptable by the car manufacturer.

At the same time, they want to ensure, that the required resources are exclusively available for them and that all proceeded and generated data is protected.

So the requirements from the end user perspective to the contents of a Service Level Agreement are:

Req. Nr.	Definition	Example
R4	Guaranteed availability rate	>80,0%
R5	Definition of the requested application	LS-DYNA V971 R3.1
R6	Guaranteed compensation if the service fails	Money
R8	Definition of the software stack	Solaris
R9	Definition of installed software/application	e.g. Java 1.6
R10	Guaranteed time of stable environment	10 weeks from now, no updates, patches...
R11	Definition of the resources	e.g. Vector machine, 16 nodes...
R12	Definition of permanent storage	100 GB
R13	Exclusive access to the resources	No other user on the resources
R14	Confidential treatment of provided and processed data	Only authorized access to data

Table 2.2.: Requirements of the End User in the Car Crash Simulation Use Case

2.1.3. An visualization Scenario - The VIS Use Case

2.1.3.1. Background

As already pointed out for the car manufacturers domain, product design processes can be considerably improved by using the capabilities of supercomputing. Bound to this is often the visualization of these results (as assumed to be integrated in the previous scenario) to use the advantages of virtual and augmented reality. Software suites like COVISE⁴ (COLlaborative VISualization and Simulation Environment) are supporting engineers in designing their own work flows taking into account results of previously executed simulations on dedicated hardware.

These software suites are usually designed to use high performance computing infrastructures of parallel and vector machines and fast networks. To support the end user, COVISE for example, delivers the means for visualisation on desktop computers but also on power walls and as highest level on "immersive virtual reality facilities designed for the exploration of and interacting with spatially engaging environments" such as shown in Figure 2.2 (CAVE - Cave Automatic Virtual Environment⁵).

With this visualization, the users are enabled to analyse their datasets/results "within

⁴COVISE - COLlaborative Visualization and Simulation Environment, <http://www.hlrs.de/organization/vis/covise/>

⁵National Center for Supercomputing Applications, <http://www.ncsa.illinois.edu/>

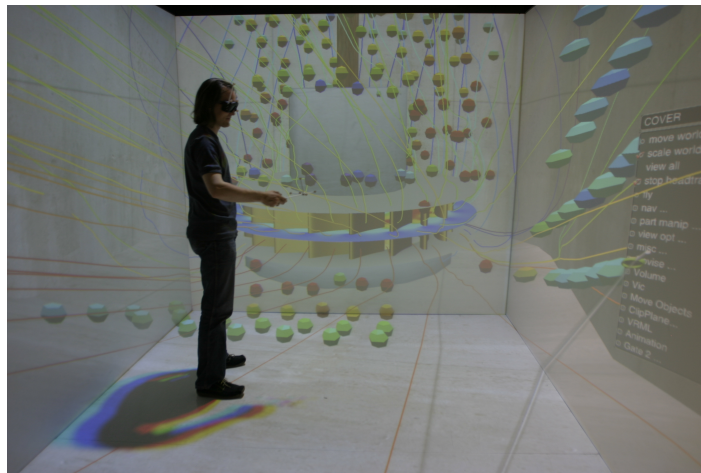


Figure 2.2.: Visualization in a CAVE

the scene" which allows for enhanced evaluation. Furthermore physical experiments or prototypes can be incorporated.

2.1.3.2. The concrete scenario

In this third scenario, a Formula 1 team is designing the new race car for the next year tournament. Within this scope, they want to evaluate simulations on their car prototype with respect to airflow around the car in comparison to the real prototype in a wind tunnel. The tunnel is only rented for some hours, as building up and maintaining an own wind tunnel would be too costly for the team.

The simulation and the real experimental data should be overlaid and visualized via COVISE. This implies a strict timing as exceeding the rented time for tunnel would be too cost intensive. So there is a clear demand (amongst others) on the availability of the simulation environment on the fixed date at the fixed time of the wind tunnel's availability.

2.1.3.3. Requirements of the Visualization Use Case

Requirements of the Visualization Use Case are mainly based in the domain of **advanced reservation** of resources. The car manufacturer wants to overlap the simulation visualization with the video of the real world experiment in the wind tunnel. This implies that all necessary computational resources have to be available at the time, the experiment is performed to allow for a correct evaluation of the experiment data. Additionally it is absolutely necessary to have a high availability

rate of the simulation service resources as the real world experiment will continue, independent of the state of the simulation. Having a drop-out would mean to lose important data for a highly qualitative result. At the same time, there is also a big confidentiality issue here as the car manufacturer is in permanent competition with other car manufacturers and by that has to ensure that information about his simulations or their results are not changing hands. The requirements from the end user perspective with respect to the contents of a Service Level Agreement are listed within Table 2.3.

Req. Nr.	Definition	Example
R2	Guaranteed permanent access to the service	Guarantee is given by the provider
R3	Ability to set a timespan in which the guaranteed access is possible (as described in R2)	shortly before, during and some time after the experiment
R4	Guaranteed availability rate	>99,0%
R5	Definition of the requested application	ANSYS CFD
R6	Guaranteed compensation if the service fails	Money
R13	Exclusive access to the resources	No other user on the resource
R14	Confidential treatment of data	Only authorized access to data
R15	Setting the start time of the service (and by that the access to the resources)	When the real world experiment starts
R16	Real-time streaming capabilities of the network	2.1 Mbps
R17	Guaranteed latency for data access	Response Time <100ms

Table 2.3.: Requirements pf the End User in the Visualization Use Case

2.2. Non scenario specific requirements

Targeting exclusively the worked-out requirements from the three use cases presented in the previous section would extremely limit a the applicability to other HPC-related use case.

To avoid this limitation, this chapter will present the non scenario specific as retrieved from studying other approaches for SLA Management such as NextGRID, BEInGRID and BREIN .

Involved Business Entities

As defined before, Service Level Agreements shall define bipartite relationships between different business parties. To allow for a proper identification of the involved business entities and to get necessary information in terms of contact persons or details.

Req. Nr.	Definition	Example
R18	Contact details of the involved parties should be retrievable	Name, Street, City and more of the service provider and customer

Table 2.4.: Non Scenario specific Requirement for Contact Details

Traceability Of Origin

When the SLA is fixed, the base on which it was designed in terms of service description is not of interest any more for the customer. In opposite, the service provider may have a high interest in this traceability, as it allows him in case of violated SLAs which were designed on the same basis to find exactly these problematic terms. Therefore this information should be retrievable during the execution of an SLA.

During the phase of establishing the SLA, additional information about who is the originator of a document and who is the receiver of it.

Req. Nr.	Definition	Example
R19	The origins of the SLA should be traceable	SLA X was based on Service Description Y

Table 2.5.: Non scenario specific requirement for Traceability of the Origin of an SLA

Validity Of The SLA (Terms)

Commonly a contract has only a limited duration of validity. This has to be inherited by Service Level Agreements to ensure that the service is only available and accessible during this period. Additionally there could also be a case where different terms have different durations of validity, what should also be addressable.

Req. Nr.	Definition	Example
R20	The validity of the SLA and its terms should be definable	SLA X is valid from today 14:00 CET until 15:00 CET

Table 2.6.: Non Scenario specific Requirement for the SLA Validity

Invocation Limit

There are application areas which allow their customers only a limited number of invocations of a service. This should be representable within the Service Level Agreement document, as well as it should be monitored accordingly.

Req. Nr.	Definition	Example
R21	Define number of invocations	Service may be used a maximum of 10 times

Table 2.7.: Non Scenario specific Requirement for the Limitation of Service Invocations

Data Handling

In the use cases, the requirement for defining the confidentiality level of data access was listed (R14), but there are more data-handling related terms which should be taken into account.

Besides the accessibility of data, there is also a need for additional data mechanisms such as backup processes, which are of high importance. If a job execution delivers intermediate results, but in the middle, the resources fail, the data gained until then would avoid a restart of the job from the beginning and by that save time and probably money.

Associated with this, but not exclusively only for backup mechanisms, the use of data staging should be possible to allow for a well defined movement of data before, during and after the execution time.

Finally, especially in highly confidential industrial areas, there should be the possibility to define the kind of data storage that is used. This is analogue to the exclusive access to system resources and should provide the means to state if the data storage is exclusively for the customer or if it is shared with other users.

Req. Nr.	Definition	Example
R22	Definition of backup mechanisms	Full data backup every hour
R23	Definition of data staging mechanisms	Move file X as source file to host Z
R24	Definition of the kind of data storage	Storage Y is exclusively reserved for customer M

Table 2.8.: Non Scenario specific Requirement for the Handling of Data

Business Definitions

In addition to the requirement for defining compensations in case the service fails (Requirement R6) some other items need to be taken into account, when designing the SLA Management Framework and the Schema.

Complementary to the penalties, it should also be possible to define rewards in case (a) a service has a better quality than agreed or (b) additional (but defined as optional) QoS parameters have been provided.

Furthermore there are more important terms which have to be fulfilled than others, which service providers or customers would like to label accordingly.

Finally, pricing of services should allow for dynamic adaptations. That leads to the need for support of different pricing models such as *package price* or *price per unit*. In a summary this means that the following requirements have to be fulfilled:

Req. Nr.	Definition	Example
R25	Definition of Rewards	If the service finishes earlier than expected, 100 Euro will be charged extra
R26	Support of different pricing models	Price per package
R27	Definition of importance of terms	Term A is less important than term b
R28	Provenance	Data provenance, recording of origin and movement of data

Table 2.9.: Non Scenario specific Requirement for Business Definitions

2.3. Categorization of Requirements

To prepare the definition of the SLA Schema accordingly, the gained requirements are categorized into a set of topics which should then act as basis for the design of SLA "building blocks". These topics are

2.3.1. Resourcing

Resourcing is representing everything which covers the necessary infrastructure and software environments as requested by the user. Amongst other this covers:

- Operating system (e.g. Solaris)
- Resource type/Processor Type (e.g. Vector machine, Quad-Core AMD Opteron(tm) Processor 23 (C2))
- Software/ Applications (e.g. Java, LS-Dyna)
- Versioning (bound to OS or Software, e.g. Solaris 10)
- Network Speed (e.g. Real-time streaming capabilities with 2.1 Mbps)
- Storage (e.g. 100 GB permanent storage)
- Access Rights (e.g. access to the resources is exclusively)
- Maintenance of the system (e.g. Stable environment for 10 weeks)

This was implied by requirements R5, R8, R9, R10, R11, R12, R13, R16.

2.3.2. Service Availability

Availability covers all the terms within an SLA which refers to the time constraints of the service. Amongst others this covers:

- Start time of the service (e.g. in 1 hour)
- Maximum runtime (e.g. for 15 minutes)
- Accessibility time (e.g. the Service needs to be accessible the next 48 hours)
- Fixed end time (e.g. 18:00 CET)
- Minimal/maximal response time (e.g. latency < 100 ms)

- Availability rate of the service (e.g. 99,9 %)
- Limited number of invocations (e.g. 10 invocations of the service are allowed)

This was implied by requirements R1, R2, R3, R4, R10, R15, R17, R21.

2.3.3. Business Handling

This category covers the business aspects of a SLA in terms of monetary issues as well as classification of importance of terms etc.

- Penalties (e.g. 1000 Euro payment when delayed)
- Rewards (e.g. 100 Euro if it is earlier delivered)
- Pricing Model (e.g. 100 Euro per Unit)
- Importance of Terms (e.g. term A is more important than term B)

This was implied by Requirement R6, R25, R26, R27.

2.3.4. Data Treatment

The handling of data of all kind (input, output and intermediate results) is addressed by the terms within this category. They are namely:

- Backup of Data (e.g. hourly full data backup)
- Data Staging (e.g. move file X to host Z before the job is executed)
- Data Storage (e.g. storage for customer M is shared with other customers)
- Data Provenance (e.g. link to a Data Provenance Recorder)
- Data Treatment (e.g. only authorized access to data)

This was implied by Requirement R13, R14, R21, R22, R23, R27, R28.

2.3.5. Contractual Terms

This term-set covers contractual issues, which can influence the establishment of an SLA.

- Contact Details (e.g. name, address of the service provider)
- Origin of the Document (e.g. the SLA was based on document Z of the service provider)
- Originator of the Document (e.g. the request for an SLA was initiated by customer X)
- Validity of the SLA (e.g. this SLA is valid from Monday 13:00 CET to Tuesday 13:00 CET)
- Timeframe for establishment of an SLA (e.g. within the next 20 minutes)

This was implied by Requirement R7, R18, R19, R20.

Chapter 3.

Related Work

This chapter will provide the basis for the developments in this thesis. Even though currently existing technologies/approaches are not mature enough to be used as they are in eBusiness, effort was taken to bring them to their current state and partially they already address the needs of SLA Management within the HPC domain. For that reason, the following sections will give an overview on the features and missing bits of the existing State of the Art and reason why (and how far) this thesis has to go beyond.

3.1. Middlewares

Middlewares as such is a wide field with many different players. This section will concentrate on the most prominent ones and show what they currently can do in terms of SLA Support. At the end of this section, a careful analysis will be given, showing what is still needed.

3.1.1. Globus Toolkit 4

The Globus Toolkit is one of the most popular Java based middlewares for Grid which delivers core service and defines interfaces and protocols to allow users to remotely access Grid resources. It includes software for security, resource and data management, communication, fault detection and monitoring and is built with the principle that even single services can be standalone used for developing applications in the Grid. The current version GT4 is based on the Open Grid Services Architecture - OGSA [14] and implements a set of Web services specifications such as WS-Notification [29], WSRF - Web Services Resource Framework [30] , WS-Security [31]

and WS-Addressing [32]. The license of GT4 is the Globus Toolkit Public License (GTPL) which is Apache-2 compliant.

Support of Service Level Agreements

GT4 as such does not provide any support of Service Level Agreements at all within its implementations but provides base services, which could be used to support the whole SLA Management lifecycle (see Chapter 1.3.1).

3.1.2. Unicore 6

The Unicore middleware was initially developed within two national German research projects from 1997-2002 [33]. Since then, there has been a continuous activity of further developments in EC funded project activities to establish a uniform access to distributed computing resources. Since 2004 it is based on an open source community development, taking into account the requirements and comments from HPC users, HPC user support teams and HPC operations teams. Unicore is already used and deployed for a set of communities/projects, amongst others DEISA [34], the Gauss Center for Supercomputing [35] or D-Grid¹.

UNICORE is a Java based grid environment and realizes/uses many Web Services/-Grid specifications such as WS-Security, WS-Addressing [36], WSRF or JSDL (cf. Chapter 3.2.6). UNICORE 6 is the most recent version of the UNICORE technology compliant with on OGSA in which WS-ResourceFramework is integrated and adopted. Unicore 6 is available under BSD license.

Support of Service Level Agreements

Unicore 6 as such has no dedicated Service Level Agreement Management components but provides basis implementations, which could be used to support the whole SLA lifecycle. In terms of the creation phase, developments based on Unicore 6 in the Chemomomentum [37] project targeting reservation of resources could be of interest, but were not integrated in Unicore 6 yet (even though proposed).

3.1.3. GRIA

GRIA - Grid Resources for Industrial Applications [38] - was initially developed within the frame of an European Project and is still further evolving. It provides a

¹D-Grid, <http://www.d-grid.de/>

service-oriented infrastructure to support Business-to-Business relationships in terms of service provisioning across organizational boundaries.

It makes use of business models, processes and semantics to allow resource owners and users to discover each other and negotiate terms for access to high-value resources. It implements an overall business process to find, procure and utilize resources capable of carrying out high-value, expert-assisted computations. By focusing on business processes and the associated semantics, GRIA enables users to provision for their computational needs more cost effectively, and develop new business models for some of their services. Services from different providers can be combined together to create applications using a simple and easy-to-use API.²

The implementation of GRIA is provided package-wise, amongst others a service provider management package is available, which may support SLA-based management. Most of the software is provided under the Lesser General Public License (LGPL) .

Support of Service Level Agreements

As mentioned above, GRIA provides the means of Service Level Agreement support within the service provider Management. This support is given by a dedicated SLA Service which allows for definition of available resources (CPU , disc, applications, DBs , licenses, etc.) and provides SLA templates that a user can propose, defining QoS. Furthermore it provides support to agree new SLAs up to the limits of the resource capacity.

3.1.4. gLite

The gLite middleware is an output from the EGEE Project. Its development was done based on the Large Hadron Collider Computing Grid (LCG) [39] and focuses on the scientific community. The middleware integrates components of other (grid) middleware packages, such as Condor [40] and the Globus Toolkit and is compatible with schedulers such as Portable Batch System - PBS [41] and LSF [42] built with interoperability in mind and provides foundation services that facilitate the building of Grid applications from various fields. In terms of implementation, there is a Java version available, distributed under an open source license. The gLite Grid services follow the Service Oriented Architecture approach which allows for an easy interaction with other Grid Services.

²Grid Resources for Industrial Applications, <http://freshmeat.net/projects/gria>

Support of Service Level Agreements

The gLite middleware provides an Agreement Service (within a number of so-called Helper Services) which is based on WS-Agreement. This service can check the compliance of an Agreement Offer document to the internal templates and can interact with a variety of resource reservation services and request a resource when required.

3.1.5. Summary

Only two of the four middlewares presented provide or at least support SLA Management capabilities. Whilst Globus Toolkit 4 and Unicore 6 provide no dedicated SLA support, gLite provides the Agreement Service, which is based on WS-Agreement and may be used in a long term for embedded SLA Management in gLite. Nonetheless this service is still immature and gLite is far away from a state where SLAs are supported in a proper way (as also deduced by the BEinGRID project in its middleware analysis³).

Compared to the other three analysed middlewares, GRIA provides a dedicated SLA Management functionality. The GRIA SLA Management service is thereby focused on development and negotiation, monitoring and enforcing of Service Level Agreements, covering parts of the SLA Lifecycle as depicted in Section 1.3.1.

However, looking at the application domain of this thesis, GRIA also has drawbacks. Especially in the area of dynamic service provisioning, there is no support yet available, even though this is a topic of high interest for HPC providers. Additionally, the GRIA SLAs are based on an own SLA Schema, which is not standardized and by that hard to adapt for the needs of HPC. In general, GRIA adverts its wide focus of the SLAs which in the special case of this thesis is more a hitch than a benefit.

3.2. Service Level Agreement and Term Description Languages

3.2.1. Web Service Level Agreements - WSLA

WSLA - Web Service Level Agreements was intended to provide a specification for the definition and monitoring of Service Level Agreements in a Web Service environ-

³IT-Tude, gLite analysis, <http://www.gridipedia.eu/glite.html>

ment. In contrast to WS-Agreement (cf. 3.2.2) it does not handle the publication or negotiation of SLAs at all. The latest version of the WSLA specification was published in 2003 by IBM [12] and there were no more updates since then. However, WSLA is, besides WS-Agreement, currently one of the two most referenced and popular specifications for Service Level Agreements. WSLA provides an SLA language, which is based on XML and by that allows for using/transferring parts of electronic contracts in a machine readable format. It thereby complements common service descriptions like e.g. WSDL. Whilst WSDL itself only describes the Web Service Interface(s), Web Service Level Agreement adds the quality of service parameters/characteristics and how they can be monitored, evaluated and assessed (cf. 3.1, as taken from the WSLA specification). Within the WSLA specification, IBM provided an run-time architecture

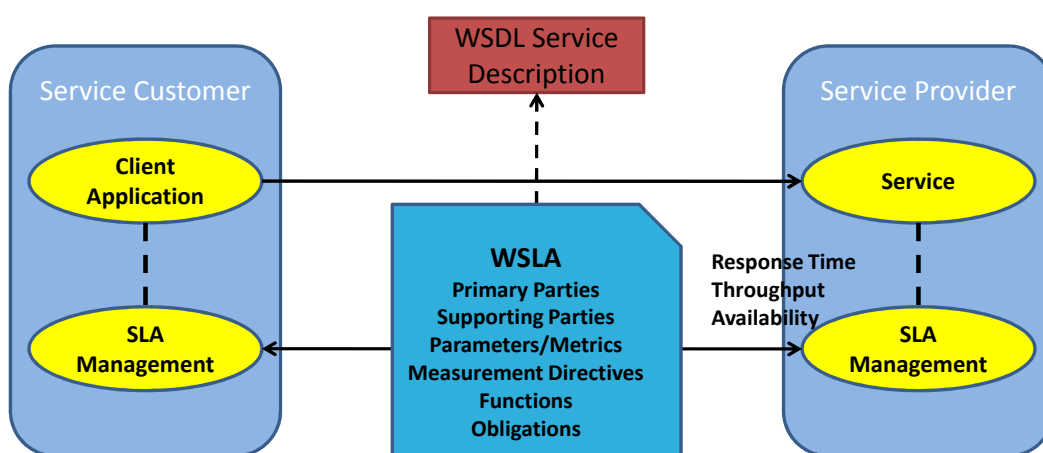


Figure 3.1.: The WSDL - WSLA Relationship

which comprises several SLA monitoring services. By introducing the concepts of integrating monitoring services provided by so-called supporting parties, WSLA allows its users to delegate certain tasks to a trusted outsider. In this context, the third parties involved in monitoring these bipartite relationships are also often called Trusted Third Parties as they imply a certain degree of trust for performing their tasks objectively.

In terms of describing Service Level Agreements, the WSLA language is structured as follows: Figure 3.2 shows the three main sections of an SLA represented with WSLA.

- **Parties**

This section represents the organizational data of the business entities involved in this Service Level Agreements. As Service Level Agreements represent bipartite relationships, the number of signatory parties is limited to a maximum of 2, whereas the number of supporting parties (the trusted third parties) is

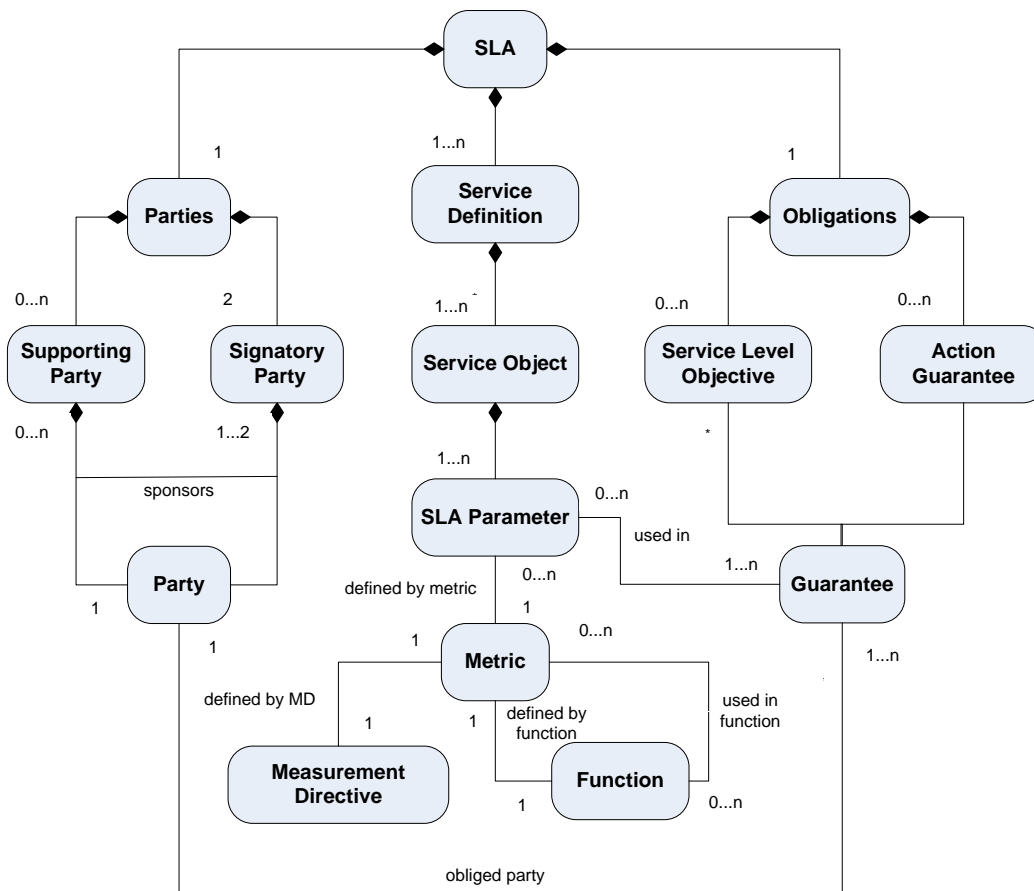


Figure 3.2.: The WSLA Structure

theoretically infinite. The description of a party covers contact information as well as WSDL definitions to the services of the organization which are exposed to the other entities.

- **Service Definition**

To enable a reliable and sufficient monitoring of the services (and their behaviour) the relevant service level parameters, as well as the respective metrics, need to be defined. This happens in the Service Definition section of WSLA, which contains a comprehensive overview of the SLA Parameters and also provides the means to define which party (especially relevant for supporting parties) is measuring which metric and the way how to do it.

- **Obligations**

The obligations section contains information about the different service levels, guaranteed with respect to the SLA Parameters (as stated in the Service Def-

inition part). This part contains the Service Level Objectives (SLOs , usually bound to a specific duration), and additionally the actions that should be taken in case of a violation of one or more of the SLOs.

Even though no longer pursued, WSLA already reached a semi-stable state in 2003 and is still the most mature specification available to define the contents of a Service Level Agreement. In combination with WS-Agreement it could provide a good technological basis for supporting the complete SLA Lifecycle. Some projects, e.g. TrustCoM [43] and BREIN [44], have tried to merge WS-Agreement and WSLA for their implementations. However, a dedicated impact on standardization was not yet reached due to ongoing discussions about the copyrights of WSLA as this is an IBM specification.

3.2.2. WS-Agreement

The Web Services Agreement Specification from the Open Grid Forum (OGF⁴) describes a protocol and a structure for establishing an agreement on the usage of Services between a service provider and a consumer. It defines a language and a protocol to represent the services of providers, create agreements based on offers and monitor the agreement compliance at runtime. WS-Agreement provides a layered model, as shown in Figure 3.3. An agreement defines a relationship between two parties that is dynamically established and dynamically managed. The objective of this relationship is to deliver a service by one of the parties. In the agreement each party agrees on the respective roles, rights and obligations. A provider in an agreement offers a service according to conditions described in the agreement. A consumer enters into an agreement with the intent of obtaining guarantees on the availability and qualities of one or more services from the provider. Agreements can also be negotiated by entities acting on behalf of the provider and / or the consumer (as e.g. presented in [45]). An agreement creation process usually consists of three steps: The initiator retrieves a template from the responder, which contains information about 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. An agreement consists of the agreement name, its context and the agreement terms. The context contains information about the involved parties and meta-data such as the duration of the agreement. The concrete structure of an Agreement and a Template as defined by WS-Agreement can be found in Section 5.2.

Agreement terms define the content of an agreement: Service Description Terms

⁴The Open Grid Forum, <http://www.ogf.org>

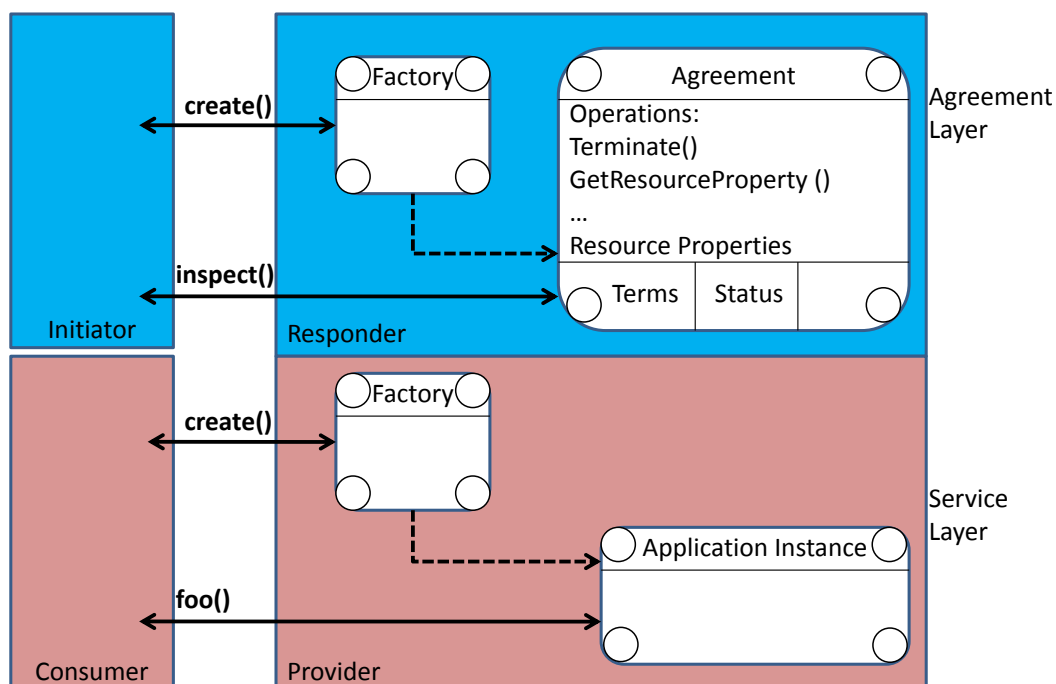


Figure 3.3.: The WS-Agreement Conceptual Layer Service Model]

(SDTs) define the functionality that is delivered under an agreement. A SDT includes a domain-specific description of the offered functionality (the service itself). Guarantee Terms define assurance on service quality of the service described by the SDTs. They define Service Level Objectives (SLOs), which describe the quality of service aspects of the service that have to be fulfilled by the provider. The Web Services Agreement Specification allows the use of any domain specific or standard condition expression language to define SLOs. The specification of domain-specific term languages is explicitly left open to allow for flexible usage what at the same time makes it more difficult to use this specification standalone. As already mentioned in the WSLA section, a merge of WS-Agreement and WSLA is already used by some project initiatives (e.g. TrustCoM or BREIN), but there are also additional specifications which may help to detail the definition possibilities whilst keeping to the WS-Agreement base structure.

3.2.3. Semantic WS-Agreement Partner Selection (SWAPS)

At the 15th international conference on World Wide Web, Oldham et al. [46] presented an approach of semantically enhancing WS-Agreement. Their approach was

motivated by trying to overcome the limitations of WS-Agreement with respect to syntactical matching to allow for intelligent partner selection based on Service Level Agreement technologies. The nature of these limitations is the plain XML based domain vocabulary used in this specification. The main focus of this proposal lies on adaptations to the Guarantee Terms of WS-Agreement (cf. Figure 3.4) which consists of tags for *Service Scope*, *Service Level Objectives*, *Qualifying Conditions* and *Business Value List*. Adaptation of WS-Agreement starts with adding structure to the

```
<wsag:GuaranteeTerm Obligated="...">
  <wsag:ServiceScope Service Name = "...">...
</wsag:ServiceScope>
  <wsag:ServiceLevelObjective>...
</wsag:ServiceLevelObjective>
  <wsag:QualifyingCondition>...
</wsag:QualifyingCondition>...
  <wsag:BusinessValueList>
</wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

Figure 3.4.: The WS-Agreement Guarantee Terms

Service Level Objectives and Qualifying Conditions. As these tags in the original specification were generic to allow every possible expression as value, this is heavily needed to allow for automatic parsing and reasoning. This additional structure is not developed from scratch, but based on the Web Service Level Agreement Language (WSLA) from IBM, namely using its *expression*, *predicate*, *parameter* and *value* tags. As new bits, (optional) tags called *unit*, *percent* and *OntConcept* were introduced and embedded in the schema. The *OntConcept* annotation tag resolves ambiguities of agreements by linking expression parameters directly to the concrete ontology concepts.

3.2.4. SLAng

SLAng is a language which was developed during the TAPAS [47] project. Its intention was to provide a clear definition of obligations of all involved partners with respect to a certain service quality. The development of SLAng was based on a reference model for inter-organisational service provision on the storage, network, application and finally on the middleware layer. Lamanna, Skene and Emmerich described, already in 2003 [48], this model which also states explicitly the need

for vertical and horizontal Service Level Agreements in the different layers. Now, after the TAPAS project, which ended in 2005, SLAng is a project of the Department of Computer Science University College London and is available as an early implementation on sourceforge⁵. The creators describe SLAng as "a language for concrete service-level agreements currently providing support for ASP SLAs. SLAng is defined using an EMOF⁶ metamodel, with embedded OCL⁷ constraints and natural-language documentation in English. This makes it extremely precise as well as being understandable. SLAng is designed so that all SLAs expressed in the language can be monitored. It includes semantics for administration, which is the process whereby the parties to the SLA agree what penalties are to be paid. It expresses constraints on the accuracy of reports used in administration that probably can be monitored. SLAng can express a number of different types of constraints including: reliability, timeliness, availability, data currency, data recovery constraints and constraints on the lifetime of the agreement. The application of these constraints can be varied according to schedules, the state of the service or the state of an external system that is can be monitored by the parties.⁸" Even though this is rather sound and well thought out, it appears that work on SLAng has stopped in 2006. The last implementation was made available in July 2006 and there is an explicit warning on the official web page that this is a very immature version which should not be used for real SLAs.

3.2.5. Web Service Offerings Language - WSOL

The Web Service Offering Language (WSOL [49]), an XML based specification, enables the formal specification of constraints and multiple classed of services for Web Services. By that it allows Web Services to be provided with different service levels (represented by the different service classes).

WSOL is fully WSDL-compatible and allows formal specification of *functional constraints, QoS constraints, simple access rights, price, management responsibilities and the relationships between service offerings*.

The different classes of a service are called *service offering, SO*. Additionally, WSLO offers the constructs of Service Offering Dynamic Relationship (SODR) which specifies outside service offering and what service offering could be replaced by another service offering in case, the first one cannot be fulfilled. This relationship is specified in a file outside of the WSOL file (to avoid frequent modifications of the service offering definitions). By that is useful for easier selection and negotiation of service offerings

⁵Sourceforge, <http://sourceforge.net/>

⁶EMOF - Essential Meta Objective Facility

⁷OCL - Object Constraint Language

⁸The SLAng SLA Language Website, <http://uclslang.sourceforge.net/index.php>

(and by that also supports dynamic adaptation of Web Service compositions). This specification was the outcome of a PhD thesis in 2004 [50].

3.2.6. Job Submission Description Language - JSDL

The Job Submission Description Language (JSDL) [6] aims to deliver a language which supports the description of requirements of computational jobs. With JSDL an alignment of job descriptions as used in different resource management systems shall be achieved to allow for a good level of compatibility. This specification is developed by the JSDL-Working Group⁹ of the Open Grid Forum and the most actual release is Version 1.0 from 2005.

Three main categories of elements are defined within the specification:

- Job identification requirements (Job name, description)
- Resource requirements (e.g. Operating System, number of nodes)
- Data requirements (e.g. Files that have to be moved to the execution host)

In the current version of JSDL not all elements are given which might be needed to describe certain jobs, but there are already examples of its uptake in commercial resource management systems like the Load Sharing Facility, LSF (see also Chapter 3.3.4). In these cases extensions of JSDL are provided which increase the coverage of potential terms and at the same time are usable by the resource management solution. From a middleware point of view, the most prominent supporting JSDL are Unicore, Globus Toolkit 4 and GRIA.

3.2.7. Summary

In this subsection a set of Service Level Agreement and/or Term Description languages has been presented which all provide different aspects usable for representing Service Level Agreements.

They all own a different level of maturity and were developed independent of each other (besides WS-Agreement and SWAPS). To give a better overview, Table 3.1 shows in how far they address the requirement categories as worked out in Section 2, namely Resourcing (Res), Service Availability (SA), Business Handling (BH), Data Treatment (DT) and Contractual Terms (CT). Additionally information on current maintenance (CM) (is it evolved or has work stopped on an immature level) is given.

⁹JSDL-Working Group, <https://forge.gridforum.org/sf/projects/jSDL-wg>

Generally speaking, most of the presented approaches pretend to address the re-

	WSLA	WSAG	SWAPS	SLAng	WSOL	JSDL
Res	✓	✓ (imp)	✓ (imp)	✓		✓
SA	✓	✓ (imp)	✓ (imp)	✓	✓	
BH	✓	✓	✓	✓	✓	
DT	✓ (imp)	✓ (imp)	✓ (imp)		✓	✓
CT	✓	✓		✓	✓	
CM		✓				✓

Table 3.1.: Frameworks addressing SLA needs

quirements categories as needed. However, it is important to have a closer look at them, by that reason Table 3.1 shows some of the check marks having an additional "(imp)" in their cell. This refers to the fact that especially WS-Agreement (and by that the build upon SWAPS) provide the possibility to define those terms but do not offer a detailed language for that. On one hand this means that the users are free to define their terms as they want which on the other hand leads to the problem that automated procession of the terms/the overall SLA is difficult (if there was no agreement on a common definition language before).

With the focus of this thesis on SLA Management for the HPC domain, two criteria are of paramount importance for the used SLA Schema: Applicability to the desired terms of end users and the maturity and by that stability of the description language. The latter already implies the drop of those languages which are immature and no more evolved. The development of WSLA stopped in 2003 at an already mature state. WS-Agreement itself is continuously evolved within the Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG) at the Open Grid Forum but provides no in-depth design language to define HPC system level terms. SWAPS is built upon it, but was not evolved any more after the presentation of its initial version. With that it lacks of maturity and alignment to WS-Agreement.

SLAng was a very promising approach but by unknown reasons work on it was stopped in 2006 (probably due to the end of funding within projects). Even the website warns of its use due to the immaturity of the work, which is a clear no-go for it. WSOL is currently not further developed. Research rather focus on its application then turning it into a standard or a specification. By this it is still immature and excluded from use in this thesis.

Finally JSDL is, though not being explicitly an SLA definition language, a good candidate for defining terms on the resource and the data availability level.

As mentioned before, recent research activities worked with a merge of WSLA and WS-Agreement and prove benefits arising from this approach. However, solely WSLA is not enough as it is in some parts rather over defined and too complicated

for the HPC infrastructure terms.

This leads to the conclusion that for the aimed SLA Schema for HPC, three candidates are providing the means to describe the SLA terms. Where none of these is applicable, own terms have to be designed. The retrieved schema is presented in Chapter 5.

3.3. Resource Manager/Scheduler

Job Queues or Scheduler will be important in the context of this thesis, as they provide the means for realizing the terms stated in the SLA.

3.3.1. PBSPRO

The Portable Batch System (PBS) was the basis for PBSPRO¹⁰, which is a distributed workload management system providing the means for queuing, scheduling and monitoring. It is the full-fledged and industry oriented commercial follow-on to PBS and allows for enhanced interactive and non-interactive running of jobs whilst taking into account the respective requirements. In terms of scheduler support, PBSPRO is very flexible. Thereby integrated schedulers provide the means of First-Come First-Serve (FCFS), Shortest-Jobs-First (SJF) or user/group priority driven scheduling but the suite itself does not exclude the integration of own schedulers.

PBSPRO allows for definition of different queue types with different features (e.g. with resource limits, queues for short jobs). By that, priority driven queuing is enabled, e.g. providing express queues which are pre-empted on demand to handle jobs of high priority customers. PBSPRO is capable of submission of JSDL jobs by using the Distributed Resource Management Application API (DRMAA)¹¹.

3.3.2. OpenPBS/Torque

According to Cluster Resources¹², "TORQUE (Terascale Open-Source Resource and QUEUE Manager) is an open source resource manager providing control over batch jobs and distributed compute nodes. It is a community effort based on the original

¹⁰PBSPRO, <http://www.pbsgridworks.com/>

¹¹DRMAA - Distributed Resource Management Application API, <http://www.drmaa.org/>

¹²TORQUE, <http://www.clusterresources.com/products/torque-resource-manager.php>

*PBS project and, with more than 1,200 patches, has incorporated significant advances in the areas of scalability, fault tolerance, and feature extensions." TORQUE is a successor of OPENPBS which was, as well as PBSPro a derivate of PBS.

3.3.3. LoadLeveler

LoadLeveler¹³ from IBM¹⁴ is a parallel job scheduling system which takes the requirements from different jobs and aligns them with the resource capabilities of the system. This means that jobs submitted to the LoadLeveler queue are not necessarily processed by order of submission, but based on the priority, resource requirements and special rules, given by the system administrators.

LoadLeveler supports

3.3.4. Load Sharing Facility - LSF

LSF - Load Sharing Facility¹⁵, is a commercial job scheduler from Platform Computing¹⁶. It aims to provide the technology for intelligent management of batch workload processing for mission-critical compute- and data-intensive applications. The LSF software thereby determines the execution environment for the user based on the input data received. The user can not influence on which exact host its job will be performed.

LSF supports several scheduling algorithms such as FCFS, fair-share, pre-emptive, backfilling but also Service Level Agreements. With respect to SLAs, LSF can work towards four different goals which are either deadline, velocity, throughput driven or a combination of the previous three goals. Pre-emption and backfilling capabilities are given and different queue types for different user groups can be easily defined (e.g. express queues).

With respect to the prioritization of jobs, LSF provides an priority escalation mechanism which leads to an increase of the priority of a job every time interval and forces their early execution, even if a fair share scheduler is used.

LSF complies with the Job Submission Description Language (JSDL) and also provides own extensions to this standard for features not addressed by JSDL.

¹³Tivoli Workload Scheduler LoadLeveler, <http://www-03.ibm.com/systems/clusters/software/loadleveler/index.html>

¹⁴IBM, <http://www.ibm.com>

¹⁵Platform LSF, <http://www.platform.com/Products/platform-lsf>

¹⁶Platform Computing, <http://www.platform.com>

3.3.5. Sun Grid Engine - SGE

The Sun Grid Engine¹⁷ - SGE is an open source batch queuing system which provides the capabilities of scheduling, dispatching and managing of the execution of parallel jobs. Additionally it allows for management and scheduling of different resources (processors, disk space, memory and software licenses).

There is also a commercial version of SGE which is called N1 Grid Engine (N1GE) .

3.3.6. Maui

The Maui¹⁸ Scheduler is an external/standalone scheduler, which is freely available as Open Source software. Maui needs to be integrated in existing local resource management systems (LRMSs, e.g. PBSPro, LSF or Loadleveler) as they deliver the information needed for the job scheduling. This brings in additional benefit, as it can replace integrated schedulers of the resource managers.

Maui does not allow for deadline based scheduling but its pre-emption mechanisms and the parameter based administration allow for at least limited QoS support. Priorities for different jobs are defined in Maui with a weighting algorithm over a set of objectives, which can be defined by system administrators.

3.3.7. Moab

Moab¹⁹ is an advanced job scheduler for use on clusters and supercomputers. It is based on Maui and released as a commercial workload management package, extending the capabilities of Maui especially in terms of industrial need. This includes amongst others graphical cluster administration tools, multiple cluster support, advanced grid support and extended resource control. A complete comparison can be found on Maui Cluster Scheduler Website in Appendix K²⁰.

¹⁷Sun Grid Engine, <http://gridengine.sunsource.net/>

¹⁸Maui, <http://www.clusterresources.com/products/maui/docs/mauiadmin.shtml>

¹⁹Moab, <http://www.clusterresources.com/products/mwm/docs/index.shtml>

²⁰Maui Cluster Scheduler, Appendix K: Maui-Moab Differences Guide, <http://www.clusterresources.com/products/maui/docs/a.kmoabcomp.shtml>

3.3.8. Summary

There are several aspects of schedulers and resource management systems, which are of interest for this thesis. Getting back to the SLA Lifecycle and the requirements gained in Chapter 2 this section was intended to show in how far SLAs at all are supported by the current State of the Art in Resource Management Systems and Scheduler.

These systems are of paramount importance in the Negotiation, Provisioning and Execution phase, as they are the connector to the plain HPC infrastructure. By that advanced reservation mechanisms are very important as they support especially the phase of negotiation and provisioning. With advanced reservation the fulfilment of an Service Level Agreement can be ensured and a smooth preparation/start of the customers service is possible.

Secondly, the support of definition of different queuing/scheduling mechanisms for different user groups is crucial, as this will allow for addressing the Business Requirements in the negotiation and provisioning phase, such as the importance of customers, or in case a service is needed really urgent.

As worked out in Section 3.2, JSDL is a well accepted standard for resource term definitions, which is already addressed by some of the Resource Management Systems. With it at least parts of the QoS parameters (in an SLA) could be described on a standardized way.

Table 3.2 shows the comparison of the presented systems with respect to the three capabilities as stated before, namely Advanced Reservation, Queue Variety (Different queues with different priorities for different Customers) and the compliance to JSDL. Looking at the presented Scheduler and Resource Management Systems (RMS) it becomes obvious that they all provide advanced reservation capabilities for clusters. However, the way how advanced reservation is realized differs from system to system. E.g. LSF provides advanced reservation time slots of 10 minutes where PBSPro supports it in time slots of seconds, in Maui the modification of an advanced reservation is possible, in LSF not, etc.

In terms of queue variation the provisioning of special queues for certain user groups is possible with today's state of the art. By that, terms defining the importance of a service may be used to align the systems, on the other hand, it will allow HPC Providers to provide Services (and their SLAs) with different quality levels (e.g. Gold/Silver/Bronze).

Even though it would be beneficial for the provisioning of QoS to the Customers, JSDL is not yet supported by all of the RMS/Schedulers. This lack of a standardized job submission interface is still an obstacle which has to be overcome and can be solved on the middleware level. E.g. Riedel et al. [51] integrated JSDL interfaces within Unicore6 for that purpose.

Capability	PBSPro	OpenPBS	LL	LSF	SGE	Maui	Moab
Advanced Reservation	✓	✓	✓	✓	✓	✓	✓
Queue Variety	✓	✓	✓	✓	✓	✓	✓
JSDL Compliance	✓(imp)	✓		✓	✓		

Table 3.2.: RMS and Schedulers addressing Advanced Reservations, Queue Variety and JSDL Compliance

This leads to the assumption that a proper solution for SLA use in the HPC domain has to address both, direct RMS but also indirect RMS access (via middleware), to allow for proper management of heterogeneous resources which provide different RMS/Scheduling solutions. By that this thesis will not only focus on provisioning of an adequate schema but also on the design of an SLA Management Framework as layer for dealing with all these issues.

3.4. SLA Frameworks

Several research activities have targeted the field of Service Level Agreements already. Most of them have also provided (at least partial) reference implementations. The following sub chapters will give an overview of the most prominent ones.

3.4.1. The NextGRID SLA Framework

The focus of NextGRID²¹ was on the development of architectural components which should lead to the Next Generation of Grids. Within the scope of the project an SLA Framework architecture was developed and a set of reference implementations were implemented, based on Globus Toolkit 4 and GRIA. Thereby the developments aimed to cover the whole SLA Lifecycle as defined by the TeleManagement Forum. SLAs in NextGRID are based on a project-own SLA Schema [52], which consists of *Name, Context, Terms and Signatory* elements. Generally the schema targets a wide application area and by that contains a lot of generic elements and is not defined fine

²¹NextGRID: Architecture for Next Generation Grids, <http://www.nextgrid.org>

enough for a specific domain (e.g. not the HPC domain). Available reference implementations²² support mainly the tasks of discovering service providers, negotiation of Service Level Agreements and provisioning of the service. With respect to the negotiation, NextGRID SLA Management follows the Discrete Offer protocol. [53]

3.4.2. The Akogrimo SLA Framework

The Akogrimo²³ project was launched to provide a middleware meeting the needs of fixed, nomadic and mobile citizens. Therefore it took results from the domains of Grid and network and merged them together in a powerful framework.

The Akogrimo SLA Framework was designed to support the enactment of Service Level Agreements in this highly flexible environment on both, Grid and network layer. By that, the terms of SLAs cover also network QoS parameter.

Generally Akogrimo did not work on a SLA schema and based its work on WS-Agreement. The use thereby was not HPC or service specific. WS-Agreement provides the basis for the SLA definition in Akogrimo, the component implementations were developed using the WSRF.NET platform (details can be found at [54]).

3.4.3. The BREIN SLA Framework

SLA Management in BREIN²⁴ was developed on basis of previous research results e.g. from NextGRID, Akogrimo or TrustCoM. With this starting point, the work in the project focused on the enhancements of the already available components and aimed to address the known gaps with respect to SLA handling.

By integration of Semantics and Multi-agent concepts in today's Grid technologies, the BREIN SLA Management has inherited also extensions from this field. On one hand, SLA negotiation in BREIN can now be performed optionally with Agents on the other hand, the concept of semantically annotating Service Level Agreements (so called SA-SLAs [55], [56]) allows for an enhanced definition of the SLA Terms. The schema itself is used for addressing the HPC domain within one of the two scenarios (dealing with ANSYS applications [57]). More on this can be found in the BREIN architecture [44] document.

Implementations of the BREIN SLA Framework components have been done with Globus Toolkit 4 and the .NET framework.

²²NextGRID: Components Availability, http://www.nextgrid.org/appendix_d/appendix_d.html

²³Akogrimo - Access to Knowledge through Grid in a mobile World, <http://www.akogrimo.org>

²⁴BREIN - Business objective driven reliable and intelligent Grids for real business, <http://www.gridsforbusiness.eu>

3.4.4. The BEinGRID SLA Framework

The Business Experiments in Grid project (BEinGRID²⁵) targets to create a Grid toolset repository with components based on the main Grid software distribution such as Globus Toolkit, GRIA or gLite. To validate the results, 25 Business Experiments were performed, using the components in a realistic scenario.

With respect to Service Level Agreements, an SLA Management architecture [58] was developed, addressing the realization of the SLA lifecycle. With the nature of the project, the SLA Management solution of BEinGRID provides a consolidated version of concepts of SLA related components as developed in previous projects. Software components for *SLA Negotiation, Monitoring & Evaluation, Resource Optimisation and Accounting* have been implemented on top of Globus Toolkit 4 and tested in the Business Experiments. They can be downloaded from IT-Tude²⁶, the BEinGRID knowledge repository.

3.4.5. SLA@SOI

The SLA@SOI²⁷ project aims to adopt a holistic SLA management approach which allows for a business ready infrastructure based on the principles of predictability and dependability. Thereby the whole process of SLA negotiation, provisioning, delivery and monitoring shall be automated.

SLA@SOI started in 2008 and has at the time of writing finished its first year. So far there is not much information about first results available which should change over time. In general this seems to be a good candidate to cover some of the requirements of the realization of SLA Management as presented later in this thesis.

3.4.6. Summary

In the previous sections five SLA Management Frameworks were presented, which approach the SLA Management from different perspectives as can be seen summarized in Table 3.3. For Instance NextGRID provided an SLA Management Framework which on a conceptional layer addressed all phases of the SLA Lifecycle and developed an own SLA Schema. For negotiation NextGRID used the Discrete Offer protocol.

Looking at the presented frameworks, two things are obvious: (i) Many activities

²⁵BEinGRID - Business Experiments in Grid, <http://www.beingrid.eu>

²⁶IT-Tude, <http://www.it-tude.eu>

²⁷SLA@SOI, <http://www.sla-at-soi.eu/>

Framework	SLA Representation	Neg. Protocol	HPC
NextGRID	Own Schema	Discrete Offer	No
Akogrimo	WSAG	Discrete Offer	No
BREIN	WSAG/WSLA	DO/Multi-round	Yes (AN-SYS)
BEinGRID	No specific	Hybrid of Multi-phase/DiscreteOffer	Yes
SLA@SOI	WSAG	Not yet specified	No

Table 3.3.: Frameworks addressing SLA Needs

were not exclusively focused on the HPC domain and thereby would need to be adapted, (ii) a variety of negotiation protocols was addressed which implies the need for flexibility with respect to the potentially used protocols.

What kind of protocols may be needed can be found in the next section (Section 3.5). In general, this thesis will take up the needed flexibility within the architecture but also with respect to the schema development.

3.5. Discovery/Negotiation Protocols

As already mentioned in the introduction, the variety of different Negotiation Protocols is high and by that also covers a variety of concepts. This may be acceptable from a conceptual viewpoint, but when it comes to the implementation of such a system usually one of them is chosen, limiting the scope of usability. McKee et al. [59] identified within the context of the NextGRID project the need for careful decision taking of using a simple protocol or more complex negotiation, based on economic parameters (especially on costs).

3.5.1. Discovery

Many approaches on the usage of SLA Templates have been provided so far. Projects like NextGRID mainly focusing on copying the yellow pages approach to the electronic discovery of services. This implies that the template(s) are stored in one (or more) third party registries, which then can be queried by customers for their preferred services. In parallel, all Service Level Agreement Templates can also be stored in a component in the service providers domain (called SLA Template Repository).

3.5.1.1. Universal Description, Discovery and Integration - UDDI

UDDI - Universal Description, Discovery and Integration is an OASIS²⁸ aims to define a universal mechanism for dynamic discovery and invocation for/of Web Services. For that purpose, it provides an XML based registry for its end users. Thereby it contains three main components: *White Pages*, *Yellow Pages* and *Green Pages*.

- **White Pages** works similar to a phone book (white pages) and delivers information about the identity of a service provider. This covers information about the business domain, contact data of the responsible person and an unique ID of the Provider which is taken from the Data Universal Numbering System (DUNS²⁹).
- **Yellow Pages** provides a categorization of the services which can be queried. The classification of services thereby follows international standards such as United Nations Standard Products and Services Code (UNSPSC³⁰)
- **Green Pages** covers the provisioning of technical information about services (exposed by their owners). Amongst others, this covers address of the service and parameters but also on the specification on interfaces.

Once pursued by big industrial players like IBM, Microsoft or SAP, the dropped out of the development of UDDI 2005 which was seen as a stop for further evolutions of the specification. The last version, UDDI Version 3.0.2 [60] was published in October 2004, which supports this assumption.

3.5.1.2. WS-ServiceGroup

This specification is part of WSRF³¹ (Web Services Resource Framework [61]), an OASIS standard, and provides the means to form a wide variety of collections of Web Services or WS-Resource [62].

With this specification the formation of registries of services and their associated WS-Resources is possible, which can then be used for discovery purposes.

The latest version of the specification was released in April, 2006. Implementations

²⁸OASIS - Organization for the Advancement of Structured Information Standards, <http://www.oasis-open.org/home/index.php>

²⁹D & B - Dun & Bradstreet, <http://www.dnb.com/us/>

³⁰United Nations Standard Products and Services Code, <http://www.unspsc.org/>

³¹WSRF - Web Services Resource Framework, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

can be found within the Globus Toolkit 4 framework, especially in the Monitoring and Discovery System (MDS)³².

3.5.1.3. Web Service Inspection Language - WSIL

The WS-Inspection Language specification [63] provides a discovery mechanism for services which follows a de-centralized approach. In opposite to UDDI, it assumes a direct check on the service providers for their offered services. By this it is designed to be complementary³³ to UDDI as it will allow for discovering those services, which are not listed in a UDDI registry.

The specification provides the possibility of aggregation of different types of service descriptions whilst not delivering a service description language itself. A WSIL document can contain more than one reference to a service description, which could also link to WSDL or UDDI documents.

The first version of this specification was released by IBM and Microsoft end of 2001 and updated in 2007.

3.5.1.4. WS-Discovery

Web Services Dynamic Discovery (WS-Discovery [64]) is a technical specification released from the OASIS consortium members BEA Systems, Canon, Intel, Microsoft and WebMethods.

It defines a multicast discovery protocol which allows a client searching for one or more preferred services by sending a probe message to a multicast group. Once a target service (provider) sees that it matches the probe, it sends a response directly to the client(Customer). In addition, the protocol defines multicast suppression behaviour, if a discovery proxy (for unicast) is available to ensure that it scales to a large number of endpoints.

The latest version of the specification was released in May 2009 (Version 1.1). Implementations of it can be, amongst others, found in Windows Vista (e.g. the "People near me" functionality), as well as it will be part of the Windows Communication Framework 4.0 (to be released in 2010).

³²MDS - Monitoring and Discovery System, <http://www.globus.org/toolkit/mds/>

³³Relationship between UDDI and WSIL, <http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.jst.ws.doc.user/concepts/cwsil.html>

3.5.1.5. Invite-Tender

The Invite-Tender protocol puts less burden of action on the Service Customer than i.e. a yellow pages search, at least in terms of finding matching business partners. Here, the Customer itself specifies the requirements for its preferred service and invites offers (tender) from service providers. Having received these offers, the Customer can choose one or more suitable tenders and enter in negotiations with the respective provider(s).

3.5.2. Negotiation

3.5.2.1. WS-Negotiation

In 2004 Hung et. al proposed an independent declarative XML language called WS-Negotiation [65] for service providers and Customers to come to an agreement. WS-Negotiation addresses as protocol bilateral and multi-issue discussions of offers and counter-offers between service provider and Customers (called Requesters in WS-Negotiation). According to Figure 3.5 the WS-Negotiation specification covers

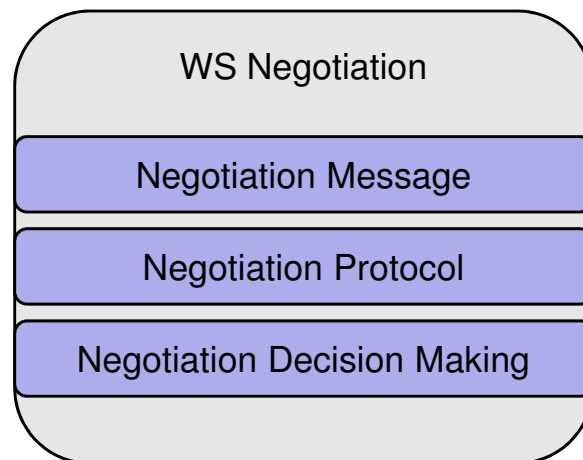


Figure 3.5.: The WS-Negotiation Structure

three main parts:

- Negotiation Message
- Negotiation Protocol
- Negotiation Decision Making

The **Negotiation Message** describes the format of exchanged information between the negotiating parties. Originally it was planned to enhance the Negotiation Message to a grade, where it defines schema and semantics of the messages, but this has not been performed yet. The messages are exchanged following a pre-defined **Negotiation Protocol** which describes the mechanisms and rules for negotiation, essentially it is an exchange of offer, counter-offers, etc. Finally **Negotiation Decision Making** is an internal decision process based on the respective strategy of the users. Thereby the assumption is that the strategies can be manifold and completely different between the negotiation participants.

3.5.2.2. WS-Agreement-Negotiation

WS-AgreementNegotiation [66] which sits on top of WS-Agreement describes the re/negotiation of agreements. The protocol was part of the early specifications of WS-Agreement but then taken out to realize a clear disjunction between the SLA description and the protocol. WS-AgreementNegotiation is constrained to express either acceptance or rejection of an offered SLA, a property which limits its use so far. That motivated the Open Grid Forum's GRAAP group and other researchers to work on extensions of the protocol, however there was no mature release of the specification until now.

The current version which is still very immature follows the Discrete Offer Protocol (see also Chapter 3.5.2.3. A new version, targeting multiphase negotiation, is currently under refinement within the group.

3.5.2.3. Discrete Offer

The Discrete Offer Protocol [53] is a very limited protocol which follows the so-called "Take it or leave it" approach. The approach provides an inflexible protocol, assuming the availability of a set of pre-defined service descriptions, from which Customers can choose. By that no degree of freedom is given to change any of the service properties, which at the same time minimizes the efforts needed (and by that the costs) to come to a final agreement (or to quickly identify that there will be no such agreement). In the NextGRID project [10] this approach was chosen for a reference implementation of SLA Negotiation in simple business cases. But already within the NextGRID project it became clear that for complex scenarios a simple "take it or leave it" approach is not sufficient any more to cover the requirements. This protocol is very good in domains where fixed services with fixed prices are sold (remember a Telecom Provider selling a phone card for mobiles for 25 Euro), as soon as flexibility is needed, this protocol may be not sufficient to address the needs of business entities.

3.5.2.4. Multiphase (n-phase) negotiation

In opposite to the discrete offer protocol the multiphase (n-phase) protocol assumes that there could be several rounds of negotiation. The n hereby stands for the number of rounds. A Customer asking for a service with preferred capabilities gets an offer from one or many service providers with different term values (parameters) and is in the following able to change them again as new proposal to fit its needs. Theoretically this could last for an infinite duration, which is the biggest concern of opponents to this approach. On the other side, it gives both business entities the chance to get to their best solution (compare this with an bazaar). Of course the implication of this model is that if the service provider decides to reserve resources for the Customer, he runs into danger of blocking his resources until the end of negotiation. But this is a decision that is up to the Provider himself (if he takes the risk to block resources or if he takes the risk to offer resources without blocking them) and depending on his business model. Intentionally this section here is kept short, in-depth details will follow in the respective chapters.

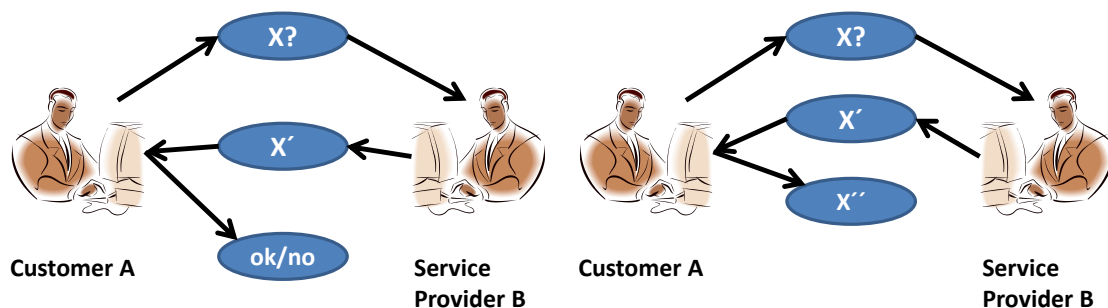


Figure 3.6.: Discrete Offer and Multiphase Negotiation

3.5.2.5. Service Negotiation and Acquisition Protocol - SNAP

In 2002 Czajkowski et al. [67] presented their work on a Service Negotiation and Acquisition Protocol, called SNAP. This protocol aims to allow remote management of Service Level Agreements beyond the borders of different resource providers. Thereby the authors of this approach concentrate on how resource reservation and provisioning can be enabled when negotiating a single SLA across multiple domains, referring to multiple resources.

To avoid cross-domain problems, SNAP introduces three types of SLAs:

- TSLAs - Tasks Service Level Agreements, which covers the performance of an activity or a task. It contains information about service steps and resource requirements
- RSLAs - Resource Service Level Agreements, which covers the right to consume a resource but not necessarily what the resource will be used for. It contains a characterization of the resource by its abstract service capabilities
- BSLAs - Binding Service Level Agreements, which covers the application of a resource to a task. This task is either defined by the TSLA or some other identifier which then can be applied to a RSLA (and vice versa)

Even though already published in 2002, the SNAP work has not found the desired uptake in the developers community. Three years after its publication described Haji et al. in 2005 [68] the implementation of a resource broker on basis of SNAP which uses a three phase commit protocol.

At the same time this is also an example, that the original SNAP idea has several weaknesses and would need investment of effort to enable its usability. One reason for that is its generality which at the same time implies the missing of concrete parts (this was already discovered by Sahai et al. [69]).

3.5.3. Auctions

In contrast to the above mentioned protocols for negotiation, auctions do not target changes of service parameter, but focus on the pricing with respect to a delivered service. In the following, a subset of different auction mechanisms will be shortly presented. Auctions in the Service Level Agreement domain become mainly of interest when the price is to be optimized but, the QoS parameters have to stay on fixed values.

3.5.3.1. English Auction

The English Auction [70] is a method which allows for multiple stage price-finding. There are two different approaches towards the realization of the English Auction, which both assume a start offer from a potential Customer being made public and in the aftermath an increase of the price with each new offer by

- a constant value
- a chosen value by the Customer

until a final offer has been made, a buyer has been found and the auction can be closed. The English Auction was amongst others provided as specification [71] by the Foundation for Intelligent Physical Agents - FIPA³⁴. It can be used for buyers who have a price limit, but not necessarily want to announce this in advance.

3.5.3.2. Dutch Auction

The Dutch Auction [72] is a mechanism, where the entity selling a service starts with a high price and lowers it during the auction, until a customer is willing to pay. This is an opposite bid order to the English Auction and usually performed until a limit has been reached. This limit is usually set by the seller as minimum acceptable price.

3.5.3.3. Vickrey Auction

The Canadian professor William Vickrey proposed an adapted type of auction - the Vickrey Auction [73] - where, similar to the English Auction, the bidder of the highest price wins. However, the big difference is that the buyer has not to pay his highest price, but the second highest (of the excelled last bid). As the Vickrey Auction is using the concept of hidden offers, there is a danger of abuse, either by the seller (as nobody really can tell, what the second-highest offer was), as well as of other bidders (who can make offers higher than their limits to increase the price). Vickrey's approach is not commonly used by offerors.

3.5.4. Summary

In opposite to the previous sections of this Chapter, this section is not intended to justify a pre-selection of candidates for the realizations presented in this thesis. It should rather show the variety of approaches towards discovery and negotiation of services and Service Level Agreements, which, theoretically should all be addressed by an SLA Management Framework for the HPC Domain.

All listed approaches should be applicable in case of certain conditions. E.g. quick discovery and negotiation protocols are needed for the Urgent Computing Scenario (as presented in Section 2.1.1). This could for instance be solved by applying the Discrete Offer protocol with an UDDI search.

This flexibility is new to all solutions provided so far and will be addressed on the SLA Management Framework and Schema level.

³⁴FIPA - The Foundation for Intelligent Physical Agents, <http://www.fipa.org/>

Chapter 4.

The Architecture of the SLA Management System

The aim of this chapter is to go step by step from the detailed definition of the Service Level Agreement lifecycle, as it is defined by the author in chapter 1.3, over a high level architecture sketch up to a detailed design for the different parts of the framework.

This chapter sets the scene with some clarifications about terminology and the basic concepts of this thesis in order to then detail the different phases of the lifecycle with use cases. This will lead in the following chapters to a breakdown on component level for those parts of the SLA Management Framework which are involved in the respective phases. The designs will address the requirements as laid out in chapter 2 and provide the necessary components to process the SLA schema as it will be presented in chapter 5.

Finally, a short prediction on future SLA establishment is given.

4.1. Terminology and concepts

In this thesis, there will be a distinction made in the SLA based type of documents which are handled during the SLA lifecycle. This sub chapter will introduce the concepts behind those documents to allow for an easier access to the following sections.

Figure 4.1 shows the different states that a Service Level Agreement document can have. The single states are described in further detail in this chapter.

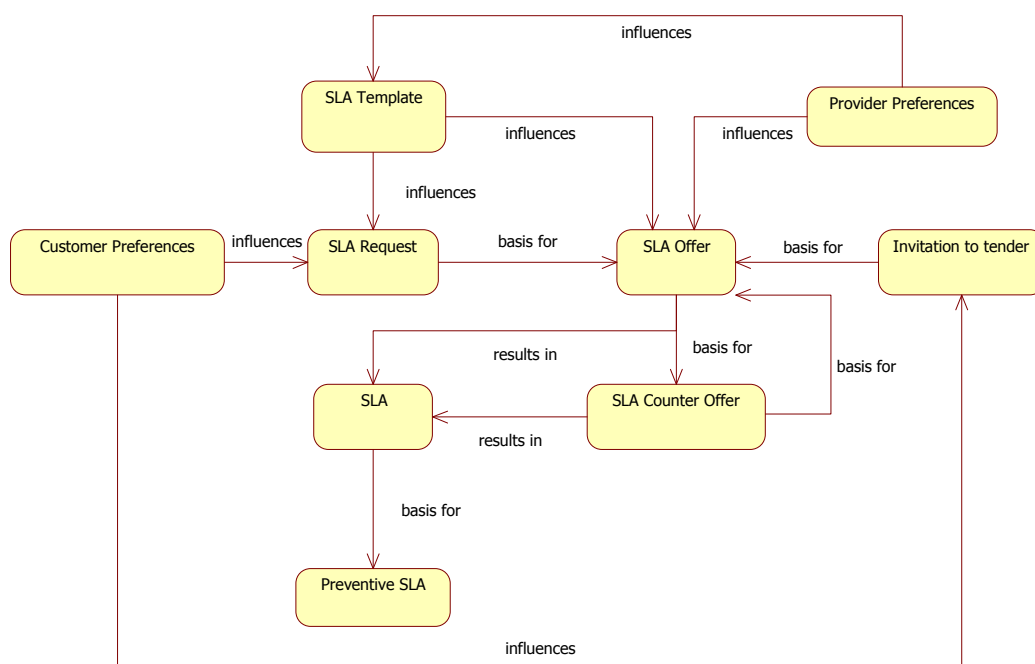


Figure 4.1.: The different States of the SLA related Documents

4.1.1. SLA Template

A *Service Level Agreement Template* contains a service description which covers functional and non-functional parameters and can also link to other, more concrete service descriptions. It is created by service providers based on the service they want to sell and used as an instrument to give potential customers a first impression of the capabilities by publishing them.

Generally not all SLA Templates are published for outsiders as, depending on the business policies of the service provider, there will be SLA Templates for certain customer groups. In general it is worth mentioning here that the number of SLA Templates that are created and stored in the service providers repository can differ from the number of those published. As in real life, the business entity providing a service may decide to offer special variations of a service (e.g. special parameters, price) only to a group of **special** customers.

4.1.2. (Invitation to) Tender

This document is created by a customer that needs a service to be executed. If the service is not urgently needed, the customer may use this concept to publish its preferences to service providers (e.g. on a Trusted Third Party TTP) Service). This is the inverse process of the Template publication of the service provider but allows the customer to receive tailored SLA Offers on the basis of his tender.

4.1.3. SLA Request

An *SLA Request* document is created to react on received templates during the discovery of service provider. If one or more templates match the needs of the customer, the candidate providers can be contacted directly with a SLA Request. The SLA Request is based on the received SLA Template(s) and has in the best case the same structure as the Template (not necessarily a must).

This document is sent to the service provider to initialize the negotiation procedure. In some literature the SLA Request is also titled *SLA Bid*.

4.1.4. SLA Offer

When receiving a request, service providers create SLA Offers based on the incoming documents as well as on their own SLA Templates and system information. This *SLA Offer* is then sent back to the requester.

In case of an invitation to tender, the service provider creates an Offer on base of the tender document.

4.1.5. SLA Counter-Offer

In most cases, even the best matching SLA Offer of the service provider does not fully match the customer's wishes by hundred per cent. Therefore the offer has to be adapted. This modified SLA Offer is sent to the service provider as a *SLA Counter Offer*.

4.1.6. SLA Agreement

The *SLA Agreement*, usually only called SLA (Service Level Agreement) is the document which is available at the end of the negotiations between customer and service provider. This document contains all the necessary information on the levels of the quality of service that both entities agreed upon in addition to environment terms, which are defined in detail in chapter 5. The SLA status is reached, when there is a positive acknowledgement of the current state of the document and it is finally signed.

4.1.7. Preventive SLA

Preventive SLA is the label for an adapted version of the original SLA. For the purpose of preventive mechanisms, this document is derived from the SLA and contains threshold values of critical parameters which, when reached, indicate an upcoming violation but still leave space for modifications on the system to avoid them.

4.2. Six phases in the life of an SLA

A first overview of the SLA lifecycle phases was already given in Figure 1.3. To get a better understanding of the underpinning concepts of this thesis, this sub-chapter will detail all the different phases.

4.2.1. Development of the Service, the associated Templates and Publication

The development phase in the SLA lifecycle is initiated either by the customer or the service provider.

4.2.1.1. Customer driven Development

In contrast to traditional approaches towards the development phase in a service lifecycle (as e.g. addressed in projects like NextGRID or or BEInGRID), this thesis takes the assumption that development is not only done by the service provider but also by the customer in terms of preparing/designing an (invitation to) tender.

This tender represents the properties of a desired service from the viewpoint of the customer and gives the service providers a hint of what they should offer (if they are interested in winning this customer).

If looking for a service, a customer *creates a Tender* based on his requirements and *stores this Tender* in their own repository. The Tender creation is thereby based on the services properties but also on the business objectives/policies of the customer. To make the Tender accessible for outsiders (service providers), it is stored in an external repository, often called blackboard.

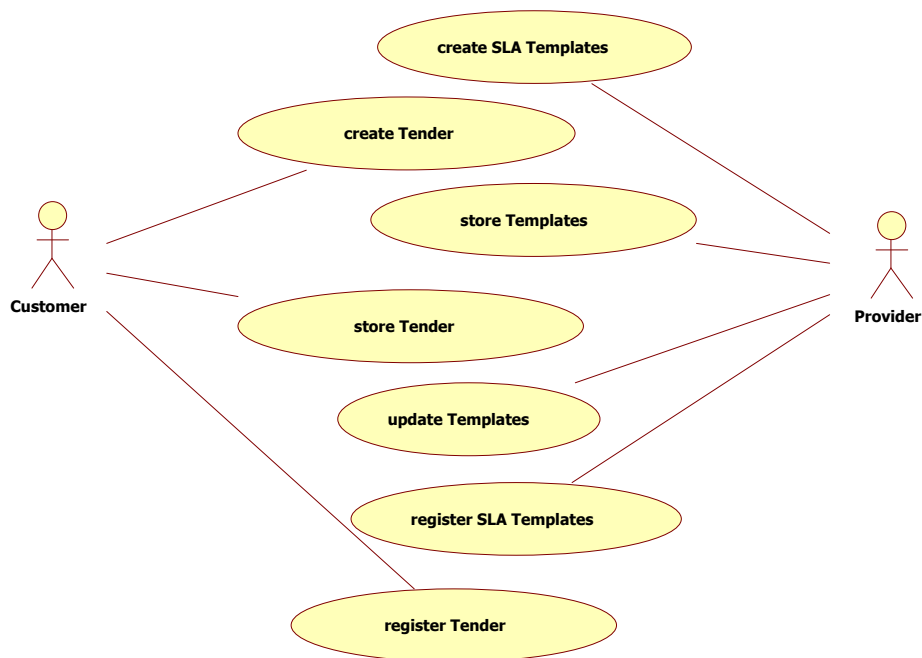


Figure 4.2.: Actions of Customers and Service Providers in the Development Phase

4.2.1.2. Service Provider driven Development

To be able to offer services electronically, the service provider needs to represent these services in a format which can be processed adequately - Service Level Agreement Templates (cf. Chapter 4.1). For that purpose, the provider *creates SLA Templates* based on his service capabilities and *stores the Templates* in his own repository.

However, giving the impression that this would happen only once when a new service is designed is wrong as the evolving needs of customers, the competition in the markets and the behaviour of the initially designed templates often require a change of the offered capabilities and/or parameters. Information about the behaviour of the service (the SLA) is thereby retrieved from the evaluation of the performance with respect to internal metrics, carried out by the assessment block.

The decision on the content of SLA Templates is influenced by several parameters. Of utmost importance is a thorough analysis of the market needs (by that the needs of potential customers), as this is the basis for valuable and sell-able services. Then, the limiting factor has to be taken into account which consists mainly of (a) the own resources capabilities and (b) the business policies.

A common approach to ensure that potential customers can locate matching services (and their providers) is the usage of so-called trusted third party registries. To advertise their capabilities, providers can *register* here their service descriptions which can then be retrieved by search/discovery mechanisms on behalf of customers. At the end of this phase SLA Templates or Tender have been created and are ready to be used for discovery and negotiation purposes.

4.2.2. Creation - Discovery and Negotiation of an SLA

The second phase of the Service Level Agreement lifecycle covers the creation of an Agreement, starting from the discovery of matching business partners to the negotiation about detailed terms of the SLA and the signing of the binding document at the end.

4.2.2.1. Discovery

Every business entity looking for a service fulfilling its demands, needs the possibility to start an adequate search process to *discover* potential service providers. This search may be performed taking into account the conceivability of this service. By defining this request in an electronically processable format, it can be used to feed discovery services for searching for matching service providers. Several projects including NextGRID and BREIN have already published concepts for this [74]. There are different approaches on how customers and service providers can be linked together based on discovery mechanisms: SLA Templates in the simplest case can be used not only as a basis for discovery and later negotiation of Service Level Agreements, but also support the automation of SLA creation on the side of the provider. Hasselmeyer et al. [53] presented such an approach (called the "yellow pages approach") which was implemented during the NextGRID [10] project. Even though

this is a common approach, it is not the *casus sui generis*. Several different approaches exist and are presented in further detail in the state of the art overview in chapter 3.

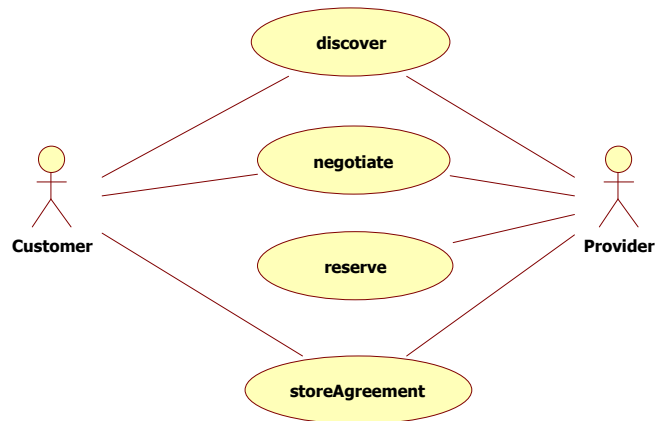


Figure 4.3.: Actions of Customers and Providers in the Creation Phase

4.2.2.2. Negotiation

The *negotiation* of a Service Level Agreement and which protocol it should follow is a heavily discussed topic. There are many defenders of the thesis that there is no negotiation needed as it is senseless (e.g. as described by Mitchell and McKee [75]). They proclaim that if they sell a fixed service for a fixed price, no negotiation is needed (e.g. the sale of a 25 Euro mobile phone prepaid card). However, looking at this example, we can map "no negotiation" to the "take it or leave it" approach which is a simple protocol allowing no changes (during negotiation) on the offers from a service provider, but it is still negotiation.

First approaches were taken to specify a protocol for SLA Negotiation (e.g. WS-Agreement), which exactly realizes this "take it or leave it" approach, but were never released in a mature state and accepted as a complete specification. One reason for that is the on-going argument about the flexibility on term negotiation of such a protocol which leads to the need for complex mechanisms on both sides, customer and service provider, when automated.

However, even looking only at the use-cases of this thesis, it becomes obvious that

there are other scenarios which may require a multi-phase negotiation, e.g. could the reliability level in the visualization use case be proportional to the price for the service or the price is customer dependent.

Independent of the message flow between the customer and service provider, there are more activities which have to be performed during the negotiation of a Service Level Agreement, namely *reservation*, *storage* and *signing*. Reservation should be foreseen as an action for the service provider, in case he wants to reserve the resources necessary for later execution of the service. This is not a must as this depends on the service providers' business objectives. There are cases (e.g. overbooking of flights) where this does not happen (even though the customer assumes it to be done), then the service provider has to handle this internally.

When a negotiation is successfully finished (meaning that both parties have taken the decision to establish this bipartite relationship with the contents of the SLA as basis), the SLA document has to be signed by both to become valid. When signed, the SLA is stored in a SLA Repository Instance, existing usually in both domains.

Within the scope of this thesis, the assumption is taken that negotiation is performed between customer and service provider and no external support is needed/provided. Other approaches are based on the concept of involving a (trusted) third party in the negotiation activities. With this involvement negotiation functionalities could be outsourced to a so called "Negotiation Broker", which acts on behalf of either customer or service provider [45].

Additionally, a third party could in the execution phase of the SLA act as a 'notary' component [76], which may own a copy of the Service Level Agreement for reference. By that it acts as a neutral party which can handle and decide on conflicts if there exist differences in the two SLAs of the business partners. In this case, the SLA has also to be stored in the notary's repository.

Alternatively the notary can be involved in the signing procedure so that at the end an encoded SLA is created which can be read, yet not altered. In this case, the Notary either "witnesses" the signing process, or performs it itself with keys provided by customer and service provider [77]. At the end of this phase, a Service Level Agreement is available and, if chosen to do so, a reservation of resources has been already performed.

4.2.3. Service Provisioning and Deployment

In this phase of the SLA lifecycle, all actions have to be taken to ensure the availability of the agreed service with respect to the defined parameters in the SLA. This is an action that is mainly performed on the side of the service provider, but depending on the Service Level Agreement there can also be obligations on the customer which

require a *configuration* on his side (e.g. provisioning of licenses). Such configuration can take place for the system devices, including additional reservation mechanisms, but also of tools/mechanisms, allowing to control the execution of the service. Control in this context means to carefully watch the status of the service as well as to evaluate its past and current behaviour with respect to the SLA.

There are cases where not only the service provider monitors the service/system performance but also the customer wants to receive status reports (probable in combination with certain data sets). To this end, this data has to be aligned with the SLA terms to avoid any confidentiality problems (a service provider normally does not want a customer to see the plain system level data and the customer is often not interested in getting this data).

Alternatively also components may be prepared which allow for so-called preventive monitoring. Similarly to the "normal" monitoring, they are configured with respect to an SLA. This time, it is a preventive SLA which is deduced from the SLA between customer and service provider but contains adapted parameters which, if reached or passed, give a clear hint of upcoming violations. This is something which is of help for service providers to control their system and service execution and allow for counter actions before a violation has occurred.

All these configurations are based on the respective Service Level Agreement contents. At the end of this phase the service is ready for execution by its user.

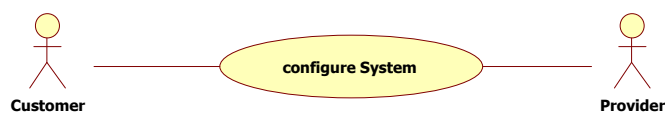


Figure 4.4.: Actions of Customers and Providers in the Provisioning and Deployment Phase

4.2.4. Execution of the Service

After the provisioning and deployment phase, the real execution of the service can start. This is a phase of vital importance for the success of a Service Level Agreement. Now both, service provider and customers, have to ensure that their obligations (as stated in the SLA) are fulfilled.

This can either happen manually or automatically, depending on the available infrastructure. This phase lasts as long as the validity of the SLA was agreed during

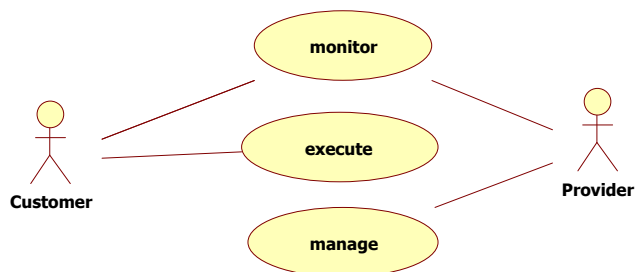


Figure 4.5.: Actions of Customers and Providers in the Execution Phase

the negotiation. The main activities in this phase are the management, the execution and the monitoring of the service, all based on the SLA terms. Customers can now use/execute the services as they wanted to, and the service provider manages it to meet his obligations.

During the execution of the service, monitoring, based on the SLAs, is performed by at least the service provider. The monitoring data is then evaluated in the assessment activity. Usually the plain system data does not exactly match one to one the SLA terms. This means that a dedicated component is needed which uses the data from the system/service data sources to format them to the SLA terminology. It may occur that the customer also wants to monitor the data. In this case, it should be explicitly agreed during negotiation which data is made available under which circumstances.

4.2.5. Assessment and corrective actions during execution

This phase is running partially in parallel to the execution phase, during which it is of high importance for the service provider to maintain control over what is happening inside his domain whilst the customer also wants to know what the status of his service is. Especially when the service is part of a workflow and by that strong dependencies exist, customers need to be able to gain knowledge about the status of the service, in the best case on demand. This implies that parallel to the execution of the service mechanisms are needed giving support for both business entities in terms of performance assessment.

Three main activities are performed during the assessment phase: *evaluation* of the

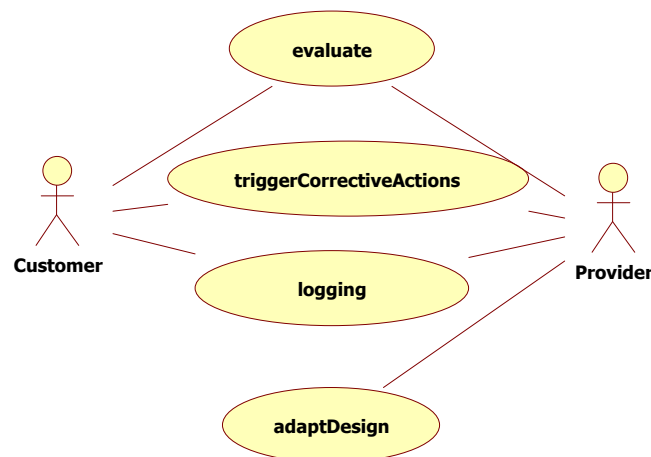


Figure 4.6.: Actions of Customers and Providers in the Assessment Phase

data monitored during execution, *logging* of the performance and the *triggering* of *corrective Actions*.

Using monitoring mechanisms ensures that SLA related data can be made available for evaluation. Monitoring itself only provides the data, the evaluation is then performed separately and is the process where the monitored data gets reviewed with respect to the Service Level Agreement and the results get rated. In theory, external Evaluation (from the customer) is equally possible as internal evaluation (by the service provider). All this depends on the implementations on the side of the service provider and its general policies of either allowing external evaluation or not. As mentioned in the description of the execution phase, evaluation inside the service provider domain can be twofold: either as a regular mechanism based on the SLA or as preventative evaluation with an adapted (preventive) SLA.

Both parties may want to log the performance of the service, especially for later accounting and billing purposes. So the means to log the information gained should be given for service providers as well as for customers.

Based on the information gained by the evaluation, *corrective actions* are possible or necessary: This could be a reconfiguration of the service providers system, especially when it was preventively foreseen, or in case of an occurred violation, but due to only a temporarily cause, which can be fixed by the provider. Then only the charge of a penalty (according to the SLA) could satisfy the customer. On the other hand, if the problem is not that easy to solve, the parties could try to agree on adapted parameters, which would be still acceptable for both. This would lead to a re-negotiation, which is a new negotiation.

In the worst case, the breakdown of the complete SLA won't be avoidable and or other parameters are not acceptable for the customer. Then the customer may decide to immediately stop the execution and to replace the service by one of another service provider. This is enacting the termination phase.

Based on the evaluation data and by that having a clear overview of the performance of the offered service (which was initially sold based on an SLA Template), the service provider can use this logged data to enact again the design process for adapting the SLA Templates to provide higher quality.

4.2.6. Termination and Decommission of the Service

The last phase of the Service Level Agreement lifecycle covers the termination and decommission of the service. Termination could happen at least on three conditions:

- The service ends "normally", which implies, that the service provider has fulfilled the SLA and delivered the required quality of service (as well as the customer has fulfilled certain obligations, when they are stated in the SLA). Usually, an SLA has a validity duration, which could be also an exit condition, when no explicit target outcome of the service was defined.
- The service is intentionally stopped by the service provider or the customer
- The service ends during its execution due to a violation of the SLA when reaching a terminating condition.

In all cases, a wrap-up has to take place that activates final accounting and billing mechanisms, checking what each party has to pay/charge and executing the payment processes. Additionally, all blocked resources for the service can be released (*dissolved*) and used otherwise.

Even though this seems to be the end of the service usage, the *termination* of the service (usage) and *decommission* does NOT implicitly mean to delete the Service Level Agreements. SLAs usually have to be stored for a certain timespan to provide a basis for future discussions, in case of disagreements for statistical purposes and for auditing. Based on the detailed description of the SLA lifecycle phases, the activity flow of the different phases can be sketched as seen in Figure 4.8. In this picture the circular impact of the assessment results on the development phase is explicitly visible. What becomes obvious when looking at the flow of the different phases is that dependencies between those building blocks are quite different. E.g. development does not necessarily need any input from the other blocks (which is sound as it is the starting point) and execution could theoretically continue without assessment (as this runs in parallel). Of course, this activity is of high importance for meeting the

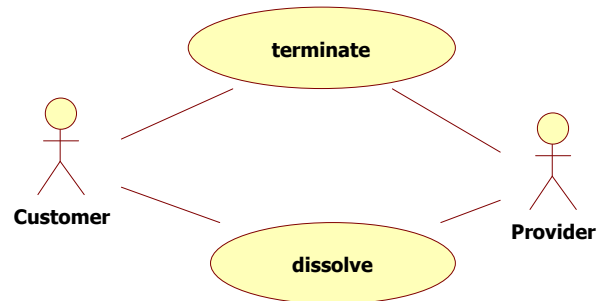


Figure 4.7.: Actions of Customers and Providers within the Termination Phase

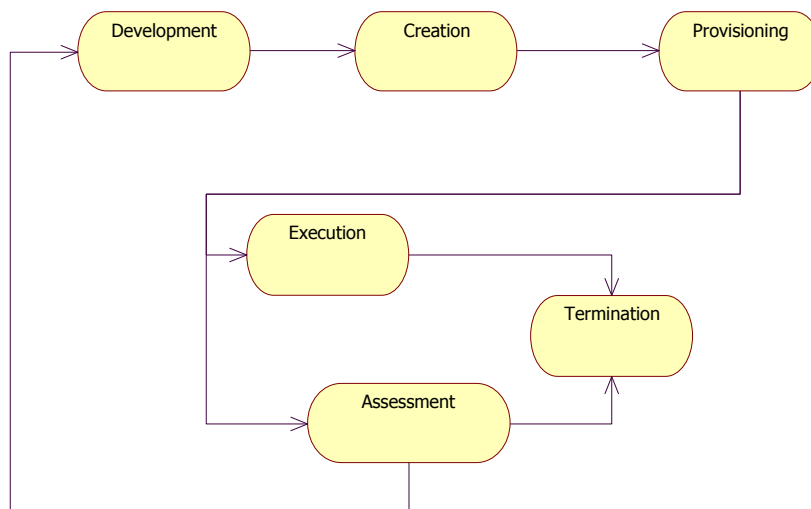


Figure 4.8.: Dependencies of the SLA Lifecycle Phases

expectations on this bipartite relationship. Additionally, the state of the SLA-related documents (as processed by the different blocks) differs as well, from a Template resulting from the development to a complete agreement after creation. Details of how such documents may look like is provided in the Schema overview in the next chapter.

4.3. The high level architecture

This sub-chapter focuses on the provisioning of the SLA Management architecture. Based on the concepts described in the previous chapters, components within the different involved entities domains are designed and presented.

4.3.1. Development

The starting point of the SLA lifecycle as derived for this thesis is the *development phase* (cf. Chapter 4.2.1). Looking at the activities which have to be performed during this



Figure 4.9.: The SLA Lifecycle - Focus on Development

phase and having the fore presented use-cases in mind, the following requirements for the development architecture are:

1. Service providers need to be able to generate SLA Templates for stating their services capabilities

entity domains: customer, service provider and trusted third party. The trusted third party components thereby take the role of offering publication capacities for (a) SLA Templates or (b) Tender.

Service Provider SLA Template Generation

In the first case, a service provider wants to offer a new service with certain capabilities and under certain conditions. For that purpose the *SLA Manager* is contacted (*triggerGeneration*) with the necessary input information. This component then request the *SLA Template Generator* to generate the according templates, taking into account the *Infrastructure Capabilities (check)* and the *Business Objectives (retrieve)* from the respective components.

All created SLA Templates are stored in the provider internal *SLA Template Repository* and can be registered in the third party *Registry* for publication.

An alternative case to the creation of new ones is the update of already created SLA Templates. This might be necessary in case that the resources or their configuration on side of the service provider have changed or if offered services (templates) are too often violated. To request the *SLA Template Generator* to update the Templates, the *SLA Manager* offers a function (*triggerTemplateUpdate*) to start this process.

Customer Tender Generation

The components on the side of the customer support the creation of Tenders. When a customer wants to publish his requirements to allow service providers to propose tailored offers, the customer *SLA Manager* is contacted (*triggerGeneration*) with the necessary input information. The *SLA Manager* then hands over this information (*generate*) to the *Tender Generator* which retrieves the business policies from the *Business Objectives* component to create the best fitting Tender to the customer's needs. Once created the Tender is stored in the customer internal *Tender Repository* as well as on the Trusted Third Party *Blackboard*.

Table 4.1 gives a short overview of all the involved components.

Component	Domain	Description
SLA Manager	C	The SLA Manager on the customer side is responsible for triggering the generation of Tenders. This is done on behalf of the customer and with defined data sets (requirements) provided by either another managing component or a human user.

Component	Domain	Description
Tender Generator	C	This component offers the functionality to generate Tenders based on the received inputs and the business objectives of the customer. When generated, the Tender is stored in a TTP blackboard and in the internal Tender Repository.
Business Objectives	C	This component owns a database with information about the Business Policies of the customer. This covers issues like "As cheap as possible".
Tender Repository	C	The Tender Repository is a customer internal component which contains all created (and valid) Tenders.
Blackboard	TTP	The Blackboard is a storage component for Customer Tenders. Service providers can discover here what customers are looking for.
Registry	TTP	The Registry is a storage component for service provider Templates. By that, customers can query it to get information about the offered services.
SLA Manager	SP	The SLA Manager on the service provider side is responsible for triggering the generation of SLA Templates. This is done on behalf of the service provider and with defined data sets (capabilities) provided by either another managing component or a human user. In addition the SLA Manager offers an interface to trigger the update of SLA Templates in case it has been observed that services based on them show misbehaviour.
SLA Template Repository	SP	The SLA Template Repository is a provider internal component which contains all created (and valid) Templates for the offered service.
Business Objectives	SP	This component owns a database with information about the Business Policies of the service provider. This covers issues like "Services are offered with high reliability and high price or less reliability and cheaper price".
Infrastructure Capabilities	SP	Infrastructure Capabilities provides the necessary information about the resources, their configuration and general properties.

Component	Domain	Description
SLA Template Generator	SP	The Template Generator provides the functionality to generate SLA Templates based on the retrieved service descriptions. Therefore it takes also into account the business objectives and the infrastructure capabilities. Additionally this component is able to update SLA Templates based on retrieved evaluation information of services based on them during execution.

Table 4.1.: Components involved in the Development Phase

With this design and these observations, it becomes obvious that the Development phase as the first phase in the overall SLA lifecycle can be triggered by three different entities:

- An SLA managing component acting on behalf of the customer who intends to publish an invitation to Tender. A component called *Tender Generator* therefore takes the properties which were communicated from the SLA Manager and aligns them with the *Business Objectives* of the customer. For the purpose of publication, this Tender is then stored in one or more Trusted Third Party Blackboards.
- An SLA managing component acting on behalf of the service provider who intends to publish his service capabilities. A component called *SLA Template Generator* therefore takes the properties which were communicated from the SLA Manager and aligns them with the *Business Objectives* and the *Service Capabilities* of the provider. For the purpose of publication, this SLA Template is then stored in one or more Trusted Third Party Registries.
- A component active in the Assessment activity which triggers an update or renewal of already designed SLA Templates. This might be necessary when SLAs based on these templates are often violated or the service provider has updated its system/resources.

4.3.2. Creation

This sub-chapter covers the Discovery and Negotiation Phase. As stated in the previous chapter, this thesis focuses mainly on the creation phase of the SLA lifecycle

(cf. Figure 4.11). Therefore this section provides a detailed design for these activities, based on the requirements which were worked out and the findings in section 4.4.



Figure 4.11.: The SLA Lifecycle - Focus on Creation

4.3.2.1. The Discovery Architecture

The requirements for the discovery architecture are as follows:

1. Service providers must be discoverable based on their capabilities
2. Support of protocols in which the customer is actively searching for service providers (e.g. Yellow Pages)
3. Support of protocols in which the customer is passively announcing his needs, waiting for service providers discovering him (e.g. blackboard announcements)
4. Flexible adaptation to the needs of customers and providers

Based on these requirements, the (symmetrical) architecture in Figure 4.12 was depicted. The Discovery architecture owns customer-side and service provider-side components, as well as two components situated in the Trusted Third Party domain. To cover the worked-out requirements, the architecture foresees support of (a) customer initiating discovery for service providers or (b) service providers looking for invitations for Tender from potential customers.

In both cases, the *SLA Manager* component of either customer or service provider is triggered with the necessary parameters (*triggerDiscovery*) and enacts the discovery process (*discover*) at the *Discovery* component.

A detailed description of the components is given in Table 4.3.

Component	Domain	Description
Discovery	C/SP	This component is responsible for enacting discovery based on the chosen protocol. Depending on this, it queries either the <i>Register</i> (when acting as a customer Discovery) or queries the Blackboard on behalf of service providers to find matching invitations to Tender.
SLA Manager	C/SP	The <i>SLA Manager</i> is the managing unit on both sides. It triggers the analysis of the parameter once received by its end user (probably through a higher layer business management component) and initiates the discovery process.
Register	TTP	The <i>Register</i> owns all published service descriptions as registered by the different service providers. It can be queried with respect to the service parameters and sends back the best fitting providers for a query.
Blackboard	TTP	To allow for an announcement of the will to use a service combined with the request for a good offer, the <i>Blackboard</i> acts as an adapted register for customers. It can be queried by service providers to find invitations to Tender which match their business field.
Selector	C/SP	This component retrieves the set of query results from the Discovery to either Blackboard or Registry and ranks its contents based on the customer/service provider preferences. By that a pre-selection of candidates for negotiation is enabled.

Table 4.3.: Description of the SLA Discovery Components

4.3.2.2. The Negotiation Architecture

With the scenarios presented in Chapter 2 and analysis of related work of the authors together with others [78], the following requirements for a SLA Negotiation Architecture were retrieved:

1. SLA terms must be negotiable between the business entities
2. The context definition of the request should be taken into account during negotiation

3. SLA Negotiation should take into account the capabilities of the providers system
4. SLA Negotiation should take into account the actual status of the providers system
5. The objectives (on a business level) of both entities should be taken into account

Based on this, the high level architecture as shown in Figure 4.10 (b) can be broken down in a detailed design as presented in Figure 4.13.

The Negotiation architecture consists of customer-side and service provider-side

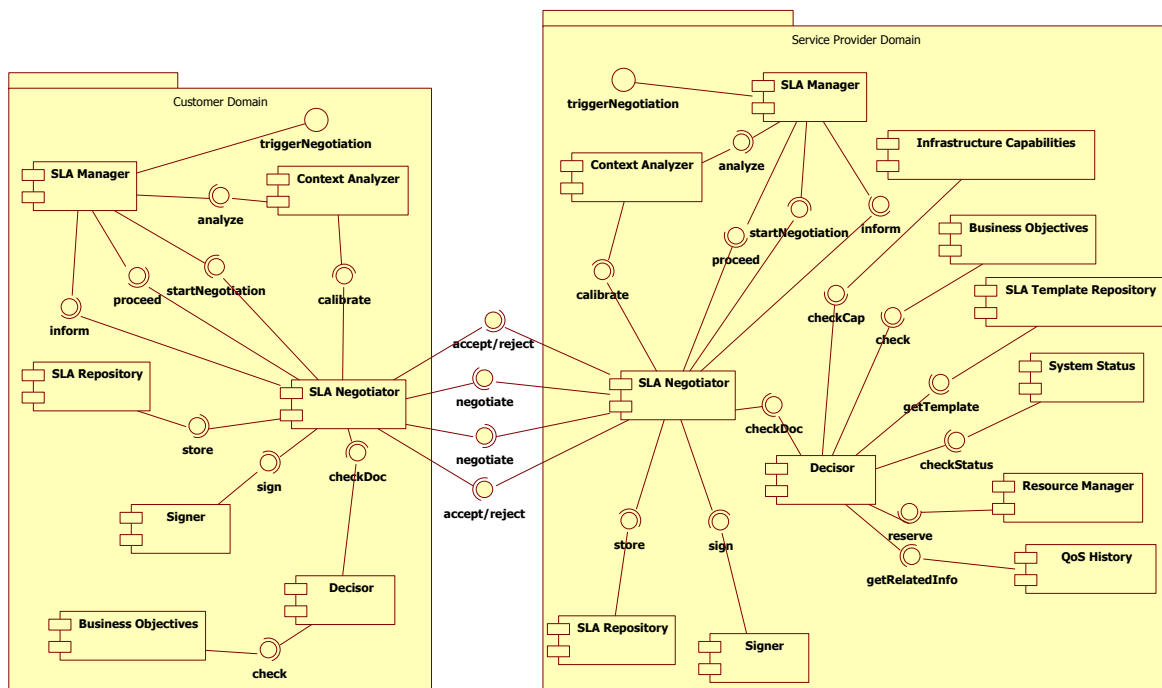


Figure 4.13.: The Negotiation Architecture

components. To address the requirement of negotiability, the communication between both domains takes place via the *SLA Negotiator*, representing the respective party. This component retrieves offers or counter-offers and handles them internally to update the respective electronic document until it is sent back to the other party. As described in Chapter 4.4, the context is an important element for the enhancements provided in this thesis, as it has an implication on the used protocols and by that on the configuration of the *SLA Negotiator*. Therefore the *SLA Manager*, when aiming to kick off the negotiation, requests an analysis of the context (*analyze* from the *Context Analyzer* which then *calibrates* the *SLA Negotiator* accordingly. This functionality is the

same for both domains.

The negotiation partner (either provider or customer) has also to adapt its *SLA Negotiator* to support the needed negotiation protocol. For that purpose, the *SLA Negotiator* informs the *SLA Manager* about an incoming negotiation request who then triggers the *Context Analyzer* to *analyze* the context and to *calibrate* the *SLA Negotiator*. When configured, the *SLA Negotiator* is able to perform the negotiation on behalf of the business entities, which involves the adaptation of requests, offers, counter offers. All based on the business entities desires. Adaptation thereby is performed by a specialized component, called *Decisor*, responsible for taking into account requirements 3,4 and 5 on side of the provider and 5 on side of the customer.

As defined in chapter 4.1, the initial negotiation message is called *SLA Request* when sent from customer to service provider or *SLA Offer*, in case the service provider initially contacts the customer (e.g. in case of an *Offer for a Tender*).

In case of the customer initiating the negotiation, it has received a set of *SLA Templates* during *Discovery*. The customer *SLA Manager* now triggers the *SLA Negotiator* to start the *Negotiation* (*startNegotiation* by handing over the *Template*. For the preparation of the *SLA Request*, the *Negotiator itself* hands this information (*checkDoc*) to the *Decisor* which creates the *Request Document* based on the *Template* and the initial preferences of the customers as used for *discovery*. Additionally, the *Decisor* checks the *Business Objectives* component for the business policies of the customer to optimize the request. The created *SLA Request* is then sent via the customer *SLA Negotiator* to the service provider *SLA Negotiator*.

Either having received an *SLA Request* or initiating negotiation as reaction on a retrieved *Tender* (which is again triggered by the *SLA Manager*, the provider *SLA Negotiator* forwards the *Request* or the *Tender* to the *Decisor* which is capable of either (a) optimization of the *SLA Document* on behalf of the service provider or (b) deciding if to agree or not.

In the first case, the *Decisor* takes the received document and interacts with a set of components on the service provider side. First of all, it contacts (*getTemplate*) the *SLA Template Repository* to see if there are already *SLA Templates* existing which cover the contents of the preferred service. In most cases, there is not a completely matching *SLA Template*, which would then lead to the next step, which covers to check the capabilities of the infrastructure *checkCap*. Here it is of interest whether the requested parameters can be, in general, fully, partially or not at all fulfilled. Having this information the *System Status* component is called, to find out the current system status and to get a prediction of the system status at the time, when the service is needed. Another factor which impacts the optimization is the information about performance of the same or similar services in past. Therefore the *Decisor* requests the performance information from the *QoS History* component *getRelatedInfo*, which logs all events during execution with respect to a certain *Service Level Agreement*. Lastly, the factor *Business Objectives* has to be considered during the optimization

check as providers have, as well as the customers, their business policies which can have a big impact on the contents of the SLA. Such an objective can state the preference of business partners as well as e.g. the maximization of the resource usage. If it is within the policies of the provider, the *Resource Manager* is requested after optimization to initially reserve the system resources based on the contents of the document, but this is not a must.

Now, the updated document can be sent back to the customer, which then itself requests the *Decisor* to check if to accept/reject an Offer or if to produce an counter-offer based on his policies.

Whenever one of the business parties *Decisor* decides to agree on the current version of the SLA document, it signals this to the other party and sends the agreed document *signed* by the *Signer* component to the opponent. When both parties have *signed* the document, it is a Service Level Agreement, which is *stored* on both sides with a unique ID in an *SLA Repository* and acts as information source for the further processes within the SLA Management.

A detailed description of the components is given in Table 4.4.

Component	Domain	Description
SLA Negotiator	C/SP	The <i>SLA Negotiator</i> is a component on both, service provider and customer side and acts on behalf of the respective entity to process incoming requests for finding an agreement from the according opponent. During the negotiation phase it is calibrated based on the business context.
SLA Manager	C/SP	The <i>SLA Manager</i> is the managing unit on both sides. It triggers the analysis of the context and initiates the negotiation process. Depending on the chosen discovery process, either the service provider or the customer <i>SLA Manager</i> calibrates the <i>SLA Negotiator</i> on the best fitting protocol and starts the negotiation process.
Context Analyzer	C/SP	This component is responsible for analyzing the context (as stated by its user) and to calibrate the <i>SLA Negotiator</i> with the correct protocol.

Component	Domain	Description
Decisor	SP	This component shows responsible for taking the decision of acceptance of an incoming SLA Request or of the optimization of the parameters and by that to create an adapted document which is then sent to the customer. In the latter case, this is optimized based on the already designed templates, the general infrastructure capabilities the current system status, the past performance of services and checking the <i>Business Logic</i>
Decisor	C	The <i>Decisor</i> component on the customer side decides similarly to the service providers <i>Decisor</i> if an incoming document is acceptable or not. Additionally it has also the capabilities to adapt this document to a so-called Counter-Offer, based on the preferences and business policies of its owner. The Counter-Offer is then sent back to the Service provider
Infrastructure Capabilities	SP	<i>Infrastructure Capabilities</i> is the component that delivers a general overview about the properties of the infrastructure. This is independent from the current state of the system, just providing the information about what is generally possible.
System Status	SP	The <i>System Status</i> part of the architecture delivers the necessary means for gaining knowledge of the current system status. In case of the negotiation, the <i>Decisor</i> can get from this component the information it needs to create an acceptable and feasible offer. Additionally the <i>System Status</i> component can also predict the status in future, to deliver the necessary information for services which are about to start later.
Business Objectives	C/SP	This component allows for taking into account the business policies of the respective entity.
SLA Template Repository	SP	The <i>SLA Template Repository</i> contains all the designed templates of the service provider. As mentioned already, the number of templates in this repository can be higher than the number published in Trusted Third Party Registries, as the service provider usually has hidden templates for certain customers.

Component	Domain	Description
QoS History	SP	This component contains the information about the past performance of services based on the respective SLAs. By that, the <i>Decisor</i> can see the risk of providing a certain quality of service level in case this has been provided before.
Resource Manager	SP	The <i>Resource Manager</i> is the responsible component which can reserve resources needed for a certain service. If preferred, the service provider can make an offer and at the same time reserve the necessary system bits, of course always with the danger of having them blocked in case another customer asks for a similar service and the previous negotiation is not finished yet.
Signer	C/SP	To make the Service Level Agreement valid, it needs to be signed by both parties. This is the task of the <i>Signer</i> .
SLA Repository	C/SP	After the (successful) negotiation of an SLA, the SLA is stored on both sides in the respective <i>SLA Repository</i> retrievable by a unique ID.

Table 4.4.: Description of the SLA Negotiation Components

4.3.3. Provisioning

Having established a Service Level Agreement, the systems of both parties, customer and provider, have to be configured according to the terms and obligations in the document. Additionally, when chosen, Trusted Third Party components have to be configured (currently only foreseen for evaluation purposes). All this is handled during the provisioning phase which has to address the following requirements:

1. Service providers need in general to configure their resources based on the SLA
2. The service provider should be able to configure his mechanisms for controlling its systems and services performance based on the SLA Parameters
3. The customer shall be enabled to configure his mechanisms to allow for evaluation of the performance of the service provider



Figure 4.14.: The SLA Lifecycle - Focus on Provisioning

4. Customers should be able to configure third party evaluation mechanisms, in case they do not want to host such a service internally
5. If not done during negotiation, service providers need to reserve their resources
6. Service providers do not want to publish their internal system level data to the outside, therefore they need to configure their system to have a converter for the SLA terms

Based on this list, the architecture sketch for the Provisioning phase was created as seen in Figure 4.15. The Provisioning architecture covers all the aforementioned requirements. When an SLA is available the service provider needs to configure its system. This is started by the *SLA Manager*, which is contacted to do so from either a human user or another layer management component (*triggerProvisioning*).

As there might be several SLAs for business relationships of the service provider with customers, the *SLA Manager* needs to be provided by that SLAID (the Unique ID) which was assigned to the SLA. By that, it can *retrieve* the SLA from the *SLA Repository* and start configuration on basis of that.

For the purpose of assessing the performance of the service, the *SLA Manager configures* a monitoring component (the *SLA Monitor*) and an evaluating component (the *SLA Evaluator*) on the basis of the SLA. If external evaluation is requested, a *Data Converter* component has to be *setup* which will provide the monitored data in the format of the Service Level Agreement terms. Internally in the service provider domain, an *SLA Evaluator* instance may be configured with a Preventive SLA to allow for preventive Evaluation. This is the same process as "normal" configuration of the *SLA Evaluator*. Additionally, the *SLA Manager* can request the reservation

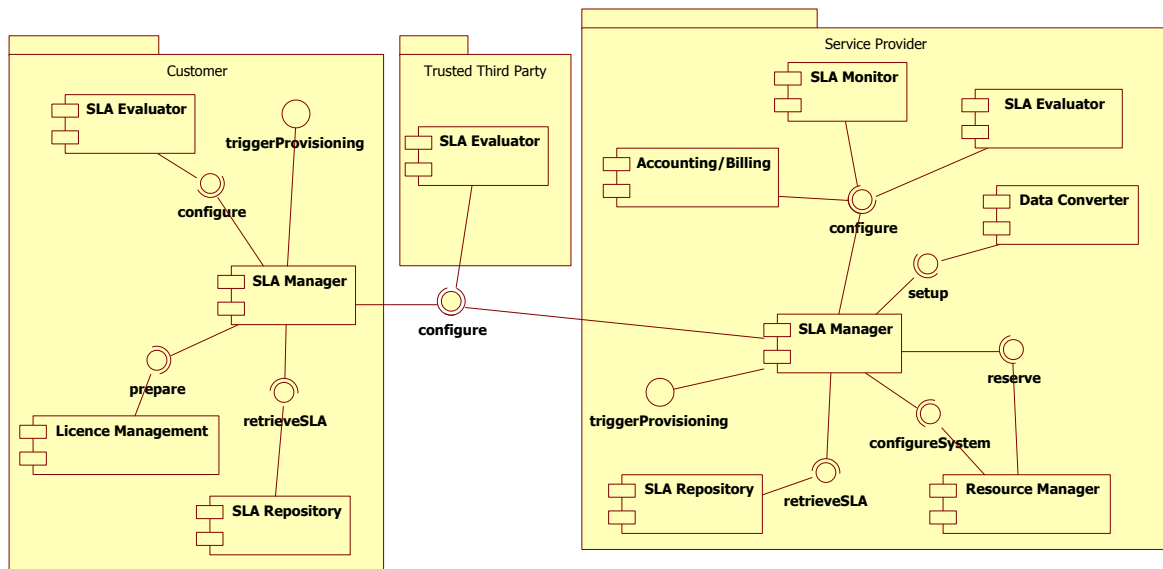


Figure 4.15.: Architecture of the Provisioning Phase

(*reserve* of the needed resources at the *Resource Manager* if not done already during the Negotiation phase. The same component is also requested by the *SLA Manager* to *configure the system(Resources)*).

Finally, the mechanisms for *Accounting and Billing* are *configured* on the basis of the SLA. By that, they will be able to unambiguously assign the values to the respective SLA-ID.

Provisioning also affects the customer domain. First of all there might be obligations on the customer, e.g. that he provides his own licenses. In that case the *SLA Manager*, when requested to start the provisioning (*triggerProvisioning*), gets (*retrieveSLA*) the SLA with the respective ID from the *SLA Repository* and *prepares* the *Licence Management* to support the execution of the service. Furthermore, if stated in the SLA, external evaluation can take place on behalf of the customer, either in his domain or provided by a Third Party. In both cases the *SLA Manager* configures the *SLA Evaluator*. Alternatively, the TTP *SLA Evaluator* can also be configured by the service provider. This is a matter of trust.

Table 4.5 gives a short overview of all the involved components.

Component	Domain	Description
SLA Manager	C	This component is responsible to configure the customer side domain for the service execution. Based on the SLA Parameters, the <i>SLA Manager</i> configures the evaluating component either in his domain or on the Trusted Third Party as well as it prepares everything to meet his obligations
SLA Evaluator	C	The customer <i>SLA Evaluator</i> is configured to evaluate the monitored data sets of the service with respect to the SLA Terms
SLA Repository	C	This component owns a database with all the SLAs of the service provider, assigned to Unique IDs. It allows for querying and retrieving the SLAs for configuration purposes
Licence Management	C	This component represents the obligations which may be on the customer for the service execution. If the customer agreed to provide his licenses for the service execution, this component is prepared to allow access to those
SLA Evaluator	TTP	The TTP <i>SLA Evaluator</i> is configured to evaluate the monitored data sets of the service with respect to the SLA Terms. It can be configured by either the customer or the service provider <i>SLA Manager</i>
SLA Manager	SP	The service provider <i>SLA Manager</i> tasks are to prepare the service providers system for the execution of the service. This covers the configuration of the <i>SLA Monitor</i> , <i>SLA Evaluator</i> and <i>Accounting/Billing</i> . Furthermore it triggers, if needed, the resource reservations and the configuration of the system and prepares, in case of external evaluation, the <i>Data Converter</i>
SLA Repository	SP	This component owns a database with all the SLAs of the customer, assigned to Unique IDs. It allows for querying and retrieving the SLAs for configuration purposes
Resource Manager	SP	The <i>Resource Manager</i> allows for reservation of the resources which are needed for the service. Through this component, the System can be configured
SLA Monitor	SP	The <i>SLA Monitor</i> is configured to provide the system and service execution data for the service provider. This allows for assessment during the execution

Component	Domain	Description
SLA Evaluator	SP	The service provider <i>SLA Evaluator</i> is configured to evaluate the monitored data sets of the service with respect to the SLA Terms
Data Converter	SP	In case of external evaluation, the service provider does not want to publish his system level data to outsiders. Within the SLA it should be agreed what kind of data is provided to the customer. Therefore the <i>Data Converter</i> is set-up to take the monitored terms and to adapt them in a format, which can be sent to an (<i>customer</i>) <i>SLA Evaluator</i>

Table 4.5.: Components involved in the Provisioning Phase

4.3.4. Execution and Assessment

The next steps in the SLA lifecycle are the Execution and the Assessment of the service. This means to execute the service and to monitor it to allow for an assessment



Figure 4.16.: The SLA Lifecycle - Focus on Execution and Assessment

of its performance. By that it allows for the detection of violations and triggering of counteractive measures. Therefore the Execution and Assessment architecture has to

support the following requirements:

1. Customer may want to evaluate the services performance from outside of the service provider domain
2. Customer need to store the information about the services performance internally for further usage
3. Service provider need to monitor and evaluate their service/system performance internally
4. Service provider need to be able to reconfigure their system if necessary
5. Service provider need to log their service/system performance internally for the purpose of accounting and billing and for further refinement of the service offerings

Based on this list, the architecture sketch for the Execution and Assessment phase was created as can be seen in Figure 4.17. During the Execution and Assessment phase,

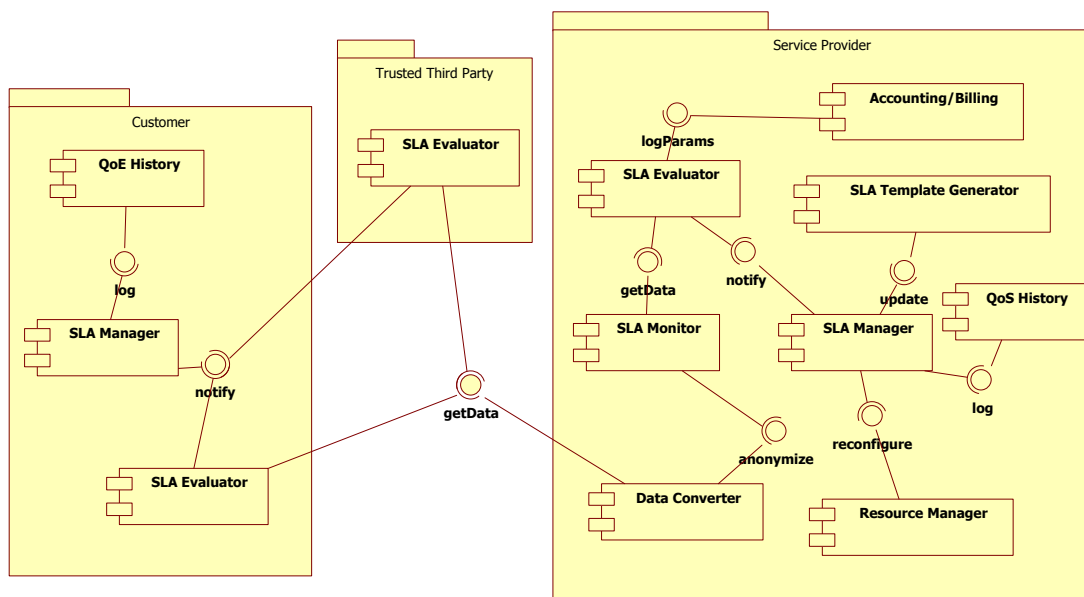


Figure 4.17.: Architecture of the Execution and Assessment Phase

the *SLA Monitor* is, depending on its configuration, constantly or in time intervals providing the data(*getData*) to the service provider *SLA Evaluator*. In case of external evaluation it also sends the data to the *Data Converter* to *anonymize* it as far as was agreed within the Service Level Agreement.

The SLA Evaluator in the provider domain evaluates the monitored data with respect to the parameters in the SLA and logs the usage of the service in the *Accounting/Billing* component. In case of events which are requesting an involvement of the providers *SLA Manager* (e.g. a violation of the SLA), the managing component is *notified*. This notification is then *logged* in the *QoS History* and the SLA Manager decides how to process it.

The action from the SLA Manager depends on the policies of the service provider. This can range from "do nothing" up to a "reconfiguration of the service". In case the latter is decided, the *Resource Manager* is triggered to *reconfigure* the system. Whenever it becomes obvious that this service is often violated, the service provider *SLA Manager* can also decide to trigger the *SLA Template Generator* for an update of the respective Templates of the service.

On side of the customer, mainly, if agreed in the SLA, the external evaluation is performed. Therefore either the customer *SLA Evaluator* or a Trusted Third Party *SLA Evaluator* is requesting the data from the *Data Converter* and evaluating it (*getData*). When an event is detected, the SLA Evaluator component *notifies* the customer *SLA Manager*, which then *logs* this in the *QoE History*.

Table 4.6 gives a short overview of all the involved components.

Component	Domain	Description
SLA Manager	C	In this phase, the customer <i>SLA Manager</i> is responsible for logging the performance of the service as evaluated by its <i>SLA Evaluator</i> in the QoE (Quality of Experience) history.
SLA Evaluator	C	This component takes the monitored data as provided by the <i>Data Converter</i> and evaluates it against the SLA terms. In case of detecting a violation, the <i>SLA Evaluator</i> notifies the <i>SLA Manager</i>
QoE History	C	This component is a storage for the evaluated performance of the service. This storage may later be used to estimate the trustworthiness of service providers as well as to check the charging of the provider on its accuracy
SLA Evaluator	TTP	This component takes the monitored data as provided by the <i>Data Converter</i> and evaluates it against the SLA terms. In case of detecting a violation, the <i>SLA Evaluator</i> notifies the <i>SLA Manager</i>

Component	Domain	Description
SLA Manager	SP	The service provider <i>SLA Manager</i> is notified by the internal <i>SLA Evaluator</i> in case of events like violations or foreseen/upcoming violations. Based on the information received, this component can decide whether to do nothing or to start the reconfiguration of the system. Furthermore it logs the performance in storage to be able to make a long term analysis of the behaviour of the service. If it detects that a misbehaving service is doing this continuously, the <i>SLA Manager</i> may request an update of the Templates at the <i>SLA Template Generator</i>
Resource Manager	SP	This component is during this phase responsible for on-the fly reconfiguration of the system to be able to fulfill the SLA.
SLA Monitor	SP	The <i>SLA Monitor</i> monitors the system and provides the data to the <i>SLA Evaluator</i> and to the <i>Data Converter</i> , if needed.
SLA Evaluator	SP	This component takes the monitored data as provided by the <i>SLA Monitor</i> and evaluates it against the SLA terms. In case of detecting a violation, the <i>SLA Evaluator</i> notifies the <i>SLA Manager</i> which then can trigger counteractive measures. The service provider internal <i>Evaluator</i> can be also used for preventive monitoring and evaluation of the service.
QoS History	SP	The <i>QoS History</i> (Quality of Service) is storage for all the events during the execution and assessment of the service. It can be used as data source for analysis of the service and system behaviour.
SLA Template Generator	SP	The <i>SLA Template Generator</i> provides the functionality to adapt/update existing SLA Templates, if this becomes necessary due to the performance assessment of the SLAs based on them
Data Converter	SP	This component takes the monitored data and provides it for the external <i>Evaluators</i> in a format, which is easy to compare with the SLA but does not contain any service provider confidential data
Accounting/Billing	SP	<i>Accounting and Billing</i> get the necessary information from the <i>SLA Evaluator</i> to perform later their tasks of final clearing

Component	Domain	Description
-----------	--------	-------------

Table 4.6.: Components involved in the Execution and Assessment Phase

4.3.5. Termination

Finally, the *termination block* (Figure 4.18) takes over the role of terminating and dissolving the SLA. As described in 4.2.6, this can either happen normally, when the



Figure 4.18.: The SLA Lifecycle - Focus on Termination

service and the SLA have ended, but also in case of violation of the contract, generally based on the definitions within an SLA. Therefore the Termination architecture has to support the following requirements:

1. Service providers need to release their bound resources for the service
2. In case of an abortion of the service, the final usage needs still to be logged for accounting and billing purposes
3. SLAs need to be disabled in terms of their validity
4. The components configured for this service need to be de-configured

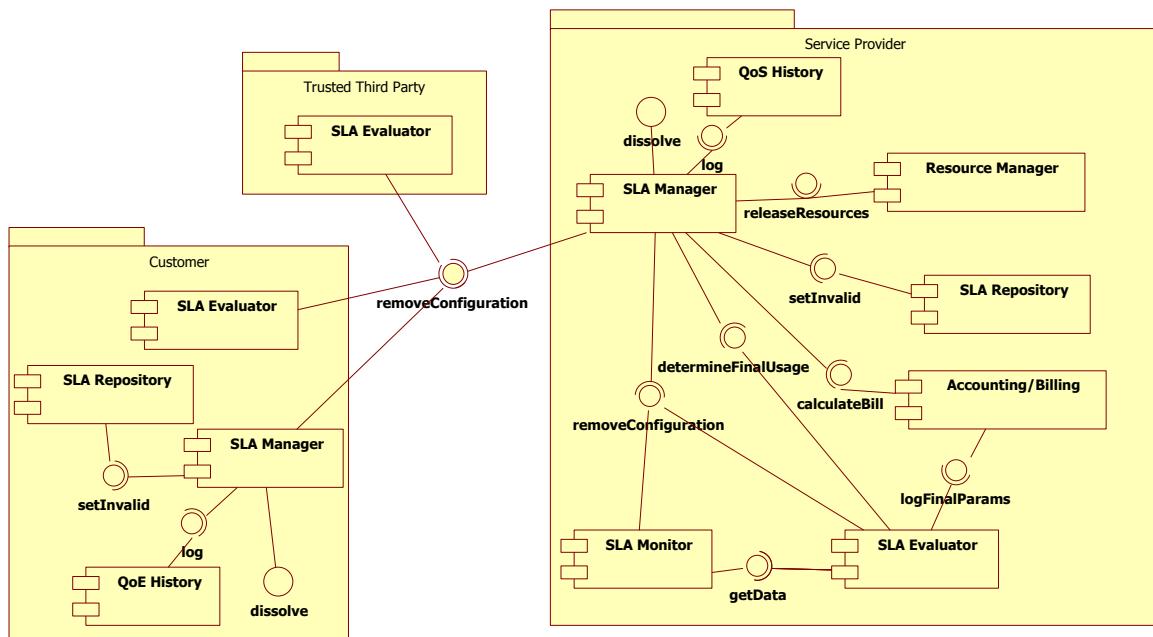


Figure 4.19.: Architecture of the Termination Phase

Based on this list, the architecture sketch for the Termination phase was created as can be seen in Figure 4.15. Independent of the reason for termination, the *SLA Manager* components in both domains are triggered from a higher level management component or a human user to *dissolve* the SLA. On side of the customer this means to *log* the reasons and the latest information at the *QoE History* and to de-configure the *SLA Evaluators* which were configured by the customer *SLA Manager*. Additionally the SLA has to be set invalid in the *SLA Repository*

On side of the service provider the *SLA Manager* has to perform several tasks in case of termination/dissolution. First of all, the *Manager* logs the latest data in the *QoS History* and sets the SLA in the *SLA Repository* as invalid. Then, before de-configuration of the components, the service provider *SLA Evaluator* is requested to *determine the final usage*. Therefore the *SLA Evaluator* queries the data from the *SLA Monitor* (*getData*), evaluates this and logs the parameters in the *Accounting/Billing* component (*logFinalParams*).

As last activities, the *SLA Manager* triggers the *Accounting/Billing* component to *calculate the bill* and *releases the Resources* via the *Resource Manager*. Table 4.7 gives a short overview of all the involved components.

Component	Domain	Description
SLA Manager	C	This component is responsible for the handling of the dissolution process on the side of the customer. It de-configures the <i>SLA Evaluator</i> and sets the SLA in the customer <i>SLA Repository</i> invalid whilst logging the last parameters to the <i>QoE History</i>
SLA Evaluator	C	The customer <i>SLA Evaluator</i> is in this phase no longer active, but de-configured by the customer or the service provider <i>SLA Manager</i>
QoE History	C	This component is storage for the evaluated performance of the service. It is filled with the latest evaluation data by the customer/TTP <i>SLA Evaluator</i>
SLA Evaluator	TTP	The TTP <i>SLA Evaluator</i> is in this phase no more active, but de-configured by the customer or the service provider <i>SLA Manager</i>
SLA Manager	SP	This component is responsible for the handling of the dissolution process on side of the service provider. It manages the wrap up of the service provider system, namely triggering the final usage parameters, logging the last performance data, de-configuration of the SLA components, and releasing the service-assigned resources
Resource Manager	SP	In this phase, the <i>Resource Manager's</i> task is to release all service-assigned resources for other usage, when requested
SLA Monitor	SP	Before the <i>SLA Monitor</i> is de-configured, it has to provide the last monitored data to the <i>Evaluator</i>
SLA Evaluator	SP	The <i>SLA Evaluator</i> requests the final monitoring data from the <i>SLA Monitor</i> , evaluates this and logs this for accounting and billing purposes
Accounting/Billing	SP	This component retrieves the data of the final usage from the <i>SLA Evaluator</i> and offers the functionality to calculate the bill on demand of the <i>SLA Manager</i>
QoS History	SP	The <i>QoS History</i> (Quality of Service) is storage for all the events during the execution and assessment of the service. In the termination phase the final parameters are logged by the <i>SLA Manager</i>

Component	Domain	Description
SLA Repository	C/SP	The <i>SLA Repository</i> is the storage for the Service Level Agreements which belong to either customer or service provider. In this phase it provides the means for setting an SLA invalid, to ensure, that no more resource are bound to this.

Table 4.7.: Components involved in the Termination Phase

4.3.6. Message flow between the different lifecycle phases

As a short overview, Table 4.8 shows the input and outputs of the different SLA lifecycle phases/architectural blocks.

Block	Input	Status
Development	Service Capabilities, System Analysis, Business Policies, Performance Indication	SLA Templates/SLA Requests created
Creation	SLA Offer/SLA Request	SLA agreed and probably reservation performed
Provisioning	SLA and reservation	Service is ready to execute
Execution	SLA	Monitoring Data and Usage Information is collected; Event logs written
Assessment	Monitoring Data, SLA,	Performance Indication gained
Termination	SLA/Accounting Data/Event log	Resource released, billing process executed

Table 4.8.: Mapping of the Service to the SLA Lifecycle Phases

4.4. Enhanced Creation of Service Level Agreements

So far, SLA Creation was handled as replacement for the actions of Service Discovery and SLA Negotiation. But looking at both in detail, it is obvious (cf. also the section on State of the Art, Chapter 3.5) that with this existing variety of Discovery and Negotiation mechanisms also all possible concatenations have to be taken into account.

How to choose the best fitting combination of Discovery and Negotiation?

For that purpose, this thesis introduces an additional factor for the choice of these parts - "context" (of business).

The definition of context is thereby aligned with the one provided in 2001 by Dey [79]

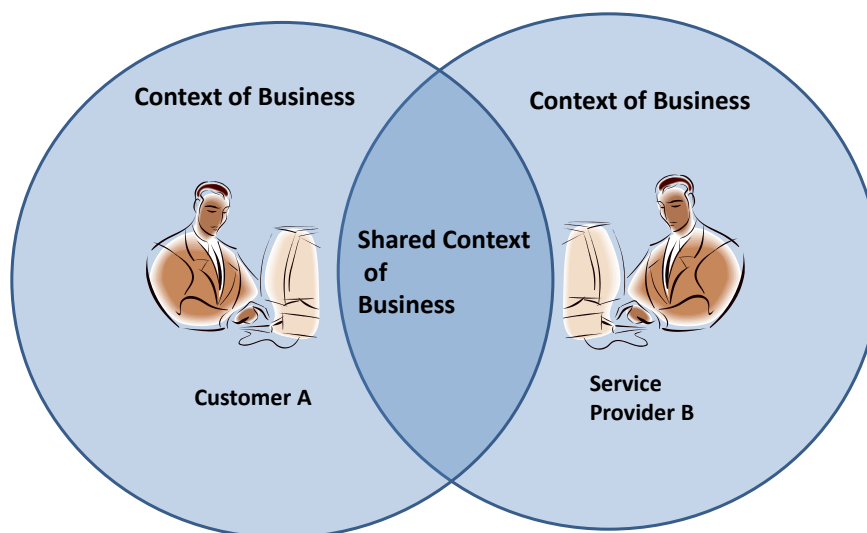


Figure 4.20.: Business Entities sharing a Common Context

who defined context as "any information that can be used to characterize the situation of an entity." This situation and its characterization is exactly what is influencing the way agreements are made.

Looking back to the three scenarios as described in Chapter 2, their context can be characterized as follows:

- **Medical Support:** This use case is settled within the Urgent Computing domain. The urgency therefore implies that the creation phase is handled as quickly as possible. Therefore also discovery and negotiation have to be performed without a huge investment of time. Potential candidates for mechanisms would

be to choose during discovery already known service providers and to negotiate using the discrete offer protocol.

- **Car Crash Simulation:** In this scenario, the degree of flexibility is very high. There is enough time to discover the best fitting provider which needs to fulfil all restrictions, especially in terms of confidentiality and reliability. Here one could use the yellow pages discovery combined with a multiphase SLA Negotiation.
- **Visualisation:** This use case is in terms of flexibility on the duration of the creation phase somewhere in between the Car Crash Simulation and the Medical Support. Depending on when the wind tunnel is available and booked, this is the deadline when creation and provisioning have to be finished. When the timespan is bigger a multiphase Negotiation could be tried, if it fails, a discrete offer one could be performed.

At the end of the creation phase, both entities should be aware of the context - they share it as shown in Figure 4.20.

Chapter 5.

An SLA Schema for the HPC domain

Having derived the architecture for SLA Management as shown in chapter 4, this chapter will show how Service Level Agreements can be represented electronically in terms of format and language. Similarly to the domain of architecture, effort has already been spent on the definition of SLA Schemata and languages. However the evolution of most of them, e.g. SLAng (cf. chapter 3.2.4) or WSLA (cf. chapter 3.2.1) stopped without the results having reached sufficient maturity or specificity to be taken up by other activities. The only one which seems to be prominent and mature enough is WS-Agreement, which provides a high degree of flexibility (this was originally intended as it did not aim to deliver a concrete SLA language), but exactly those flexibility causes the need of adaptation of the specification to fit to the needs of a special domain.

The aim of this chapter is to take WS-Agreement as a starting point and to derive, based on the discussed requirements in chapter 2 an SLA Schema for the High Performance Computing domain.

To overcome the obstacles of current solutions a compromise between a detailed definition of terms (down to atomic level) and at the same time keeping the degree of flexibility high is needed. Therefore this schema will provide the means of fine granular definition of terms but also place holders for further extensions which might not yet be addressed. All that with a dedicated focus on the High Performance Computing domain.

5.1. Scope for the Schema definition

Before developing a proper Service Level Agreement Schema for a certain domain, its scope needs to be defined. This means to concretely identify (a) what will the schema be used for and (b) what should be its contents

5.1.1. Usage of the Schema

Generally the schema should be usable for the whole SLA Lifecycle and by that be processable in the previously presented architecture (cf. chapter 4). In 4.1 the different kinds of SLA documents were presented, namely *SLA Template*, *Tender*, *SLA Request*, *SLA Offer*, *SLA Counter-Offer*, *SLA Agreement* and *Preventive SLA*.

Their relationship to the different SLA Lifecycle Phases can be seen in Table 5.1. This

Lifecycle Phase	Involved Document Type
Development	SLA Template, Tender, SLA Request
Creation	SLA Template, Tender, SLA Request, SLA Offer, SLA Counter-Offer, SLA Agreement
Provisioning	SLA Agreement, Preventive SLA
Execution	SLA Agreement, Preventive SLA
Assessment	SLA Agreement, Preventive SLA
Termination	SLA Agreement

Table 5.1.: Mapping the SLA Documents to the Lifecycle Phases

this thesis aims to deliver a schema, which allows for instantiation of all these different kind of documents for SLA Management. In order to do that, it will have to define carefully common parts of the documents but also document specific ones. E.g. most of the documents used in the *Development* and *Creation* phase will need to contain information needed to come to an agreement whilst later, when the agreement is established, this information is not needed anymore.

5.1.2. Contents of the Schema

The contents of the schema go beyond the definition of only service related parameters. Within this thesis, the contents of an SLA build, but limit themselves not only, on the design objectives of WSLA (cf. 3.2.1).

For a proper representation of a bipartite relationship the following information must be contained:

- An unique identifier of the document
- Information/Description of the involved business parties (including probable supporting third parties), including their roles

- Detailed definition of the service terms and the agreed guarantees from both, customer and service provider
- Definition of the obligations on the parties
- Exclusion terms defining, what the parties are not responsible for
- Constraints which have to be taken into account by the parties during the creation of the agreement

5.2. WS-Agreement as starting point

Even though the content definition was based on the design objectives of WSLA, the analysis of the state of the art with respect to maturity, usability and provided capabilities, recommended WS-Agreement as the most promising candidate for providing the basis for the SLA Schema.

WS-Agreement itself only provides a definition of two document types, the SLA and the SLA Template. This is generally not sufficient for the aim of this thesis as this envisages seven different document types (SLA Template, Tender, SLA Request, SLA Offer, SLA Counter-Offer, SLA Agreement and Preventive SLA). This is a first mandatory change/update with respect to this specification as well as others are necessary which are presented in the following sections of this chapter after a short introduction to the general structure of the WS-Agreement documents which are taken as basis for schema developed in this thesis.

5.2.1. WS-Agreement: The Agreement Structure

It's basic structure contains sections for *Name*, *Context* and *Terms*, as shown in Figure 5.1. According to the latest WS-Agreement Specification, the **Name** part is an optional placeholder to assign a certain name to the agreement. It was introduced to allow for adding a human-understandable name to an SLA but is NOT a unique identifier.

Context provides that information about the agreement, which are not automatically service specific. E.g. this can be the information about the business entities or the duration of the agreement.

Finally, the last part contains the **Terms** of an agreement. This contains the concrete service definitions as well as guarantees which were agreed by customer and Provider.

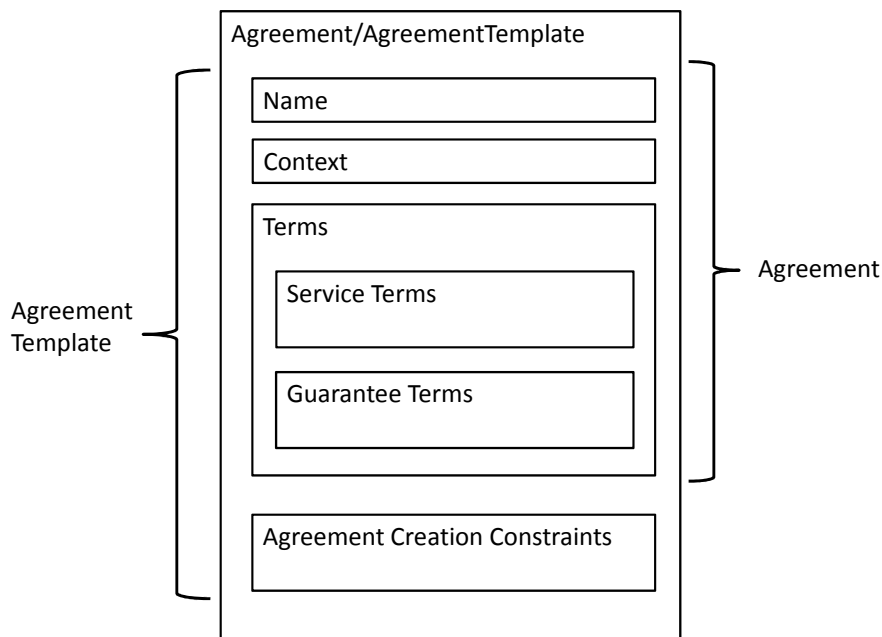


Figure 5.1.: The High Level Structure of WS-Agreement (Templates)

5.2.2. WS-Agreement: The Template Structure

The Template Structure of WS-Agreement is similar to the Agreement Structure. There is just an additional part for **Creation Constraint** which is intended to provide constraints on the values that the various terms may take in a concrete agreement.

5.3. The Schema

As described before, the basic structure of the derived SLA Schema builds upon WS-Agreement (Version 1.0, GFD.107, March 2007). However, when going deeper into the detail and focusing on the different sub-nodes of the elements, adaptations are needed. This implies in some cases only minor modifications, but in other cases, bigger changes are required. Additionally, as WS-Agreement does not provide a language to define the detailed terms of an SLA, the provided schema will take the best fitting parts from other specifications such as JSDL or WSLA and embed them in the elements, where applicable. In general, all adaptations are shortly discussed in the sub chapters of the section.

The first change was the decision to give the Schema an own namespace - hpcwsag (HPC WS-Agreement).

In the following, the different elements of an Agreement and a Pre-SLA Document are described. This is a change with respect to WS-Agreement as this thesis introduces the Pre-SLADocument as the basis for the *SLA Template*, *Tender*, *SLA Request*, *SLA Offer* and *SLA Counter-Offer* documents type. By that, it supports negotiation protocols beyond the classical "customer-asks-service provider" approach.

The description of the elements is given either as own definition, or, where applicable as a citation from the standards used (e.g. "Taken from WS-Agreement": marks that the following text is a citation from the current version of the WS-Agreement specification).

The high level structure of an SLA Agreements looks like shown in Figure 5.2. With the parts *Name*, *Context* and *Terms* it is similar to the WS-Agreement base

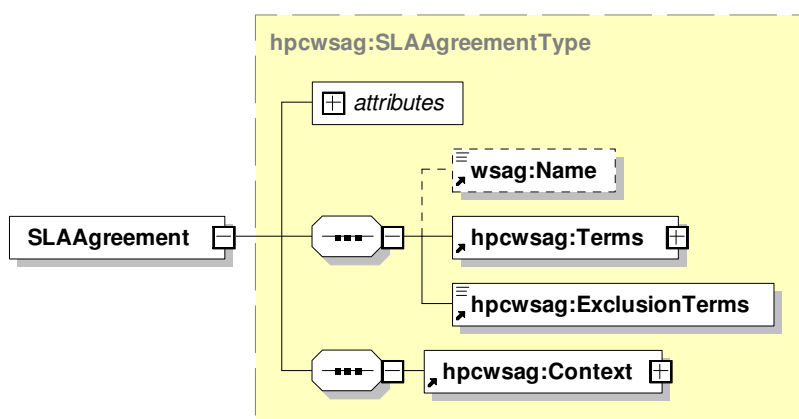


Figure 5.2.: The High Level Structure of the Agreement Schema

structure. As well as the specification, it foresees an attribute *hpcwsag:AgreementId* which is a mandatory unique identifier of the agreement. But there is also an addition: *ExclusionTerms*.

This element was developed based on the examination of Service Level Agreement handling of other centers or industry (e.g. Amazon EC2 SLAs¹ or the Michigan State University HPC Center [80]). They use *Exclusion Terms* to assure themselves against events which can not be controlled by themselves (e.g. attacks from outside).

The structure of a PreSLA-Document is based on the structure of WS-Agreement Templates with some modifications. Similarly it adds the part *Creation Constraints*, as defined by WS-Agreement, but the *Context* part is now different. This was necessary to avoid not-needed tags within the different bits, which would have made the

¹Amazon EC2 SLA, <http://aws.amazon.com/ec2-sla/>

structures more complicated than it has to be. The adapted structure is shown in Figure 5.3. As well as the Agreement, the PreSLA-Document owns as attribute a

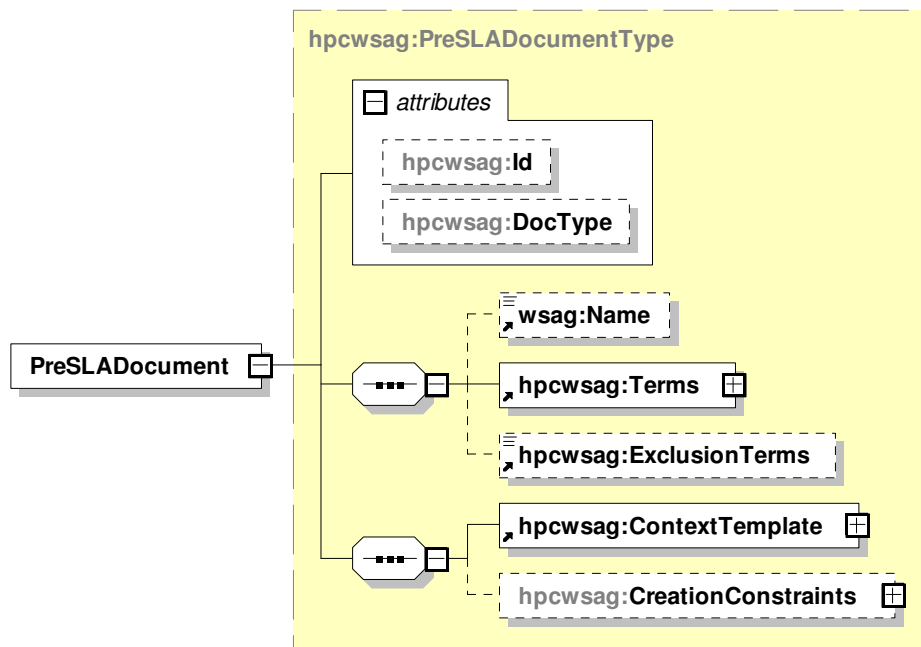


Figure 5.3.: The High Level Structure of the PreSLA-Document Schema

unique identifier - *hpcwsag:Id*. This attribute is mandatory and has to be updated, if documents are adapted or re-designed. In addition, the attribute *hpcwsag:DocType* delivers the means to define the type of document it represents. This can be *Template*, *Tender*, *Offer*, *Request*, *Counter-Offer* or *Preventive*.

5.4. Name

The Name field is the same as defined in WS-Agreement:

"This is an OPTIONAL name that can be given to an agreement. The name of an agreement is independent of the name(s) of the template(s) it is based on. The Name element is NOT a unique identifier. It MAY be used to provide a human-understandable name to an agreement."

5.5. Context

As mentioned in the introduction, the *Context* part of this schema contains those datasets which are not directly service parameters but SLA specific. Most of them are based on the analysed non-scenario specific requirements as presented in chapter 2.2. Thereby the Context bits of the Agreement and the Templates differ by two elements, as will be presented in the following subsections.

Important common elements of Agreement and Template are the definition of the involved parties, which covers the *customer*, the *service provider* and potential *Third Party(ies)*. This is an implication of Requirement R18 ("*Contact details of the involved parties should be retrievable*").

Requirement R19 ("*The origins of the SLA should be traceable*") pointed to the possibility of embedding information about the origins of an SLA into the document to allow for enhanced backward traceability. This is realized with a place holder called *DocumentBase*, which can be used, clearly state those origins.

In addition to the so far presented parts of the *Context*, the *Context* of a PreSLA-Document may contain additional information about the *Initiator* and the *Responder* of the Agreement, which is as well a consequence of Requirement R19.. Finally, as one of the requirements in chapter 2.2 referred to the definition of the validity of the SLA and its terms (R21), the *Validation* element shows responsible for specifying the exact points in time when the agreement will be valid.

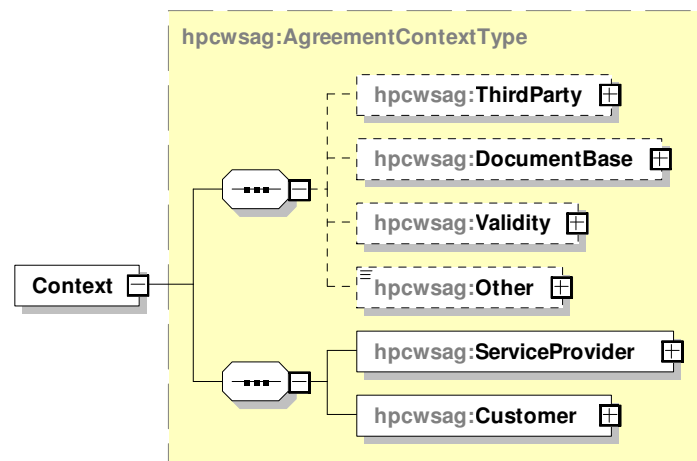


Figure 5.4.: Structure of the Context Part of an Agreement

5.5.1. hpcwsag:Customer

This element identifies the customer as requested by R19. An Agreement has to have exactly one customer description. The structure of the customer as well as of the service provider Element is shown in Figure 5.5, called Party(Type). As sub elements the content of traditional contact forms were chosen.

Thereby the sub elements are defined as shown in Table 5.2.

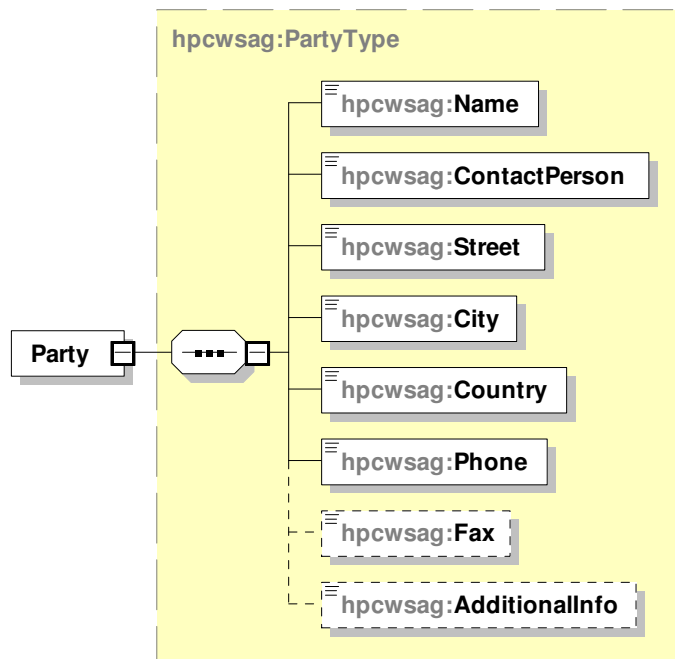


Figure 5.5.: Structure of the Party Description

5.5.2. hpcwsag:ServiceProvider

This structure identifies the service provider as requested by R19. An Agreement must have exactly one service provider description. It follows the same structure as the customer element.

Element	Spec	Description
Name	hpcwsag	This element owns the (company) name of the Party.
ContactPerson	hpcwsag	This element defines the name of a contact person.
Street	hpcwsag	This element contains the street of the Party.
City	hpcwsag	This element identifies the city of the Party.
Country	hpcwsag	This element identifies the country of the Party.
Phone	hpcwsag	This element identifies the phone number of the Party.
Fax	hpcwsag	This element identifies the fax number of the Party.
AdditionalInfo	hpcwsag	This element is a place holder for further descriptions of the Party.

Table 5.2.: Elements of the Party Description

5.5.3. hpcwsag:ThirdParty

This structure identifies a Third Party as requested by R18. In case that both customer and service provider agree on the involvement of a third party for providing an additional service, it is described in this element. As well as the other party descriptions, it is based on the PartyType structure but has two additional fields to identify its role and the remitter of the service. The Third Party structure is designed as shown in Figure 5.6. Table 5.3 contains the detailed description of the contents of the Third Party description.

Element	Spec	Description
Role	hpcwsag	This element describes the role in terms of tasks of the third party. Currently it is foreseen that this acts as an Evaluation service provider.
Owner	hpcwsag	This element describes on behalf of which party, the third party executes its tasks.

Table 5.3.: Elements of the Third Party Description

5.5.4. hpcwsag:DocumentBase

To enable the original base of the SLA Document (as requested by R19), the *Document-Base* element provides the means to integrate the necessary information. Thereby it

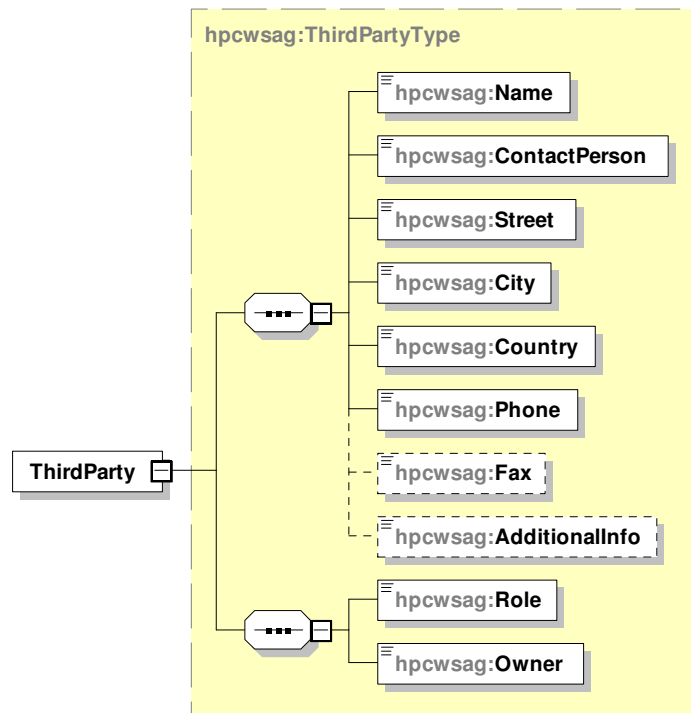


Figure 5.6.: Structure of the Third Party Description

is important to allow for the definition of a unique ID document, the name of the document as well as the document type, according to chapter 4.1. Figure 5.7 shows the structure of the DocumentBase element.

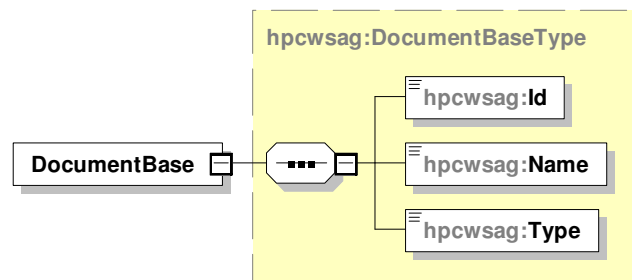


Figure 5.7.: Structure of the DocumentBase Element

Table 5.4 gives a short documentation of the definition of the DocumentBase subterms.

Element	Spec	Description
Id	hpcwsag	This element refers to the specific version of the document which was the basis for this Agreement or PreSLA-Document. If the <i>DocumentBase</i> element is present, the <i>Id</i> has to be specified.
Name	hpcwsag	This element refers to the specific name of the document which was the basis for this Agreement or PreSLA-Document. If the <i>DocumentBase</i> element is present, the <i>Name</i> has to be specified.
Type	hpcwsag	This element defines the kind of document, the Agreement or Pre-SLA was based on. The value range is Template, Tender, Offer, CounterOffer, Request, Preventive. If the <i>DocumentBase</i> element is present, the <i>Type</i> has to be specified.

Table 5.4.: Elements of the DocumentBase

5.5.5. hpcwsag:Validity

Requirement R20 (*"The validity of the SLA and its terms should be definable"*) states the need for an explicit definition of the duration of validity of the Service Level Agreement and its terms. The first is addressed in this element, which defines the validity duration of the SLA within the Context definitions. Its structure is presented in Figure 5.8.

For the representation of the contents of this element, the WSLA Period Type² was chosen as it covers all the necessary contents for this purpose. WSLA Period Type itself is based on the RFC3060 - Policy Core Information Model [81].

²WSLA Period Type, http://soa-blog.net/wsla/infocenter/index.jsp?topic=/net.soablog.wsla.doc/html/spec/period_type.html

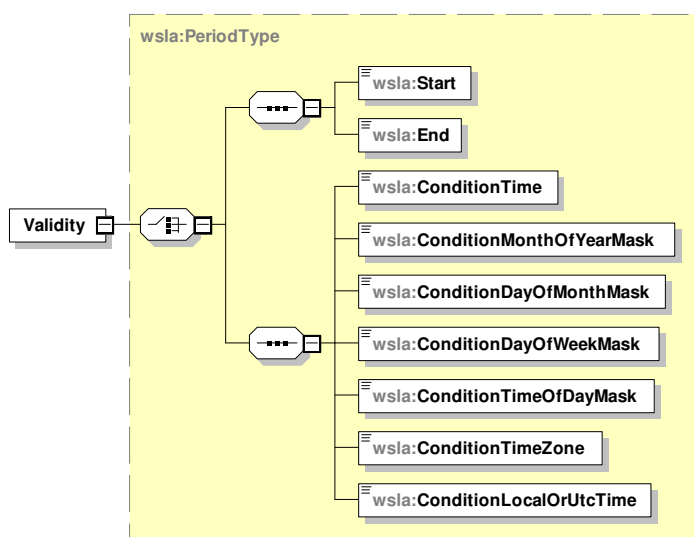


Figure 5.8.: Structure of the Validity Element

Table 5.5 shows the definition of the single elements as taken from WSLA.

Element	Spec	Description
Start	wsla	This element defines the point in time when the validity period starts.
End	wsla	This element defines the point in time when the validity period ends.
ConditionTime	wsla	This element defines the time period that the SLA is valid.
ConditionMonthOfYearMask	wsla	This element defines the time period with respect to the months of the year when the SLA is valid.
ConditionDayOfMonthMask	wsla	This element defines the time period with respect to the days of the month when the SLA is valid.
ConditionDayOfWeekMask	wsla	This element defines the time period with respect to the days of the week when the SLA is valid..
ConditionTimeOfDayMask	wsla	This element provides the means to specify a range of times in a day when the SLA is valid.

Element	Spec	Description
ConditionTimeZone	wsla	This element defines the time zone which is bound to the time definitions.
ConditionLocalOrUtcTime	wsla	This element indicates the origin of time, namely if it is the local time or the Utc time.

Table 5.5.: Elements of the Validity Structure

5.5.6. Additional elements in the PreSLA Context

As requested in R19, the PreSLA-Document contains, compared to the Agreement, information about the Initiator and the Responder of the Agreement. WS-Agreement provides similarly a definition of these entities as an *anyType*. This was taken up as definition of the PartyID, a unique identifier of the respective Initiator/Responder. Additionally also the Role of the initiating/responding party can be defined. By that, the Context part of an PreSLA-Document is structured as shown in Figure 5.9.

5.5.6.1. hpcwsag:AgreementInitiator

This element defines the initiator of an agreement finding process. It consists of a definition of the role of a party as well as an unique identifier. The structure of this element is shown in Figure 5.10.

The single sub elements of the AgreementInitiator element are listed and described in Table 5.6.

5.5.6.2. hpcwsag:AgreementResponder

This element defines the responder to an agreement initiation. Its structure follows the same as the one of the AgreementInitiator.

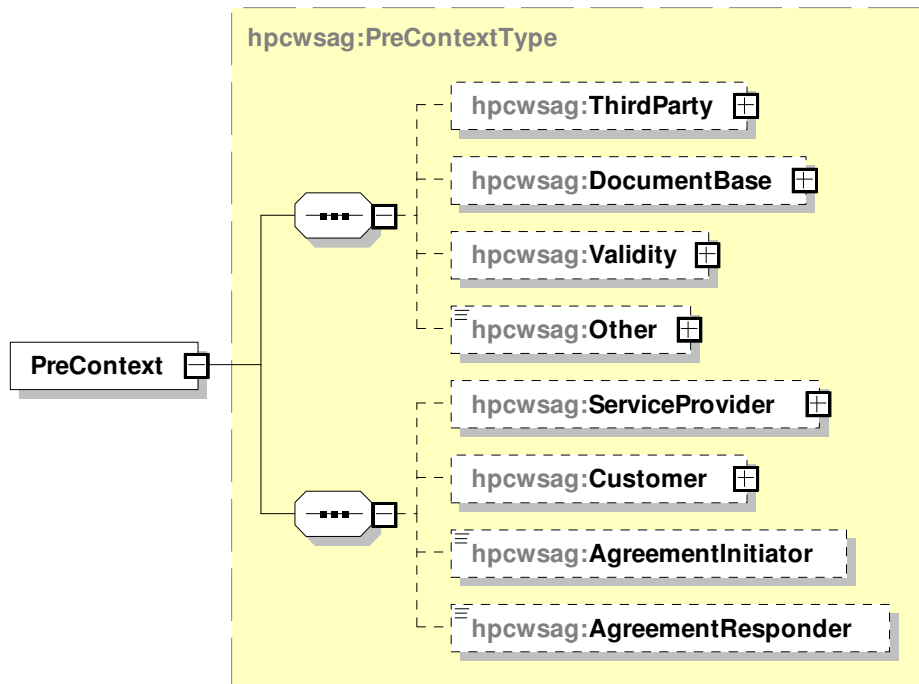


Figure 5.9.: Structure of the Context Part of an PreSLA-Document

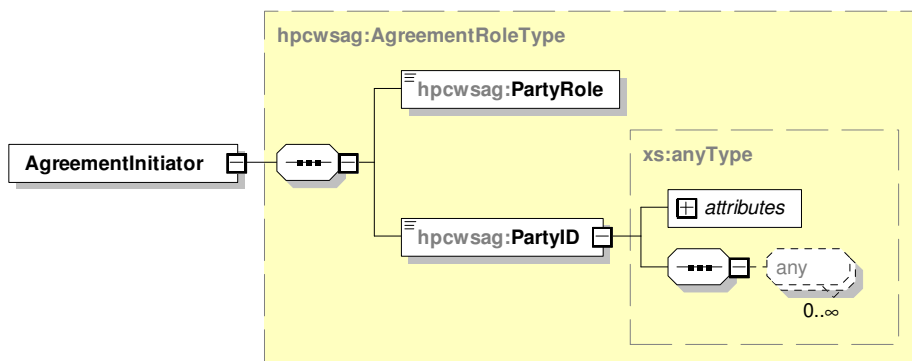


Figure 5.10.: Structure of the AgreementInitiator Element

Element	Spec	Description
PartyRole	hpcwsag	This element defines the role of the initiating party. It can be selected from <i>Customer/Service Provider/Trusted Third Party</i> .
PartyID	hpcwsag	This unique identifier of the party may be a URI or an EndpointReference but also everything else. Therefore this element was defined as anyType.

Table 5.6.: Elements of the AgreementInitiator Structure

5.6. Terms

This part of the Service Level Agreement contains all agreed upon information for the service(s) and its/their capabilities. Getting back to the categorization of the requirements gained in chapter 2, the following high level categories were derived:

- *Resourcing* covering resources and applications
- *Service Availability* covering mainly time constraints
- *Business Handling* covering e.g. penalties, rewards
- *Data Treatment* covering e.g. data access, handling
- *Contractual Terms* covering those aspects which set the contract frame

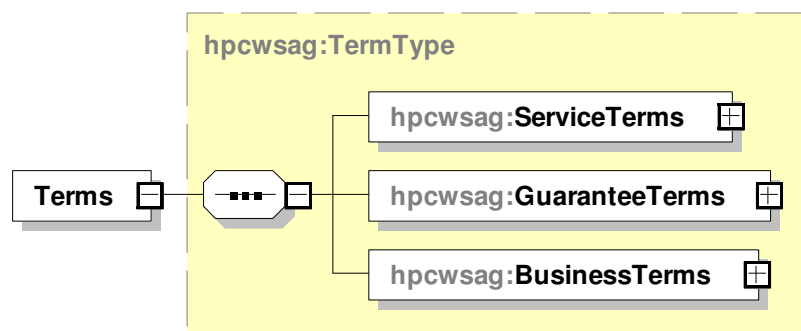


Figure 5.11.: Structure of the SLA Terms

WS-Agreement splits the *Terms* section in two parts: *Service Terms* and *Guarantee Terms*. However with the further definition of the terms, the dependencies and categories get mixed up. For that reason, the decision was taken to split the terms in three parts,

namely *Service Terms*, *Guarantee Terms* and *Business Terms* to avoid any ambiguities (as can be seen in Figure 5.11).

5.6.1. hpcwsag:ServiceTerms

A service and its description is the core of a Service Level Agreement. The *Service Terms* are the means for providing a general *Service Description* together with a *Service Reference*. The Service Terms address Requirement R5 ("Definition of the requested application"), its structure can be found in Figure 5.12.

To represent these terms, WS-Agreement was chosen as its elements cover the needs for the HPC Schema.

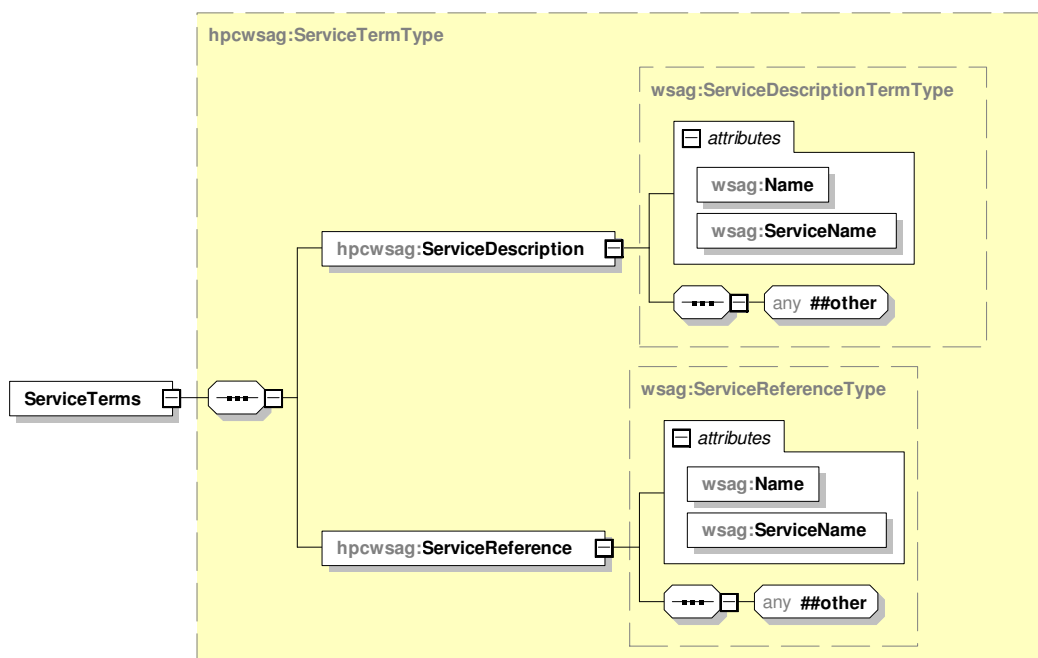


Figure 5.12.: Structure of the Service Terms of the SLA

5.6.1.1. hpcwsag:ServiceTerms/hpcwsag:ServiceDescription

The *Service Description* element is taken from WS-Agreement, it encloses a description of a service or part of a service. Its elements are described in Table 5.7.

Element	Spec	Description
Name	wsag	This element defines the Name of a Service Description Term.
ServiceName	wsag	This MANDATORY attribute identifies a service across multiple service description terms.
Any	wsag	This element is a place holder for a partial or full description of the domain-specific service this service description term is about.

Table 5.7.: Elements of the ServiceDescription

5.6.1.2. hpcwsag:ServiceTerms/hpcwsag:ServiceReference

Analogously to the *Service Description*, the Service Reference is also taken from WS-Agreement. This element is used to point to a service (e.g. by an EPR). It's contents are shown in Table 5.8.

Element	Spec	Description
Name	wsag	This is the name given to this set of service references.
ServiceName	wsag	his attribute identifies a service across multiple service description terms.
Any	wsag	This element is a domain-specific representation of a reference to a service.

Table 5.8.: Elements of the ServiceReference

5.6.2. hpcwsag:GuaranteeTerms - The PreSLA-Document

Service Level Agreements are intended to define the Quality of a (provided) service. But not only the definition is important, also the statements on assurances from both, customer and service providers have to be given to ensure a good and successful

business relationship between both.

Guarantee Terms as used in this SLA Schema cover the statements on those parameters, which decide on a fulfilment of the service or not. This covers the definition of *Resource* related terms, *Application* specific ones, the *Service Availability*, *Data Treatment* related terms and *Other* related contents.

However, to simplify the negotiation process, this element is split for the PreSLA-Document up into two subparts: *Non-Negotiable* and *Negotiable Terms*. By that it allows for concrete definition of which terms may be adapted by the respective negotiation opponent and which not.

The structure of the Guarantee Terms can be found in Figure 5.13.

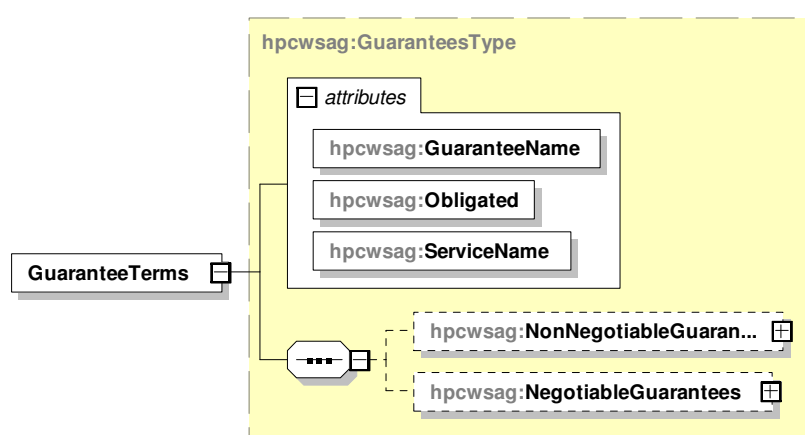


Figure 5.13.: The Guarantee Terms Structure of the PreSLA-Document

Table 5.9 contains the descriptions of the single Guarantee Term attributes.

Element	Spec	Description
Name	hpcwsag	This is the name given to a guarantee.
Obligated	hpcwsag	This attribute defines which party is obliged to keep the defined Guarantee Terms, either the <i>customer</i> , the <i>service provider</i> or the <i>Third Party</i> .
ServiceName	hpcwsag	This is the name of the service, the Guarantee is assigned to.

Table 5.9.: Attributes of the GuaranteeTerms

5.6.2.1. hpcwsag:GuaranteeTerms/hpcwsag:NonNegotiableGuarantees

This element covers all those Terms which are not open for negotiation. The negotiation opponent knows that he has to take these terms as they are.

The element is structured as shown in Figure 5.14.

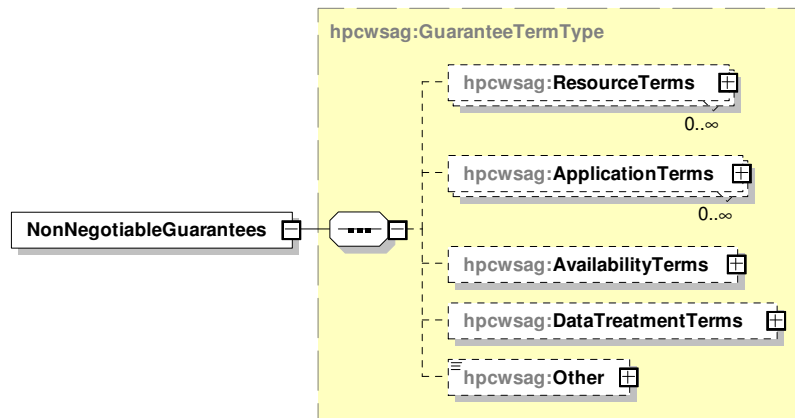


Figure 5.14.: The Non-Negotiable Guarantee Terms Structure

5.6.2.2. hpcwsag:GuaranteeTerms/hpcwsag:NegotiableGuarantees

This element covers all those Terms which are open for negotiation. The negotiation opponent is allowed (depending on the used negotiation protocol) to change the terms and to send the updated document back to the initiator.

The *NegotiableGuarantees* element is structured the same way as the *NonNegotiableGuarantees* part.

5.6.2.3. hpcwsag:NonNegotiableGuarantees/hpcwsag:ResourceTerms

Resource Terms refer to those bits, which define the properties of the used resources. Examining existing approaches, the choice was taken to use definition of Resources Types as in JSDL as this covers most of the requirements as stated in the category "Resourcing" (cf. chapter 2.3.1). Those were namely:

- Operating System (short Term (OS))
- Resource (*Res*)

- Software (*Soft*)
- Versioning (*Vers*)
- Network (*Nw*)
- Storage (*Stor*)
- Access (*Acc*)

Table 5.10 shows in how far the above listed requirements are addressed by the provided JSDL elements.

Besides the Software requirement everything is addressed with the chosen representation. This leads to a need for an extra definition of Software/Application requirements which is represented in the schema as an extra sub guarantee term called *Applications* (cf. 5.6.2.4).

Element	OS	Soft	Vers	Res	Stor	Nw	Acc
CandidateHosts				✓			
FileSystem				✓	✓		
ExclusiveExecution							✓
OperatingSystem	✓		✓				
IndividualCPUSpeedElement				✓			
IndividualCPUTime				✓			
IndividualCPUCount				✓			
IndividualNetworkBandwidth						✓	
IndividualPhysicalMemory				✓			
IndividualVirtualMemory				✓			
IndividualDiskSpace					✓		
TotalCPUTime				✓			
TotalCPUCount				✓			
TotalVirtualMemory				✓			
TotalDiskSpace				✓			
TotalResourceCount				✓			

Table 5.10.: JSDL Resource Types addressing the Resource Requirements of the HPC Schema

An overview of the structure of the Resource Terms can be found in Figure 5.15. The descriptions of the single elements of the Resource Terms (as taken from JSDL) are listed in Table 5.11.



Figure 5.15.: JSDL representing the Resource Terms

Element	Spec	Description
CandidateHosts	jsdl	This element is a complex type specifying the set of named hosts which may be selected for running the job.
FileSystem	jsdl	This element describes a file system that is required by the job.
ExclusiveExecution	jsdl	This is a boolean that designates whether the job must have exclusive access to the resources allocated to it by the consuming system.
OperatingSystem	jsdl	This is a complex type that defines the operating system required by the job.
CPUArchitecture	jsdl	This element is a string specifying the CPU architecture required by the job in the execution environment.
IndividualCPUSpeedElement	jsdl	This element is a range value specifying the speed of each CPU required by the job in the execution environment.
IndividualCPUTime	jsdl	This element is a range value specifying the total number of CPU seconds required on each resource to execute the job.
IndividualCPUCount	jsdl	This element is a range value specifying the number of CPUs for each of the resources to be allocated to the job submission.
IndividualNetworkBandwidth	jsdl	This element is a range value specifying the bandwidth requirements of each individual resource.
IndividualPhysicalMemory	jsdl	This element is a range value specifying the amount of physical memory required on each individual resource.
IndividualVirtualMemory	jsdl	This element is a range value specifying the required amount of virtual memory for each of the resources to be allocated for this job submission.
IndividualDiskSpace	jsdl	This is a range value that describes the required amount of disk space for each resource allocated to the job.

Element	Spec	Description
TotalCPUTime	jsdl	This element is a range value specifying total number of CPU seconds required, across all CPUs used to execute the job.
TotalCPUCount	jsdl	This element is a range value specifying the bandwidth requirements of each individual resource.
TotalPhysicalMemory	jsdl	This element is a range value specifying the required amount of physical memory for the entire job across all resources.
TotalVirtualMemory	jsdl	This element is a range value specifying the required total amount of virtual memory for the job submission.
TotalDiskSpace	jsdl	This is a range value that describes the required total amount of disk space that should be allocated to the job.
TotalResourceCount	jsdl	This element is a range value specifying the total number of resources required by the job.
Other	jsdl	This is a place holder for further extensions on resource terms, which are not yet addressed by the JSDL Resource Terms.
IndividualNetworkBandwidth	jsdl	This element is a range value specifying the bandwidth requirements of each individual resource.

Table 5.11.: Elements of the ResourceTerms

5.6.2.4. hpcwsag:NonNegotiableGuarantees/hpcwsag:ApplicationTerms

This element describes the applications and their properties. Based on Requirements R8 ("Definition of installed software") and R9 ("Definition of versions") requirements at least the software/application name (as unique identifier) and the version number have to be provided.

As well as for the *Resource Terms*, JSDL is also the most promising candidate for the definition of these terms.

The structure of the Application Terms is shown in Figure 5.16, its elements are listed in Table 5.12.

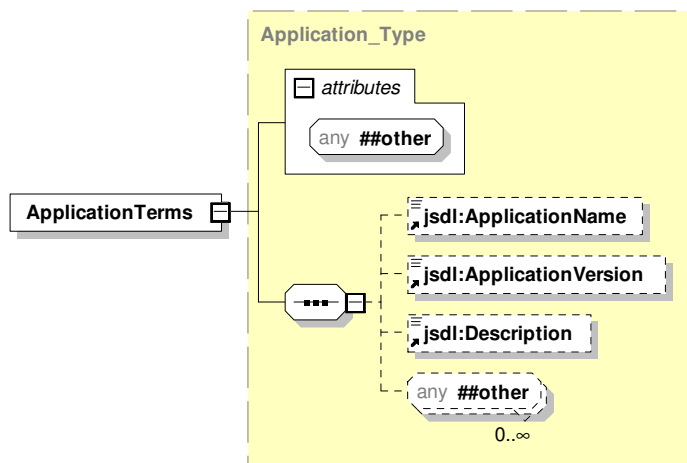


Figure 5.16.: JSDL representing the Application Terms

Element	Spec	Description
ApplicationName	jSDL	This element is a string that defines the name of the application and is used to identify the application independent of the location of its executable on a host or system.
ApplicationVersion	jSDL	This element is a string that defines the version of the application to execute.
Description	jSDL	This element provides descriptive, human readable, information about its containing complex element.
Any	jSDL	Any other content MAY be added here to extend the number of application terms.

Table 5.12.: Elements of the ApplicationTerms Structure

5.6.2.5. hpcwsag:NonNegotiableGuarantees/hpcwsag:AvailabilityTerms

The *Availability Terms* describe all *Service* or *Guarantee Term* related issues in terms of timely access properties as well (if requested) of the number of accesses.

As requirements for *Service Availability* the following items were worked out in chapter 2:

- Definition of start and/or maximal runtime (short term *S/E*) as well as of the maximum runtime (*Max*) of the service [as defined in R1, R15]
- Accessibility(*Acs*) time of the Service (duration of availability) [R2,R3]
- Definition of maximal and/or minimal acceptable response time (*RT*) [R17]
- Definition of the availability rate (*AR*) of the service [R4]
- Limitation of the invocation number(*IN*) of the service [R21]

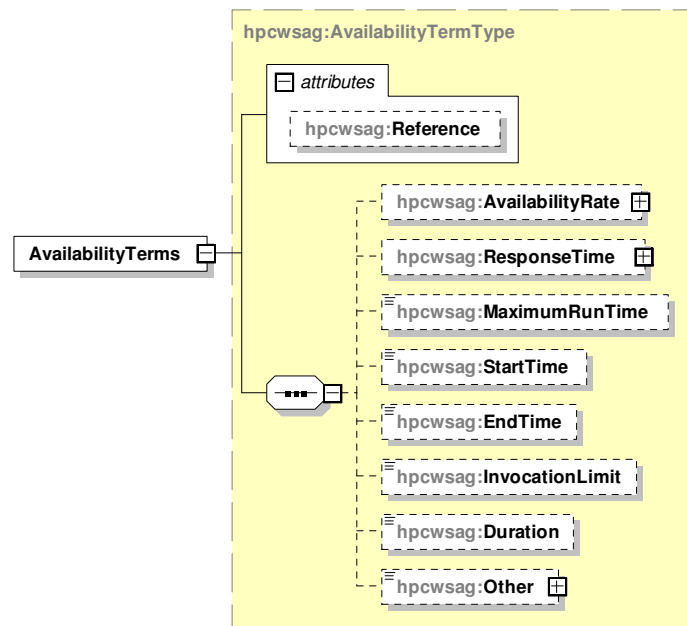


Figure 5.17.: Structure of the Availability Terms

On base of these items, the structure of the Availability Terms was derived as shown in Figure 5.17. In the following the single terms (as shown in Figure 5.17) are presented and described in Table 5.13, after that Table 5.16 shows in how far the single elements address the listed requirements.

Element	Spec	Description
AvailabilityRate	hpcwsag	Defines a minimal percentage of availability for the agreed service.
ResponseTime	hpcwsag	This element defines a response time range which has to be met by the service.
MaximumRunTime	hpcwsag	This element defines the maximal acceptable run time of the service as requested from the customer.
StartTime	hpcwsag	This element - when present - defines an adapted start time of the validity of the referred term or service.
EndTime	hpcwsag	This element - when present - defines an adapted end time of the validity of the referred term or service.
InvocationLimit	hpcwsag	If present, this element defines the maximal number of invocations of an agreed service by its customer.
Duration	hpcwsag	With the help of this element, users can define the duration of the availability of the service.
Other	hpcwsag	This is a place holder for further extensions on availability terms, which are not yet addressed.

Table 5.13.: Elements of the AvailabilityTerms Structure

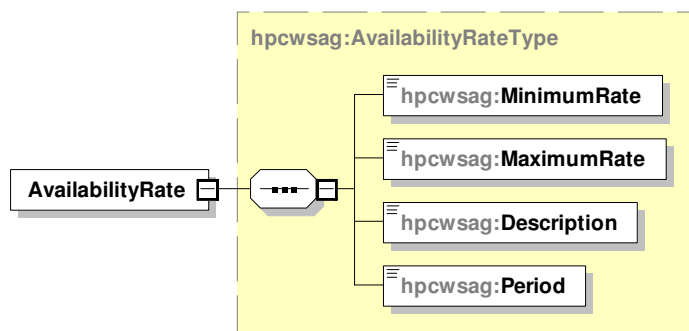


Figure 5.18.: Structure of the AvailabilityRate Element

As previously seen, the AvailabilityTerms contain a description of the Availability Rate of the service. This sub element is shown in Figure 5.18 and its elements are described in Table 5.14.

Element	Spec	Description
MinimumRate	hpcwsag	This element defines the minimal value of the rate.
MaximumRate	hpcwsag	This element defines the maximal value of the rate.
Description	hpcwsag	This is an optional element which allows for giving an additional description.
Period	hpcwsag	This element allows for defining the concrete period, when the rate has to be met.

Table 5.14.: Elements of the AvailabilityRate Structure

As well as the AvailabilityRate, the ResponseTime also has a substructure (cf. Figure 5.19). The elements of the ResponseTime element are listed in Table 5.15.

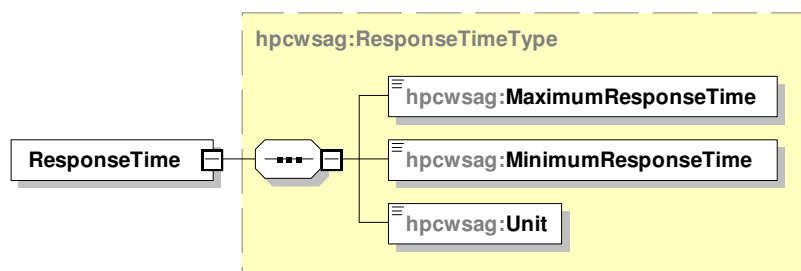


Figure 5.19.: Structure of the ResponseTime Element Terms.

Element	Spec	Description
MinimumResponseTime	hpcwsag	This element defines the minimal value of response time from the service.
Unit	hpcwsag	This element defines the unit of the response time values (e.g. ms).
MaximumResponseTime	hpcwsag	This element defines the highest value of response time from the service.

Table 5.15.: Elements of the ResponseTime Structure

Element	S/E	Max	Acs	RT	AR	IN
AvailabilityRate					✓	
ResponseTime				✓		
MaximumRunTime		✓				
StartTime	✓					
EndTime	✓					
InvocationLimit						✓
Duration			✓			

Table 5.16.: Addressing the Availability Requirements with the Schema Elements

5.6.2.6. hpcwsag:NonNegotiableGuarantees/hpcwsag:DataTreatmentTerms

Ensuring the correct handling of data (input, output or intermediate) is often a requirement of customers to their service providers. To enable the concrete definition of the data treatment, the terms for data handling were designed on base of the requirements listed in chapter 2.3. By that, it offers the means to define the different subcategories of data treatment, namely *Data Access* (as requested by Requirement R14), *Data Backup* (R22), *Data Staging*(R23) and *Data Storage*(R24). Additionally, the schema also provides a place holder for further term definition (as can be seen in Figure 5.20).

Table 5.17 covers the descriptions of the DataTreatmentTerms elements.

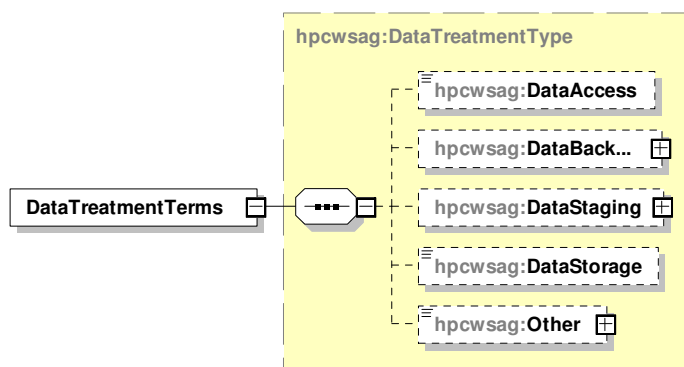


Figure 5.20.: Structure of the Data Treatment Terms

Element	Spec	Description
DataAccess	hpcwsag	This element is intended to allow the definition of the access policy. Value range here is Unlimited, Special Group, Exclusive.
Confidentiality	hpcwsag	This element defines the level of confidentiality of the data.
DataBackUp	hpcwsag	This element provides the means to define the way the data (intermediate and result data) should be backed up during the runtime of the service.
DataStaging	hpcwsag	This element can be used to define files, which should be moved to and from the execution host. Its sub elements are inherited from the JSDL specification.
DataStorage	hpcwsag	This element can be used to define the way the data is stored. It can address either a <i>shared</i> or on an <i>exclusive</i> storage.
Other	hpcwsag	This is a placeholder for further extensions on data treatment terms, which are not yet addressed.

Table 5.17.: Elements of the DataTreatmentTerms Structure

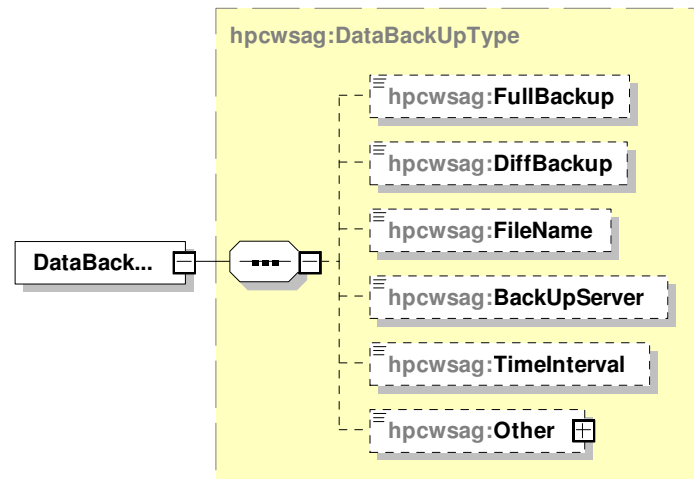


Figure 5.21.: Structure of the DataBackUp Element

Element	Spec	Description
FullBackup	hpcwsag	This element - when present - states that a backup should be performed fully for the whole datasets.
DiffBackup	hpcwsag	This element - when present - states that a backup should be performed only for changed datasets.
FileName	hpcwsag	This element, states the name of the file which should be backed up.
TimeInterval	hpcwsag	This element specifies the interval of the backups.
Other	hpcwsag	This is a placeholder for further extensions on data backup terms, which are not yet addressed.

Table 5.18.: Elements of the DataBackUp Structure

According to Figure 5.20 there are several elements of the DataTreatmentTerms which consist of sub elements, further detailing the service descriptions. Therefore the details of the DataBackUp element can be found in Figure 5.21 and the sub element descriptions in Table 5.18. Similarly the DataStaging Terms will be described. Figure 5.22 shows the structure of the DataStaging Term. The single elements of the DataStaging element are inherited from JSDL and their definition can be found in Table 5.19.

Element	Spec	Description
FileSystemName	jsdl	This element defines the name of the FileSystem.
CreationFlag	jsdl	This element defines whether the staged file should append or overwrite an existing file.
DeleteOnTermination	jsdl	This element can be used to define that the file is deleted when the job is terminated.
Source	jsdl	This element contains the name of the source where the data can be get from.
Target	jsdl	This element contains the name of the target of the data movement.

Table 5.19.: Elements of the DataStaging Structure

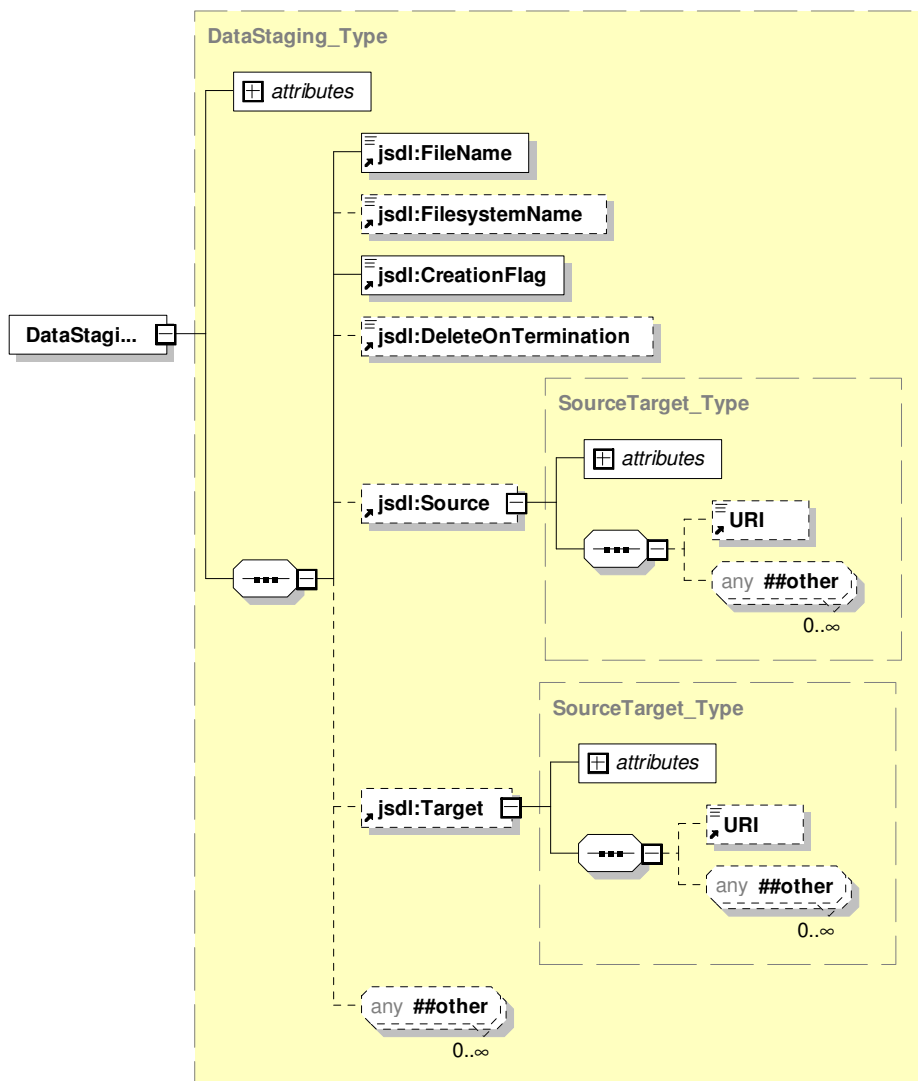


Figure 5.22.: Structure of the DataStaging Element

5.6.2.7. hpcwsag:NonNegotiableGuarantees/hpcwsag:OtherTerms

The *OtherTerms* element is the place for all the terms which are not covered by the *Availability*, *Application*, *Resource* or *Data Treatment Terms*.

5.6.3. hpcwsag:GuaranteeTerms - The Agreement-Document

With respect to the PreSLA-Document, the Agreement does not contain the split of Guarantee Terms into negotiable and non-negotiable parts. Therefore the *GuaranteeTerms* of the Agreement is build up as shown in Figure 5.23. All definition of

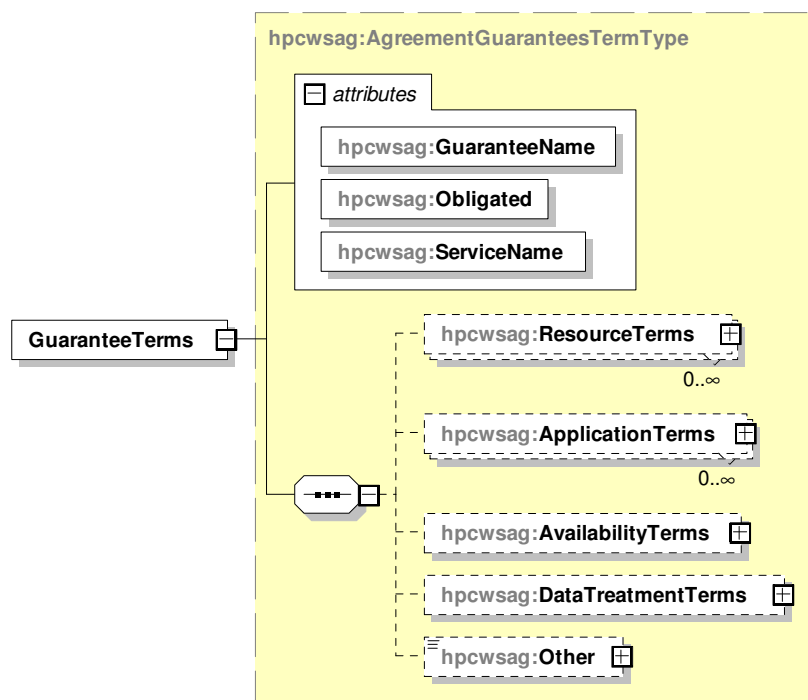


Figure 5.23.: Structure of the Agreement Guarantees

elements are the same as for the PreSLA-Document.

5.6.4. hpcwsag:BusinessTerms

A business relationship between a customer and a service provider normally does not always build upon a high level of trust between both parties. And even if both want to assure themselves to get a compensation in case something goes wrong or to gain a reward (if agreed) when better performing than planned.

This element aims to map the set of requirements on Business Handling as presented in chapter 2.3.3. After a more detailed analysis, the choice was taken to use the *BusinessValueListType* of WS-Agreement, as it provides the means to specify the *Importance*, *Penalties*, *Rewards* which covers already Requirements R6, R25 and R27.

Additionally it offers placeholder to define *Preference* and *CustomBusinessValues* which may be additionally needed in so far not foreseen use cases.

In addition to the inherited terms from WS-Agreement, Requirement R26 ("Support of different pricing models") is addressed by an extra element, which allows for the definition of *Variable* and *Fixed Prices*.

Figure 5.24 shows the according structure of the *Business Terms*.

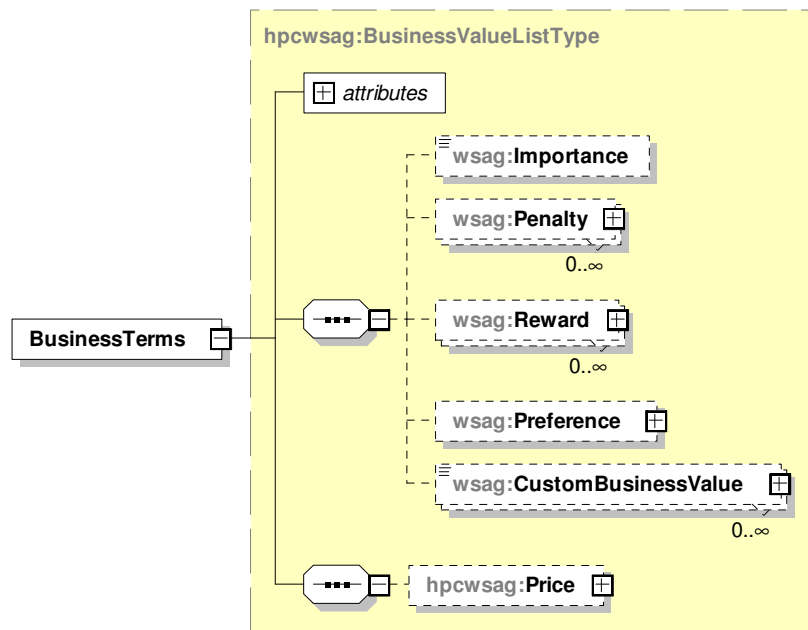


Figure 5.24.: Structure of the Business Terms

The single elements which build the Business Terms are presented in Table 5.20.

Element	Spec	Description
Importance	hpcwsag	This element - when present - expresses relative importance of meeting an objective.
Penalty	hpcwsag	This element - when present - expresses the penalty to be assessed for not meeting an objective.
Reward	hpcwsag	This element - when present - expresses reward to be assessed for meeting an objective.
Preference	hpcwsag	This element specifies a list of fine-granularity business values for different alternatives, where each alternative refers to a Service Description Term and its associated utility.

Element	Spec	Description
CustomBusinessValue	hpcwsag	Zero or more domain specific customized business values can be defined.
@Name	hpcwsag	This is an unique name of the Business Term itself.
@GuaranteeName	hpcwsag	This element refers to the Guarantee Term(s) these Business Terms refer to.
Price	hpcwsag	This element defines the type of the price and the respective values. It allows for definition of fixed prices as well as variable prices.

Table 5.20.: Elements of the Business Terms Structure

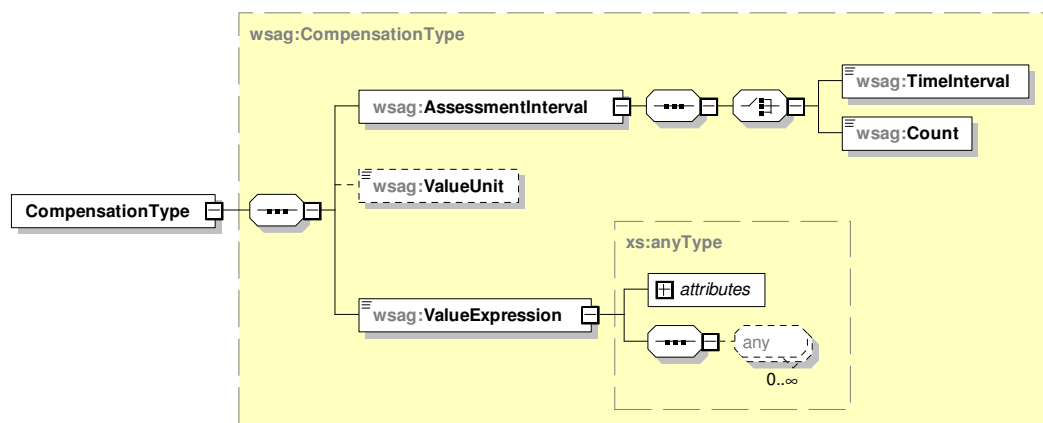


Figure 5.25.: Structure of the CompensationType

Both, *Penalty* and *Reward* follow the structure of *CompensationType* as defined in WS-Agreement (cf. Figure 5.25). The definition of the single terms can be found in Table 5.21. The single elements of the *AssessmentInterval* type are defined as in Table 5.22.

The *Price* element is split up as can be seen in Figure 5.26. Its description can be found in Table 5.24. As there are different pricing models (fixed and variable), they are as well further defined in sub elements. The structure of the fixed price terms is shown in Figure 5.27 with descriptions in Table 5.25.

Element	Spec	Description
AssessmentInterval	wsag	This element defines the interval over which a penalty is assessed.
ValueUnit	wsag	This element defines the unit for assessing penalty, such as USD. This is an optional element since in some cases a default unit MAY be assumed. (From WSAG)
ValueExpression	wsag	This element defines the assessment amount, which can be an integer, a float or an arbitrary domain-specific expression.

Table 5.21.: Elements of the CompensationType Structure

Element	Spec	Description
TimeInterval	wsag	This element - when defined - defines the assessment interval as a duration.(From WSAG)
Count	hpcwsag	This element - when present - defines the assessment interval as a service specific count, such as number of invocations. (From WSAG)

Table 5.22.: Elements of the AssessmentInterval Structure

Element	Spec	Description
ServiceTermReference	wsag	This element can appear multiple times. Each Service Term Reference refers to a service term and represents an alternative way of achieving the associated service level objective. The corresponding, utility (specified below) specifies the utility gained by achieving this objective.(From WSAG)
Utility	wsag	This element can appear multiple times, one corresponding to each Service Term Reference. (From WSAG)

Table 5.23.: Elements of the Penalty Structure

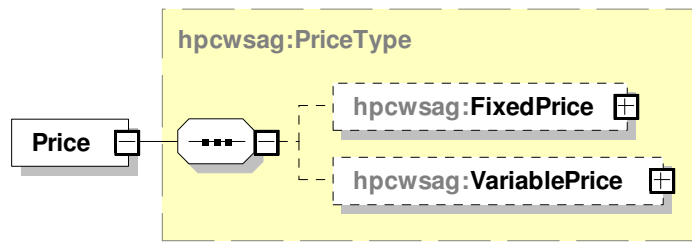


Figure 5.26.: Structure of the Price Terms

Element	Spec	Description
FixedPrice	hpcwsag	This element provides the means to define a fixed price for either a complete service or per sub-units of the service.
VariablePrice	hpcwsag	This element provides the means to define a price range for either a complete service or per sub-units of the service.

Table 5.24.: Elements of the Price Structure

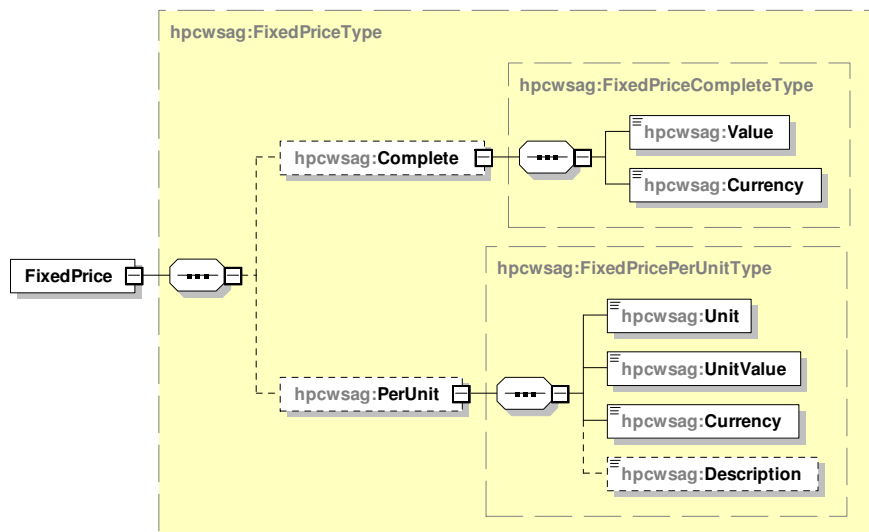


Figure 5.27.: Structure of the Fixed Price Terms

Element	Spec	Description
Complete	hpcwsag	This element defines the fixed price for the whole service.
PerUnit	hpcwsag	This element defines the price per unit for the whole service.

Table 5.25.: Elements of the FixedPrice Structure

Furthermore the fixed price *Complete* element is further refined as detailed in Table 5.26. Complementary, Table 5.27 defines the *PerUnit* element.

Element	Spec	Description
Value	hpcwsag	This element defines the value of the fixed price for the whole service.
Currency	hpcwsag	This element defines the currency of the fixed price for the whole service.

Table 5.26.: Elements of the Complete Structure for a Fixed Price

Element	Spec	Description
Unit	hpcwsag	This element defines the unit which the price belongs to.
UnitValue	hpcwsag	This element defines the value of the price per unit.
Currency	hpcwsag	This element defines the currency of the value of the price per unit.
Description	hpcwsag	This element allows for adding additional descriptions for the price per unit.

Table 5.27.: Elements of the Per Unit Structure for a Fixed Price

In addition to the fixed price element, the elements of the variable price terms are defined as shown in Figure 5.28. Similarly to the variable price terms, it consists of the complete/per unit price definitions (cf. Table 5.28). These elements consist again

of sub elements as listed in Table 5.29 and Table 5.30.

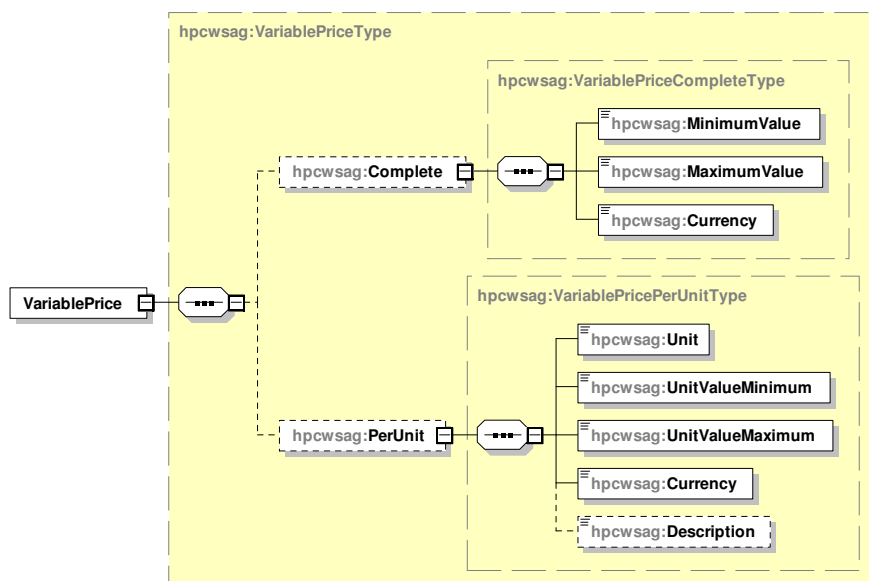


Figure 5.28.: Structure of the Variable Price Terms

Element	Spec	Description
Complete	hpcwsag	This element defines the variable price for the whole service.
PerUnit	hpcwsag	This element defines the price range per unit for the service.

Table 5.28.: Elements of the Variable Price Structure

Element	Spec	Description
MinimumValue	hpcwsag	This element defines the minimum value of the price for the whole service.
MaximumValue	hpcwsag	This element defines the maximum value of the price for the whole service.
Currency	hpcwsag	This element defines the currency of the price range for the whole service.

Table 5.29.: Elements of the Complete Structure for a Variable Price

Element	Spec	Description
Unit	hpcwsag	This element defines the unit which the price belongs to.
UnitValueMinimum	hpcwsag	This element defines the minimum value of the price per unit.
UnitValueMaximum	hpcwsag	This element defines the maximum value of the price per unit.
Currency	hpcwsag	This element defines the currency of the value range of the price per unit.
Description	hpcwsag	This element allows for adding additional descriptions for the price per unit.

Table 5.30.: Elements of the Per Unit Structure for a Variable Price

5.7. Exclusion Terms

As discussed in the introduction, the *Exclusion Terms* element was introduced to allow customers and mainly service providers to define those events, which they cannot control themselves. This acts as an assurance against recourse claims resulting from one or more of those events.

The best way to define these terms is a simple string, which is assumed to be a lawyer provided text covering all the exclusion aspects.

5.8. Creation Constraints

As already pointed out in Section 5.3, the *CreationConstraints* element is only needed for the definition the Offers, Templates, Tender and Requests but not for the SLA Agreement or the preventive SLA.

This element contains information about the time constraints for finding an agreement (e.g. until when an Offer or Request is valid) and that kind of information that is needed to find a basic common context, as described in 4.4.

The design of this element was based on rather implicit requirements from the use

Element	Spec	Description
ExpirationTime	hpcwsag	This element specifies the time at which this pre-SLA document (not a signed SLA) is no longer valid.
CreationContext	hpcwsag	This element allows for the definition of different properties levels which shall support the choice of the used protocols and in longer term enhance the process of finding an agreement.

Table 5.31.: Elements of the Creation Constraints Structure

cases. Besides setting a expiration time for the validity of the Pre-SLA document, the means have to be given to define an overall time limit until the agreement has been established.

To target the field of Urgent Computing and beyond, the urgency should be also be definable. For the customer to transmit the urgency of his needs to the service provider but as well the other way around. When defining the urgency in a SLA Template, a pre-selection would be possible for these cases which would simplify the negotiation process.

In addition to this, the quality of the contents of a PreSLA document should be definable as well as the reliability of the provided service.

Intentionally not all of the elements of the Creation Constraints have to be stated, but they offer a high value in minimization of the document exchange by detailing the needs of the involved parties.

The element is structured as shown in Figure 5.29.

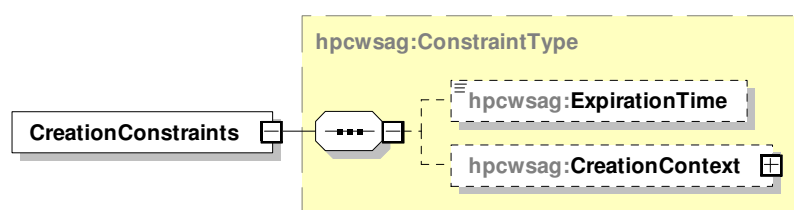


Figure 5.29.: Structure of the CreationConstraints

The elements of the CreationConstraints element are defined in Table 5.31.

The Creation Context covers a set of elements which are shown in Figure 5.30. Their definitions are given in Table 5.32.

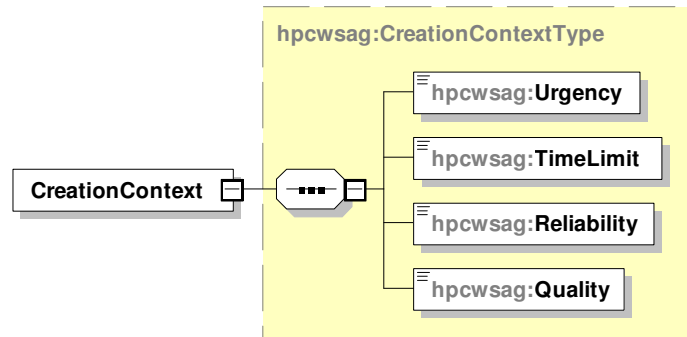


Figure 5.30.: Structure of the CreationContext Element

Element	Spec	Description
Urgency	hpcwsag	This element - when present - defines the <i>Urgency</i> of coming to an agreement. Its value range is low, medium, high. If there is no value assigned, the system is free to decide.
TimeLimit	hpcwsag	This element - when present - gives a clear hint as to what kinds of discover/negotiation are acceptable. The time limit is defined as a deadline time instance.
Reliability	hpcwsag	This element contains information about the reliability of the needed business partner. Its values range for reliability is low, medium, high and if there is no value assigned, the system is free to decide.
Quality	hpcwsag	This element defines the targeted quality of the service.

Table 5.32.: Elements of the CreationContext Structure

Chapter 6.

Application And Verification Of The Concepts

The presented solutions within this thesis (architecture and schema) were developed close to the real HPC business and therefore are usable within the HPC domain. This means, that not everything will be realizable with today's state of the art technologies. However the content of this work will give advices of the direction of future research topics.

This chapter will evaluate in how far the solutions support the defined use cases with respect to the capabilities of today's HPC centres and their resource management systems and also provide a short analysis of existing products/research results which could be used to realize the necessary missing capabilities.

After a short analysis, in how far the offering of service can be enhanced with the provided solutions, the schema will be applied to the use cases to show in how far customers and service providers may define the different SLA-related documents (e.g. SLA Templates, Offers, Requests). Furthermore a dedicated section will examine the *Guarantee Terms* in more detail to see in how far those resource management systems which are currently used in HPC centres may support the single elements. Then the SLA lifecycle will be revisited, and existing solutions will be discussed which may be applied to realize the SLA Management until finally the requirements to future resource management systems will be summarized, based on the analysed problems with today's systems.

6.1. Realizing the scenarios

Generally the change from the WS-Agreement Agreement/Template structure to the Agreement/PreSLA structure provides better support for both, customers and service providers for the management of Service Level Agreements. The current state of the

art in SLA languages provides no solution, which could be easily used in the HPC domain. This led to the redefinition of parts of concepts from solutions/specifications such as WSLA, WS-Agreement or JSDL, all having the scenarios in mind, which were presented in chapter 2.

What is new in the presented approach is threefold:

- A domain specific extension of common concepts up to an easy to handle SLA schema for High Performance Computing Services
- Addition of new concepts to allow for a targeted SLA/PreSLA creation for all involved parties
- Evolution whilst keeping as close as possible to existing standards

The latter addresses especially the addition of the *Context* elements into the *Creation-Constraints*. On one hand this allows customers to provide extended information about their needs and give an impression of their willingness for open or focused negotiation. On the other hand, service providers can define focused SLA Offers/Templates on the respective category of users (e.g. for Urgent Computing purposes). The following subsections will show how different PreSLA-Documents may be defined for the respective use cases and what kind of negotiation/discovery approach may be used.

Intentionally only those xml parts will be shown which are of interest and give added value for the understanding of the capabilities of the schema.

6.1.1. Instantiating the SLA documents for the Medical Support Scenario

The Medical Support Scenario is a typical example for the domain of "Urgent Computing". With the approach presented in this work, HPC service providers will be able to define special SLA Templates for urgent requests, which are very concrete and allow for quick negotiation.

In this use case, the assumption is that several service providers are specialized on the provisioning of "Urgent Computing" services and have prepared and published the respective SLA Templates (cf. 6.1.1.1). This is represented by setting the *Urgency* element to the respective value.

As the Goodwill Hospital needs the HemeLB application to be available urgently, they decide to discover matching service providers by a simple Yellow Pages search. Having found a candidate service provider, an SLA Request is created, based on the Template information and the desires of the customer which then (in this case) leads to an SLA Offer and finally an SLA.

6.1.1.1. SLA Template

Service providers creating SLA Templates for this kind of urgent requests, can benefit from the *CreationConstraints* element as provided by the schema. Generally, a Provider will not define all parameters of his offered service into detail, preferably, higher level terms are published which leave a certain degree on flexibility of the mapping down to the realization on infrastructure level.

In our case, the service provider may design SLA Templates for the HemeLB service as follows.

```
<hpcwsag:PreSLADocument hpcwsag:DocType="Template"
  hpcwsag:Id="UrgentHemeLB001" , . . . .
. . .
```

Listing 6.1: "Defining a PreSLA Document as Template"

This defines the document type as (SLA) Template.

Name

As in the schema definition (cf. Section 5.4), the *Name* can be optionally given to the agreement for provisioning of a human understandable identifier. In our case this may look like:

```
. . .
<wsag:Name>UrgentHemeLB</wsag:Name>
. . .
```

Listing 6.2: "The Name Definition in the MS SLA Template"

Terms

The *Terms* section of the SLA Template is used to provide more information about the *Service Terms* as well as of *Guarantee Terms* and *Business Terms*. In our example, the service provider defines his *Service Description* in the *Service Terms* element (not explicitly shown in the listing). The listing shows the categorization of guarantees in non-negotiable and negotiable ones. Within them, the HPC Provider defines the Application and its version as static terms, whereas the availability rate is up for discussion.

```
...
<hpcwsag:ServiceTerms>
  <hpcwsag:ServiceDescription wsag:Name="UrgentHemeLB
  Description" wsag:ServiceName="HemeLB">
    ...
  </hpcwsag:ServiceDescription>
  <hpcwsag:ServiceReference wsag:Name="UrgentHemeLBReference"
  wsag:ServiceName="UrgentHemeLB">
    <wsa:EndpointReference
      xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>
        http://goodwill.castlerock.org/hemelb
      </wsa:Address>
    </wsa:EndpointReference>
  </hpcwsag:ServiceReference>
</hpcwsag:ServiceTerms>
<hpcwsag:GuaranteeTerms hpcwsag:Obligated="ServiceProvider"
hpcwsag:GuaranteeName="UrgentHemeLB"
hpcwsag:ServiceName="UrgentHemeLB">
  <hpcwsag:NonNegotiableGuarantees>
  <hpcwsag:ResourceTerms>
    <jSDL:ExclusiveExecution>true</jSDL:ExclusiveExecution>
  </hpcwsag:ResourceTerms>
  <hpcwsag:ApplicationTerms>
    <jSDL:ApplicationName>HemeLB</jSDL:ApplicationName>
    <jSDL:ApplicationVersion>2.0</jSDL:ApplicationVersion>
  </hpcwsag:ApplicationTerms>
  </hpcwsag:NonNegotiableGuarantees>
  <hpcwsag:NegotiableGuarantees>
  <hpcwsag:AvailabilityTerms>
    <hpcwsag:AvailabilityRate>
      <hpcwsag:MinimumRate>95.0</hpcwsag:MinimumRate>
      <hpcwsag:MaximumRate>99.9</hpcwsag:MaximumRate>
      <hpcwsag:Period>PT12H</hpcwsag:Period>
    </hpcwsag:AvailabilityRate>
  </hpcwsag:AvailabilityTerms>
  </hpcwsag:NegotiableGuarantees>
</hpcwsag:GuaranteeTerms>
<hpcwsag:BusinessTerms hpcwsag:GuaranteeName="UrgentHemeLB"
hpcwsag:Name="UrgentHemeLBBT">
  <hpcwsag:Price>
```

```

    <hpcwsag : FixedPrice >
      <hpcwsag : Complete >
        <hpcwsag : Value >200</hpcwsag : Value >
        <hpcwsag : Currency >Euro</hpcwsag : Currency >
      </hpcwsag : Complete >
    </hpcwsag : FixedPrice >
  </hpcwsag : Price >
</hpcwsag : BusinessTerms >
...

```

Listing 6.3: "Terms in the SLA Template"

In this example, the *UrgentHemeLB* service is offered for an exclusive execution and with an availability rate between 95 and 99.9 percent and over a duration of 12 hours. The price for this offer is 200¹ Euro (as a complete package).

ExclusionTerms

As the application field of Urgent Computing is a highly critical area, the service provider will write down a set of exclusion terms, such as "caused by factors outside of our reasonable control, including any force majeure event or Internet access or related problems beyond the demarcation point of the service provider", as currently used by Amazon S3².

PreContext

The next bit is the *PreContext* in which the service provider can specify his details.

```

...
<hpcwsag : PreContext >
  <hpcwsag : ServiceProvider >... </hpcwsag : ServiceProvider >
</hpcwsag : PreContext >
...

```

Listing 6.4: "Context in the MS SLA Template"

¹This price is clearly too low for doing such a simulation as of today. However it was chosen with the assumption that in future a simulation like this cannot cost more as a radiograph if used in clinical practice

²Amazon Simple Storage Service, <http://aws.amazon.com/s3-sla/>

CreationConstraints

To give interested customers a first hint of the properties of the offered service, the service provider defines in its templates the element *Urgency* as high.

```
...
<hpcwsag:CreationConstraints>
  <hpcwsag:CreationContext>
    <hpcwsag:Urgency>High</hpcwsag:Urgency>
  </hpcwsag:CreationContext>
</hpcwsag:CreationConstraints>
...
```

Listing 6.5: "Creation Constraints as in the SLA Template"

6.1.1.2. SLA Request

After the Goodwill Hospital has performed a Yellow Pages Search, they have now a set of potential candidates for the HemeLB service. To get into contact with the service provider and to start the negotiation process, an SLA Request is created, stating all the desired parameters of the service.

Name

The Name tag of the Request is taken one by one from the Template of the service provider.

Terms

Similar to the Name tag, also the Service Terms are taken as defined by the service provider. Additionally, the Guarantee and Business Terms are taken and partially adapted, where necessary.

```
...
<hpcwsag:GuaranteeTerms hpcwsag:Obligated="ServiceProvider"
  hpcwsag:GuaranteeName="UrgentHemeLB"
  hpcwsag:ServiceName="HemeLB">
  <hpcwsag:NonNegotiableGuarantees>
    <hpcwsag:ApplicationTerms>
```

```

<jsdl:ApplicationName>HemeLB</jsdl:ApplicationName>
<jsdl:ApplicationVersion>
  2.0
</jsdl:ApplicationVersion>
</hpcwsag:ApplicationTerms>
<hpcwsag:AvailabilityTerms>
  <hpcwsag:AvailabilityRate>
    <hpcwsag:MinimumRate>99.9</hpcwsag:MinimumRate>
    <hpcwsag:MaximumRate>99.9</hpcwsag:MaximumRate>
    <hpcwsag:Period>PT7H</hpcwsag:Period>
  </hpcwsag:AvailabilityRate>
  <hpcwsag:MaximumRunTime>PT4H</hpcwsag:MaximumRunTime>
  <hpcwsag:StartTime>
    2009-10-18T12:00:00Z
  </hpcwsag:StartTime>
  <hpcwsag:EndTime>
    2009-10-18T16:00:00Z
  </hpcwsag:EndTime>
</hpcwsag:AvailabilityTerms>
</hpcwsag:NonNegotiableGuarantees>
</hpcwsag:GuaranteeTerms>
<hpcwsag:BusinessTerms hpcwsag:GuaranteeName="UrgentHemeLB"
  hpcwsag:Name="UrgentHemeLBBT">
  <wsag:Penalty>
    <wsag:AssessmentInterval>
      <wsag:TimeInterval>PT7H</wsag:TimeInterval>
    </wsag:AssessmentInterval>
    <wsag:ValueExpression>1000 Euro</wsag:ValueExpression>
  </wsag:Penalty>
  <hpcwsag:Price>
    <hpcwsag:FixedPrice>
      <hpcwsag:Complete>
        <hpcwsag:Value>200</hpcwsag:Value>
        <hpcwsag:Currency>Euro</hpcwsag:Currency>
      </hpcwsag:Complete>
    </hpcwsag:FixedPrice>
  </hpcwsag:Price>
</hpcwsag:BusinessTerms>
...

```

Listing 6.6: "Terms as in the SLA Request"

Here, the Goodwill Hospital adapts the Availability Rate properties by stating that they want a guarantee of 99.9 percent availability over a period of 7 hours (the approx execution time of service and surgery). Additionally the maximum acceptable runtime of the service is defined as 4 hours as well as a concrete Start is given. All these are non-negotiable terms.

To assure itself, the customer also defines a penalty that he wants the service provider to agree on, in case within the 7 hours, the SLA Terms are violated.

ExclusionTerms

The Hospital defines no Exclusion Terms.

Context

Listing 6.7 shows the definition of the *DocumentBase* for this SLA Request.

```
...
<hpcwsag:PreContext>
  <hpcwsag:DocumentBase>
    <hpcwsag:Id>UrgentHemeLB001</hpcwsag:Id>
    <hpcwsag:Name>UrgentHemeLB</hpcwsag:Name>
    <hpcwsag:Type>Template</hpcwsag:Type>
  </hpcwsag:DocumentBase>
  <hpcwsag:Customer>... </hpcwsag:Customer>
  <hpcwsag:AgreementInitiator>
    <hpcwsag:PartyRole>Customer</hpcwsag:PartyRole>
    <hpcwsag:PartyID>
      http:\\goodwill.castlerock.com\\Negotiator
    </hpcwsag:PartyID>
  </hpcwsag:AgreementInitiator>
  <hpcwsag:AgreementResponder>
    <hpcwsag:PartyRole>ServiceProvider</hpcwsag:PartyRole>
    <hpcwsag:PartyID>
      http:\\hpcc.castlerock.com\\Negotiator
    </hpcwsag:PartyID>
  </hpcwsag:AgreementResponder>
</hpcwsag:PreContext>
...
```

Listing 6.7: "Context of the SLA Request"

By that, the service provider can easily check the respective SLA Template and derive the changes proposed by the customer. Additionally the customer here defines that he is the initiator of the Agreement and the service provider will be the Responder.

CreationConstraints

By setting the *CreationConstraints* the Hospital clearly defines this request as very urgent and sets a time limit of 30 minutes. This states that after passing this limit, the request is no more valid.

Furthermore high reliability is requested.

```
...
<hpcwsag:CreationConstraints>
  <hpcwsag:CreationContext>
    <hpcwsag:Urgency>High</hpcwsag:Urgency>
    <hpcwsag:TimeLimit>P30M</hpcwsag:TimeLimit>
    <hpcwsag:Reliability>High</hpcwsag:Reliability>
  </hpcwsag:CreationContext>
</hpcwsag:CreationConstraints>
...
```

Listing 6.8: "Creation Constraints as in the SLA Request"

6.1.1.3. SLA

Assuming that the service provider agreed on the request and finalized an SLA Offer, both will sign it and at the end have a Service Level Agreement established. This is expressed by using the SLA Agreement label and the respective AgreementId.

```
...
<hpcwsag:SLA Agreement
  hpcwsag: AgreementId="UrgentHemeLBCastleRock01"... >
...
```

Listing 6.9: "The MS Agreement Id"

Name

The final SLA contains also the Name of the Agreement.

```
...  
<wsag:Name>UrgentHemeLB01</wsag:Name>  
...
```

Listing 6.10: "The Name Definition of the MS SLA Agreement"

Terms

In this use case, all the terms keep unchanged with respect to the SLA Request.

ExclusionTerms

Finally, the service provider added a lawyer approved text, stating that he is not responsible for problems with the hospitals equipment as well as for a fall out of the network line between the hospital and the HPC Center.

Context

The Context of the Agreement covers the information about the base document, the service provider and customer details, as well as the Validity of this Agreement.

```
...  
<hpcwsag:Context>  
  <hpcwsag:DocumentBase>  
    <hpcwsag:Id>UrgentHemeLB001</hpcwsag:Id>  
    <hpcwsag:Name>UrgentHemeLB</hpcwsag:Name>  
    <hpcwsag:Type>Template</hpcwsag:Type>  
  </hpcwsag:DocumentBase>  
  <hpcwsag:Validity>  
    <wsla:Start>2009-10-18T11:00:00Z</wsla:Start>  
    <wsla:End>2009-10-18T20:00:00Z</wsla:End>  
  </hpcwsag:Validity>  
  <hpcwsag:ServiceProvider>...</hpcwsag:ServiceProvider>  
  <hpcwsag:Customer>...</hpcwsag:Customer>  
</hpcwsag:Context>  
...
```

Listing 6.11: "Context of the Agreement"

This Service Level Agreement is now used for preparation of the service provider System. Monitoring and Evaluation will take place on basis of this SLA until the service is finished.

6.1.2. Instantiating the SLA documents for the Improved Car Design Scenario

In terms of urgency, the Car Design scenario is significantly relaxed. The car manufacturer needs the simulation service, but has time to find fitting service providers in terms of the service properties but also of the price.

Therefore the decision is taken to launch an invitation to tender to get offers from the service providers.

6.1.2.1. Tender

Within the *Tender* the car manufacturer can detail what he wants to be provided. He bases his tender on the PreSLA Schema and labels in adequately.

```
<hpcwsag:PreSLADocument hpcwsag:DocType="Tender"  
  hpcwsag:Id="CCS01" ,... >
```

Listing 6.12: "Defining a PreSLA Document as Tender"

Name

As *Name* the car manufacturer decides on a simple representation of the desired service.

```
<wsag:Name>Car Crash Simulation Service</wsag:Name>
```

Listing 6.13: "The Customer chosen Name of the Service"

Terms

Within the Terms section, the customer defines now all his requirements, from resource specific terms to those defining the handling of the data.

It is of high importance for the car manufacturer to have a stable system to ensure the comparability of the simulation results. For that purpose he defines a set of non-negotiable terms, amongst others the requested applications down to the version

level (here: LS-DYNA R3.43919 and Java jdk1.6.0_7). As the results of the simulation will be highly confidential data sets, he defines this in the *DataTreatmentTerms* ensuring also limited access and an exclusive storage. Furthermore he wants to have the resources exclusively to ensure no disturbances from outsiders, 100 GB of disk space (both non-negotiable) and a guarantee, that the service provider will keep the environment for 10 weeks stable.

The only negotiable term from his side is the availability rate, which should be between 80 and 100 percent in the period of 70 days (10 weeks).

```
<hpcwsag:Terms>
  <hpcwsag:ServiceTerms>... </hpcwsag:ServiceTerms>
  <hpcwsag:GuaranteeTerms
    hpcwsag:Obligated="ServiceProvider"
    hpcwsag:GuaranteeName="CCSGuarantee01"
    hpcwsag:ServiceName="CCSService">
    <hpcwsag:NonNegotiableGuarantees>
      <hpcwsag:ResourceTerms>
        <jsd1:ExclusiveExecution>
          true
        </jsd1:ExclusiveExecution>
        <jsd1:TotalDiskSpace>
          <jsd1:Exact>107374182400</jsd1:Exact>
        </jsd1:TotalDiskSpace>
      </hpcwsag:ResourceTerms>
      <hpcwsag:ApplicationTerms>
        <jsd1:ApplicationName>
          LS-DYNA
        </jsd1:ApplicationName>
        <jsd1:ApplicationVersion>
          R3.43919
        </jsd1:ApplicationVersion>
      </hpcwsag:ApplicationTerms>
      <hpcwsag:ApplicationTerms>
        <jsd1:ApplicationName>Java</jsd1:ApplicationName>
        <jsd1:ApplicationVersion>
          jdk1.6.0_7
        </jsd1:ApplicationVersion>
      </hpcwsag:ApplicationTerms>
      <hpcwsag:DataTreatmentTerms>
        <hpcwsag:DataAccess>Limited</hpcwsag:DataAccess>
```

```

    <hpcwsag:Confidentiality>
      High
    </hpcwsag:Confidentiality>
    <hpcwsag:DataStorage>
      Exclusive
    </hpcwsag:DataStorage>
    <hpcwsag:Other/>
  </hpcwsag:DataTreatmentTerms>
  <hpcwsag:Other>
    Guaranteed Stable Environment for 1 year needed
  </hpcwsag:Other>
</hpcwsag:NonNegotiableGuarantees>
<hpcwsag:NegotiableGuarantees>
  <hpcwsag:AvailabilityTerms>
    <hpcwsag:AvailabilityRate>
      <hpcwsag:MinimumRate>80.0</hpcwsag:MinimumRate>
      <hpcwsag:MaximumRate>100.0</hpcwsag:MaximumRate>
      <hpcwsag:Period>P70D</hpcwsag:Period>
    </hpcwsag:AvailabilityRate>
  </hpcwsag:AvailabilityTerms>
</hpcwsag:NegotiableGuarantees>
</hpcwsag:GuaranteeTerms>
<hpcwsag:BusinessTerms
  hpcwsag:GuaranteeName="CCSGuarantee01"
  hpcwsag:Name="CCSGuarantee01BT">
  <wsag:Penalty>
    <wsag:AssessmentInterval>
      <wsag:TimeInterval>P10D</wsag:TimeInterval>
    </wsag:AssessmentInterval>
    <wsag:ValueExpression>500 Euro</wsag:ValueExpression>
  </wsag:Penalty>
</hpcwsag:BusinessTerms>
</hpcwsag:Terms>
...

```

Listing 6.14: "Terms of the Car Crash Simulation Tender"

To assure that the terms are kept, a penalty agreement is requested which would lead to a payment of 500 Euro if within an interval of 10 Days, the SLA is violated.

PreContext

The Context of the Tender is not different to the ones of the previous use case. In this simple case it contains the information about the customer.

CreationConstraints

The *CreationConstraints* of this Tender are simple. Urgency is Low, as there is not deadline yet approaching and the Time Limit for sending an Offer based on this Tender is 15 Days.

```
...
<hpcwsag:CreationConstraints>
  <hpcwsag:CreationContext>
    <hpcwsag:Urgency>Low</hpcwsag:Urgency>
    <hpcwsag:TimeLimit>P15D</hpcwsag:TimeLimit>
  </hpcwsag:CreationContext>
</hpcwsag:CreationConstraints>
...
```

Listing 6.15: "Creation Constraints as in the CCS Tender"

6.1.2.2. The SLA Offer from a Service Provider

Assuming that the Tender was published, a service provider finding it can create focused SLA Offer(s) on base of the Tender contents. Therefore he uses the Service Terms (mainly the Service Description)

```
<hpcwsag:PreSLADocument hpcwsag:DocType=" Offer "
  hpcwsag:Id="CCSOffer01" ,... >
```

Listing 6.16: "Defining a PreSLA Document as Offer"

Terms

Within the Terms section, the service provider now updates the Service Terms on base of his service and proposes in our case two different Guarantee Terms (together with respective Business Terms).

Both Guarantee Terms are similar with respect to the definition of the disk space, the applications and versions and the data treatment. Also, the guarantee is given for the

stable environment.

Listing 6.17 shows the differences between the Guarantee Terms and the Business Terms.

```

...
<hpcwsag:GuaranteeTerms hpcwsag:Obligated="ServiceProvider"
hpcwsag:GuaranteeName="CCSGuaranteeA"
hpcwsag:ServiceName="CCSService">
  <hpcwsag:NonNegotiableGuarantees>
    <hpcwsag:ResourceTerms>
      <jsdl:ExclusiveExecution>
        true
      </jsdl:ExclusiveExecution>
      <jsdl:OperatingSystem>
        <jsdl:OperatingSystemType>
          <jsdl:OperatingSystemName>
            Super-US
          </jsdl:OperatingSystemName>
        </jsdl:OperatingSystemType>
        <jsdl:OperatingSystemVersion>
          10
        </jsdl:OperatingSystemVersion>
      </jsdl:OperatingSystem>
      <jsdl:CPUArchitecture>
        <jsdl:CPUArchitectureName>
          x86
        </jsdl:CPUArchitectureName>
      </jsdl:CPUArchitecture>
      <jsdl:TotalCPUCount>
        <jsdl:Exact>16</jsdl:Exact>
      </jsdl:TotalCPUCount>
    </hpcwsag:ResourceTerms>
    <hpcwsag:ApplicationTerms>
      <jsdl:ApplicationName>LS-DYNA</jsdl:ApplicationName>
      <jsdl:ApplicationVersion>
        R3.43919
      </jsdl:ApplicationVersion>
    </hpcwsag:ApplicationTerms>
    <hpcwsag:ApplicationTerms>
      <jsdl:ApplicationName>Java</jsdl:ApplicationName>
      <jsdl:ApplicationVersion>

```

```
    jdk1.6.0_7
  </jsdl:ApplicationVersion>
</hpcwsag:ApplicationTerms>
<hpcwsag:AvailabilityTerms>
  <hpcwsag:AvailabilityRate>
    <hpcwsag:MinimumRate>90.0</hpcwsag:MinimumRate>
    <hpcwsag:MaximumRate>90.0</hpcwsag:MaximumRate>
    <hpcwsag:Period>P70D</hpcwsag:Period>
  </hpcwsag:AvailabilityRate>
</hpcwsag:AvailabilityTerms>
<hpcwsag:DataTreatmentTerms>
  <hpcwsag:DataAccess>Limited</hpcwsag:DataAccess>
  <hpcwsag:Confidentiality>
    High
  </hpcwsag:Confidentiality>
  <hpcwsag:DataStorage>Exclusive</hpcwsag:DataStorage>
  <hpcwsag:Other/>
</hpcwsag:DataTreatmentTerms>
<hpcwsag:Other>
  Guaranteed Stable Environment for 1 year
</hpcwsag:Other>
</hpcwsag:NonNegotiableGuarantees>
</hpcwsag:GuaranteeTerms>
<hpcwsag:GuaranteeTerms hpcwsag:Obligated="ServiceProvider"
  hpcwsag:GuaranteeName="CCSGuaranteeB"
  hpcwsag:ServiceName="CCSService">
  <hpcwsag:NonNegotiableGuarantees>
    <hpcwsag:ResourceTerms>
      <jsdl:ExclusiveExecution>
        true
      </jsdl:ExclusiveExecution>
      <jsdl:OperatingSystem>
        <jsdl:OperatingSystemType>
          <jsdl:OperatingSystemName>
            Super-US
          </jsdl:OperatingSystemName>
        </jsdl:OperatingSystemType>
      </jsdl:OperatingSystemType>
      <jsdl:OperatingSystemVersion>
        10
      </jsdl:OperatingSystemVersion>
    </hpcwsag:ResourceTerms>
  </hpcwsag:NonNegotiableGuarantees>
</hpcwsag:GuaranteeTerms>
</hpcwsag:Other>
</hpcwsag:DataTreatmentTerms>
</hpcwsag:AvailabilityTerms>
</hpcwsag:ApplicationTerms>
</jsdl:ApplicationVersion>
```



```

<jsdl:CPUArchitecture>
  <jsdl:CPUArchitectureName>
    x86
  </jsdl:CPUArchitectureName>
</jsdl:CPUArchitecture>
<jsdl:TotalCPUCount>
  <jsdl:Exact>18</jsdl:Exact>
</jsdl:TotalCPUCount>
</hpcwsag:ResourceTerms>
<hpcwsag:ApplicationTerms>
  <jsdl:ApplicationName>LS-DYNA</jsdl:ApplicationName>
  <jsdl:ApplicationVersion>
    R3.43919
  </jsdl:ApplicationVersion>
</hpcwsag:ApplicationTerms>
<hpcwsag:ApplicationTerms>
  <jsdl:ApplicationName>Java</jsdl:ApplicationName>
  <jsdl:ApplicationVersion>
    jdk1.6.0_7
  </jsdl:ApplicationVersion>
</hpcwsag:ApplicationTerms>
<hpcwsag:AvailabilityTerms>
  <hpcwsag:AvailabilityRate>
    <hpcwsag:MinimumRate>95.0</hpcwsag:MinimumRate>
    <hpcwsag:MaximumRate>95.0</hpcwsag:MaximumRate>
    <hpcwsag:Period>P70D</hpcwsag:Period>
  </hpcwsag:AvailabilityRate>
</hpcwsag:AvailabilityTerms>
<hpcwsag:DataTreatmentTerms>
  <hpcwsag:DataAccess>Limited</hpcwsag:DataAccess>
  <hpcwsag:Confidentiality>
    High
  </hpcwsag:Confidentiality>
  <hpcwsag:DataStorage>Exclusive</hpcwsag:DataStorage>
  <hpcwsag:Other/>
</hpcwsag:DataTreatmentTerms>
<hpcwsag:Other>
  Guaranteed Stable Environment for 1 year
</hpcwsag:Other>
</hpcwsag:NonNegotiableGuarantees>
</hpcwsag:GuaranteeTerms>

```

```
<hpcwsag : BusinessTerms
  hpcwsag : GuaranteeName="CCSGuaranteeA "
  hpcwsag : Name="CCSBusiness">
  <hpcwsag : Price >
    <hpcwsag : FixedPrice >
      <hpcwsag : Complete>
        <hpcwsag : Value>400</hpcwsag : Value>
        <hpcwsag : Currency>Euro</hpcwsag : Currency>
      </hpcwsag : Complete>
    </hpcwsag : FixedPrice >
  </hpcwsag : Price >
</hpcwsag : BusinessTerms >
<hpcwsag : BusinessTerms
  hpcwsag : GuaranteeName="CCSGuaranteeB "
  hpcwsag : Name="CCSBusiness">
  <hpcwsag : Price >
    <hpcwsag : FixedPrice >
      <hpcwsag : Complete>
        <hpcwsag : Value>600</hpcwsag : Value>
        <hpcwsag : Currency>Euro</hpcwsag : Currency>
      </hpcwsag : Complete>
    </hpcwsag : FixedPrice >
  </hpcwsag : Price >
</hpcwsag : BusinessTerms >
</hpcwsag : Terms>
...

```

Listing 6.17: "Terms of the Car Crash Simulation Tender"

GuaranteeTermA provides less CPUs (16 versus 18) and a lower availability rate (90 vs 95 percent) compared to *GuaranteeTermB*. Due to that, the price for the two guarantees also differs and it is up for the customer to decide which of those *GuaranteeTerms* he wants to have.

To ensure that no unexpected requests or changes in the terms appear any more, the service provider has declared all terms as non-negotiable.

PreContext

The Context of the Car Crash Simulation Offer, designed by the service provider, contains now the information of the Tender, on which it is based. In Addition the service provider and customer details are contained in the Offer.

```

<hpcwsag:DocumentBase>
  <hpcwsag:Id>CCSTender01</hpcwsag:Id>
  <hpcwsag:Name>Car Crash Simulation Service</hpcwsag:Name>
  <hpcwsag:Type>Tender</hpcwsag:Type>
</hpcwsag:DocumentBase>

```

Listing 6.18: "DocumentBase of the CCS Tender"

CreationConstraints

The service provider decides to reserve the offered resources for the Car Crash Simulation until he gets an answer from the customer. For that reason, the *CreationConstraints* element of the CCSOffer contains the *ExpirationTime* element which defines until when this Offer is valid.

```

...
<hpcwsag:CreationConstraints>
  <hpcwsag:ExpirationTime>
    2009-10-31T17:00:00Z
  </hpcwsag:ExpirationTime>
</hpcwsag:CreationConstraints>
...

```

Listing 6.19: "Creation Constraints as in the CCS Tender"

6.1.2.3. SLA

After careful analysis, the car manufacturer decides for *GuaranteeTermB* and communicates this to the service provider. Now the SLA is created, containing additionally to the presented terms in the Tender and Offer, the start and end time of the validity of the SLA and by that of the service.

6.1.3. Instantiating the SLA documents for the Visualization Scenario

The Visualization scenario is a classical example for advanced reservation needs of customers. In this case, there is a strong dependency between two different data sources, namely the wind tunnel and the simulation results.

This implies that the service is needed at a certain starting time and with guaranteed quick data access (with a pre-defined response time limit).

6.1.3.1. SLA Template

service provider offering such advanced reservation templates may produce SLA Templates for that purpose. This would not vary from the presented once in the previous chapters so it won't be shown in XML format to avoid an unnecessary extension of the overall chapter.

The Template could refer to an ANSYS CFD service which provides a certain version (6.1) of the simulation service together with a range of the availability rate. Bound to the parameters, the price will vary with respect to the provided quality.

6.1.3.2. SLA Request

Name

As already presented in the last sections, the customer refers with the name to the name in the template he has build his Request on.

PreContext

The Context of the Request is not different to the ones of the previous use cases. In this simple case it contains the information about the customer.

CreationConstraints

The Creation Constraints in this case are also rather simple to define. As a high reliability is needed and the service is very important to be available exactly at the defined time with the defined quality, the customer could request for a high reliability and gold quality of the service.

```
...
<hpcwsag:CreationConstraints>
  <hpcwsag:CreationContext>
    <hpcwsag:Reliability>High</hpcwsag:Reliability>
    <hpcwsag:Quality>Gold</hpcwsag:Quality>
```

```

    </hpcwsag:CreationContext>
  </hpcwsag:CreationConstraints>
  ...

```

Listing 6.20: "Creation Constraints as in the VIS Request"

Terms

Within the Terms section, the customer will define his needed parameters. In this section only the definition of the Network Bandwidth (as stated in R16), the request for guaranteed quick access to the data (referring to a minimal response time as in R17) as well as the pre-defined start time (according to R15) are shown as Schema representation. As they are of highest importance, they are defined as fixed values in the section of non negotiable terms.

```

...
<hpcwsag:NonNegotiableGuarantees>
  <hpcwsag:ResourceTerms>
    <jsdl:IndividualNetworkBandwidth>
      <jsdl:Exact>2.1</jsdl:Exact>
    </jsdl:IndividualNetworkBandwidth>
  </hpcwsag:ResourceTerms>
  <hpcwsag:AvailabilityTerms>
  <hpcwsag:ResponseTime>
    <hpcwsag:MaximumResponseTime>
      100
    </hpcwsag:MaximumResponseTime>
    <hpcwsag:Unit>ms</hpcwsag:Unit>
  </hpcwsag:ResponseTime>
  <hpcwsag:StartTime>
    2009-10-18T12:00:00Z
  </hpcwsag:StartTime>
  </hpcwsag:AvailabilityTerms>
</hpcwsag:NonNegotiableGuarantees>
...

```

Listing 6.21: "Non Negotiable Guarantee Terms as in the VIS SLA Request"

6.1.3.3. SLA

If an SLA is resulting from the negotiation process, it will contain the non-negotiable parameters as defined by the customer. Furthermore all other aspects as validity of the SLA or, if agreed, of single guarantee terms will be stated in the Agreement.

6.2. Processing the Guarantee Terms

In the state of the art section 3.3 resource management systems such as Load Sharing Facility LSF (LSF, cf. 3.3.4), LoadLeveler (cf. 3.3.3) or PBSPro(cf. 3.3.1) were described, which may already in their current versions deliver some of the functionalities, needed to support SLA management. Going step by step through the five guarantee term categories from the Schema (cf. 5.6), namely *Resource Terms*, *Application Terms*, *Availability Terms* and *Data Treatment Terms* it will be shown, in how far the current existing systems can support Service Level Agreements as defined in this thesis.

6.2.1. Resource Terms

Resource Terms, as defined in this schema, were intentionally chosen to base on the JSDL specification as this specification provides the necessary parameters for the resource term definitions. This specification found uptake already in the commercial LSF job scheduler, which so far supports the JSDL terms only partially but also provides own extensions. Additionally a LSF owns a parser that was developed to consume a JSDL document and to convert it into a LSF usable job script.

Another example, where the usage of JSDL was also already evaluated is LoadLeveler. These attempts were described by Rodero et al. [82] but until now JSDL did not find its way yet into one of the releases of the product. Besides these examples, there seems to be no other resource management system, which supports JSDL directly, so far.

However, there exist some solutions

6.2.2. Application Terms

The same as for *Resource Terms* applies also for the *Application Terms*. The needed elements here were all provided by JSDL which is supported by some (but not many) resource management systems.

6.2.3. Availability Terms

The *Availability Terms* as presented in chapter 5.6 cover amongst others elements like *Availability Rate*, *Response Time*, *Maximum Runtime* and *Start/End Time* of the service. The application and estimation of these terms is partially supported by today's resource management systems. Mainly the commercial ones provide the functionality of advanced reservation which ensures the start of the service at a certain point in time.

To allow for setting e.g. the maximum runtime but also the end time of the service, its runtime has to be predicted. This implies always a certain degree of uncertainty as many factors have to be taken into account for this estimation. Factors may be the complexity and the sort of service, as well as the status and the workload of all potentially involved resources.

There were approaches towards this prediction (e.g. by Glasner et al. [83]) but there is no dedicated support by any of the resource management systems at all.

6.2.4. Data Treatment Terms

This kind of terms can as well partially be handled with the current resource management systems. If preferred and negotiated, the location of the data storage of the output files can be set (e.g. with PBSPro).

The *Data Access* and *Confidentiality* can be related to each other. It's up for the service provider to configure its system accordingly, similarly to the executive execution case, there might be also a dedicated physical data storage partition requested. This is not automatically processable as of today.

Intermediate data backups seem not to be supported by today's resource management system, even though they may be needed by the customers.

6.3. Realization of the SLA Lifecycle with today's State of the Art Technologies

Whilst the previous chapter gave an insight into the applicability of the schema, this section will now concentrate on an evaluation of the concepts of the architecture, mainly with focus on the creation process.

Getting towards a common terminology

One of the assumption made within this thesis was that both service provider and customer use (for the HPC domain settled use cases) a common terminology for their desired terms, which are mainly coming from the infrastructure area (e.g. CIM³). Generally, this is from a business viewpoint problematic as in most cases customers will not define their requests on a infrastructure level only and intentionally choose all the terms in a language the service provider understands and vice versa. What is missing here is awareness of the terminology the opponents which is a solvable problem, when humans interact but hard to realize on an automatic level.

The usage of ontologies seems to be a promising approach, which was already topic of projects like OntoGrid⁴ or RealityGrid⁵ and resulted in base concepts such as Semantic OGSA [84] or the Grid Resource Ontology⁶.

Based on these results, further developments have been performed by recent research activities and lead to outcomes like Semantic Annotated Service Level Agreements (SA-SLA) from the BREIN project which are currently still under evaluation by implementation. This concept provides a light weighted annotation of Service Level Agreements, which are represented in this project as a merge of WS-Agreement and WSLA. This could be a future basis for enhanced Service Level Agreement handling, not covering full flexibility of term usage but at least allowing for an enhanced processing of terminology.

6.3.1. Development

Referring to the development phase (cf. chapter 4.3.1), the *Template Generator* as well as the *Tender Generator* could be implemented as GUIs providing entry fields for all terms of the Service Level Agreement. Having to convert only the GUI terms into an XML representation, this would be easy to automate. First realization approaches have been already undertaken in the BREIN and BEInGRID project but are far away from maturity.

However, due to the fact, that also the Business Objectives of the respective party have to be taken into account, this still requires human interaction with the system. So far, there is no technology available which would support automatic adaptation of Service Level Agreement Terms on basis of Business Level Objectives. The concept of involving them (which is a must from an economical viewpoint) was already

³Common Information Model - CIM, <http://www.dmtf.org/standards/cim/>

⁴OntoGrid, <http://www.ontogrid.net/ontogrid/index.html>

⁵RealityGrid, <http://www.realitygrid.org/>

⁶The Grid Resource Ontology, <http://www.unigrids.org/ontology.html>

discussed in several publications ([85], [86], [87], [88], [89]) but the current state of the art is far away from enabling a fully autonomous acting technical realization.

A general issue here is that generated SLA Templates from the service provider can only be as valuable as carefully they are designed. Each Template will have its own and unique identifier, but if the names are well chosen, later retrieval mechanisms may be enabled to find similar Templates easier. This is a rule that should be taken into account when designing or updating Templates for new services.

The realization of a registry or a blackboard is technologically possible. Projects like NextGRID have already provided implementations of Service Registries [90] and the blackboard principle was realized in the Multiagent community [91].

6.3.2. Creation

6.3.2.1. Discovery

Discovery is from a technical viewpoint realizable, again its a matter of the used terminology. The more concrete the search terms for the services are, the better the discovery results are. Technologies like UDDI 3.5.1.1 or WS-Discovery subsub:WSD are mature enough and already used in practice. Additionally mechanisms like UDAP (as presented in [88]) were created and evaluated in the frame of research projects, such as NextGRID. In general a Discovery Service can provide the same logic for the search of Tender or Templates, just needs different interfaces to call for the different approaches. When using a common terminology, the selection and ranking of the hits can be automated, otherwise human interaction is needed.

6.3.2.2. Negotiation

Negotiation provides the same problem with respect to the Business Level Objective handling, as stated in the previous chapter on *Development*. In this phase, as well as in the *Discovery* phase, semantic problems may occur. This is especially the case, when a negotiation opponent enters a value/string which is not recognized by the system. This leads then either to the need of a human interaction or in worst case, an agreement, though it may be fulfilled, will never be established as the terminology is not clear.

An extreme complex and not fully automatable task is the creation of an SLA Offer or later Counter-Offers on side of the service provider. When preparing the SLA Document for the customer, several parameters/information sources and datasets have to be taken into account.

As a reprise of chapter 4.3.2 actions that need to be performed are:

- Check of SLA Templates
- Check of the Infrastructure Capabilities
- Check of the Business Objectives
- Check of the QoS History entries
- Check of the System Status
- Prediction of the System Behaviour
- Reservation of Resources

Checking the Template(s)

Once receiving a Request for an offer, the service provider can look up in his SLA Template repository (realized usually as an XML database) for (a) the SLA Template this Offer refers to (which **MUST** be provided within a Request) and (b) other templates which refer to the service, preferably with similar terms. All the terms need to be analyzed carefully and cross checked with the system and its status.

What can the infrastructure do?

Gaining the information about the infrastructure capabilities and comparing it to the requested terms is a simple task, when the terms directly refer to the resource descriptions. As soon as the terms deviate from those stored in the infrastructure capabilities database, the matching needs to be supported by the involvement of human users.

Handling business objectives

With respect to the Business Objective handling, the same applies as previously said, concepts are there, any technological and mature realization is not available.

The QoS History

A realization of the QoS History and their consultancy in case a new SLA Offer is created was initially described by Tserpes et al. [92] in the frame of the NextGRID project. It showed added value, but was not further developed beyond the end of the project. Nevertheless, the integration of this concept in the current HPC system, will provide added value to the service providers performance and reliability.

Getting the current status of the system

The activities with the highest complexity are the tasks of the System Status component. As described in Table 4.4, its task is not only to look at the current state of the system (and its resources) but also to predict the future status. These are actions which are not provided by any resource management system.

Generally no resource management will give an direct answer on the feasibility of a requested service and its properties, therefore approaches so far have always dealt with providing middleware components as a connector between the batch systems and those components needing the information. It will be up for discussion in how far future resource management systems should inherit functionalities to provide enhanced information sets, this will be addressed again in the conclusions Section 7. In the state of the art section 3.3 resource management systems such as LSF (cf. 3.3.4), LoadLeveler (cf. 3.3.3) or PBSPro(cf. 3.3.1) were described, which have to be integrated as information source for the system/resource related datasets. They can be used to query the status information of the queues, all jobs or jobs chosen by their ID.

Predicting the future

Current resource management systems do not allow for a concrete prediction of future behaviour and utilization of the system. The current state of the art in resource management does not allow for checking in advance, when and how long the job will run, before the job is submitted to the queue. By that, promising advanced reservation (not the execution of it) in terms of analysing the impacts of it gets extremely complicated. Additionally the promise of certain time dependencies (start and end time in the future) which have no buffer time is always risky as one will have to get the status of all the jobs in the queue and try to estimate on base of this the necessary information. Even this estimation will only approximate the real values, and only get close to them when no unforeseen occurrences happen. There are are

workarounds existing like the usage of different queues with different prioritizations or the submission of dummy requests to see the feedback from the schedulers, but this is still a field where research is needed.

Generally speaking, prediction of behaviour and utilization of a service providers system can be done based on information bits which can be retrieved from current resource management systems. But putting the pieces together to an estimation of feasibility of an SLA Offer has to be performed by a complete different component, as previously described.

Reservation of Resources

Depending on the business policies of the service provider, he might decide to reserve the offered system resources whilst the ongoing negotiation. This is not necessarily a must, as this means at the same time that the resources are blocked and cannot be sold otherwise.

Several Resource Managers (PBSPro, LSF or Torque combined with MAUI (cf. 3.3.6) allow for advanced reservation of resources. There were several activities performed within this field, which target the enabling of advanced reservation on basis or connected to Service Level Agreements (e.g. as presented by Di Stefano et al. [93] or Waeldrich et al. [94]).

6.3.3. Provisioning

The Provisioning of the service heavily depends on the terms in the Service Level Agreement as well as on the end users business policies. As stated in the lifecycle overview, not necessarily only service providers have to prepare their system for the service execution, there might be also obligations on the customer, often with respect to provisioning of licenses for the executed applications.

- Configuration of the Monitor and Evaluator
- Set-up of the Data Converter
- Preparation of License Management
- Configuration of the System
- Reservation of Resources

Configuration of Monitor and Evaluator

Currently there are several implementations of SLA Monitors and Evaluators available, the most up to date ones are available on IT-Tude⁷, the online software repository of the BEInGRID project.

These components provide usually the means of configuration with a SLA Document based on a certain SLA schema. By this, it should be easy to configure these components.

Of course there is also a strict dependencies on the used monitoring tools. The above mentioned implementations e.g. base on Ganglia which is connectible with e.g. PBSPro or LSF.

Data Converter Set-up

Monitoring Systems as of today provide the data in a plain format. They are not designed for conversion of the datasets to a higher abstraction level. Therefore this task has to be taken over by another, specialized component and cannot be fulfilled by any of the already existent components of a HPC System.

So far, such a component does not exist in terms of implementation. Concepts for that were designed but a realization or a the availability of a stable implementation is far beyond the current state of the art.

Licence Management

Licence Management beyond the borders of a single cluster is a relatively new topic, which is currently addressed by the BEInGRID and the SmartLM⁸ projects. Both work towards solutions, which enable the usage of externally available licenses, brought in by the customers themselves (see also Li et al. [95] or the LM Whitepaper by Simmendinger [96]).

System Configuration

As discussed in Section 6.2, at least those terms represented in JSDL are already today easy to proceed with some of the resource management systems. For enhanced and automated service configuration a full understanding of the contents of the

⁷IT-Tude, <http://www.it-tude.com>

⁸SmartLM - Grid-friendly software licensing for location independent application execution, <http://www.smartlm.eu>

Service Level Agreement with respect to the service providers system parameters would be needed which is today not possible.

Therefore semantically enhanced components will be needed, which are capable of performing necessary mappings to the SLA terms (as described e.g. in [97]). With respect to the realization of enhanced system configuration on basis of SLAs, the outcomes of the BREIN project, as well as of the SLA@SOI⁹ research activity might be supportive.

Reservation of Resources

In case the resources were not reserved during the negotiation phase, this is done during the *Provisioning* phase. For further description see the section on negotiation (chapter 6.3.2.2).

6.3.4. Execution and Assessment

During this phase, the following tasks need to be performed

- Monitoring and Evaluation
- Logging for Accounting/Billing Purposes
- QoS Logging
- Anonymization of Data
- Reconfiguration of the Resources
- Triggering the Update of SLA Templates

Monitoring and Evaluation

With respect to Monitoring and Evaluation, the implementations as described in chapter 6.3.3 can be used to perform the assessment of the Execution.

⁹SLA@SOI, <http://sla-at-soi.eu/>

Logging for Accounting/Billing Purposes

Here interfaces to accounting/billing components are needed. Several accounting components already exist, which may be used for this purposes. Amongst others, the BEInGRID project provides such a component. However, also commercial accounting and billing systems should be taken into account as realization of this functionality.

QoS/QoE Logging

The QoS Logging is performed by the SLA Manager. It uses the provided interfaces from the *QoS History*, which was described in Section 6.3.2.2. The *QoE History* can be realized as a derivation from the *QoS History*.

Anonymization of Data

The anonymization of data is performed by the *Data Converter* component which was described in the previous section. Such a component needs to provide an interface which can be called by an *SLA Monitor* and needs to be able to process the data in the format the external Evaluator (customer or Third Party) needs to get.

Reconfiguration of the Resources

This task can be performed with most of the existing resource management systems, however, the *SLA Manager* needs for that the functionalities to decide on the new configuration. Besides this, the same issues apply as for the System Configuration in chapter 6.3.3.

Triggering the Update of SLA Templates

This is a functionality which has to be provided by the *SLA Template Generator*. It should be implemented/enhanced in a way which allows for forwarding the SLAID/TemplateID of the misbehaving service together with detailed information about the problematic terms. Until now, there is no component available which offers this functionality.

6.3.5. Termination

During this phase, the following tasks need to be performed

- Stop Job execution In case of the decision to stop an SLA (e.g. due to continuous violations), associated jobs need to be stopped immediately in their execution. This is possible with current resource management systems.
- Remove scheduled jobs This has to be performed when an SLA is unexpectedly broken and the decision was taken to terminate the whole relationship. In that case, all scheduled jobs have to be taken from the queues, which is supported by most of the existing management systems.
- Retrieval of Job execution statistics for final accounting When a job/service has been stopped earlier than expected, the final usage statistics are needed for final accounting and billing. This can be done with most of the existing resource management systems.
- De-configuration of the system All the instances of components which were set up before the service execution have to be de-configured. Actual components such as SLA Monitors, SLA Evaluators offer interfaces for this.

6.4. Enabling enhanced offering of services

Obviously customers, when asked, would prefer full flexibility in requesting and adapting all terms of service descriptions. From a viewpoint of a service provider, this is not eligible as negotiation of all terms would become extremely complex, time consuming and in most cases not cost-efficient.

service providers create SLA Templates for their services based on the capabilities of the service and, of course, on the experience made with customer requests so far (for addressing the market needs). To simplify the handling of requests and of the publication of capabilities, service providers need the means to create SLA Templates addressing different categories of users and circumstances. However, it is also obvious that the more variations of term combination exist, the more complex the design of SLA Offers or Counter-Offers may become. A pragmatic approach would be to design one SLA Template for each possible combination, but this would lead to an increased management overhead which is not in line with the most service providers business policies.

One approach to keep the number of SLA Templates low is to offer only categories of services. Classically, this was often referred to as the Gold, Silver and Bronze approach (see [98], [99]), which refers to different quality levels. This categorization is

today provided by cloud computing vendors such as Zimory¹⁰ (a spin off of T-Labs) or Amazon with its EC2¹¹ service. However, their SLAs are very simple and provide no freedom to negotiate.

The HPC SLA Schema provided within this thesis provides more flexibility for its end users than this simple *Gold, Silver and Bronze* approach but still sticks to the concept of categorizing the service offers. The three presented use cases are stakeholders for different categories:

- The Medical use case for *Urgent Computing*
- The Car Crash Simulation for *Guaranteed Environment*
- The Visualization use case for *Advanced Reservation*

In that cases, service providers could pre-define SLA Templates for the different categories with variation of terms (which then may implement the Gold, Silver, Bronze approach). The more flexible a service provider wants to be in terms of his offered services, the more categories he may implement. However, the categories and the Templates have to be designed very carefully as there are implications of each categories on the availability of others.

When, for example, having a *Advanced Reservation* service sold, then the assigned resources are blocked for any other usage, which could lead to collisions with other services which maybe started before but take longer than expected. This is then also a matter of prioritization of the service providers, which is currently not flexibly manageable by HPC management systems.

This implies another requirement to future resource management systems, which is the embedding of Business Relationship Management (BRM) into the decision process within the SLA Lifecycle. This is needed during *Development, Creation, Provisioning* but also during the *Execution and Assessment* phase.

Especially when a bigger number of Service Level Agreements has to be managed parallel prioritization is needed to ensure the highest business benefits for the service provider whilst satisfying (at least most) of its customers. Not all of the current resource management systems allow for different job prioritization approaches, some of them only provide fair sharing capabilities which is not sufficient for effective job handling. Means for e.g. pre-emption have to be given to enhance the service providers resource management to a level, where he is flexible enough to satisfy his customers.

¹⁰<http://www.zimory.com/>

¹¹<http://aws.amazon.com/ec2/>

6.5. Deploying the SLA Management in a real world environment

To finally validate the presented approach, Figure 6.1 shows a deployment diagram of the SLA Management Framework in a HPC Centre. Thereby emphasis was placed on the logical deployment of the single components (as described in chapter 4 within different firewalls as well as to show the interactions amongst them.

Figure 6.1 is a simplified diagram, which concentrates only on the customer and service provider domain. Components which may be placed in a Trusted Third Party domain (such as the Registry), are for the purpose of simplification assigned to the one or other domain. Generally several of the components in the customer domain could be outsourced to ensure easy access to the system. Additionally the majority of customer-side bits could be simply run on a notebook.

In case that the best effort is sufficient for the customer it may directly access the Grid Access Nodes as it is possible today and does not mandatory need to involve the SLA Management Framework components.

The service providers components have to be aligned with HPC service provider internal firewalls. In the deployment diagram this is defined by some sub-domains which represent different firewalls. For instance resource manager and scheduler shall only be accessible by authorized components within the service provider domain which are represented by the Grid Access Nodes in the diagram. Furthermore some of the SLA Management components such as the service provider SLA Monitor have to be within the resource domain to get access to the necessary data.

Another internal domain that exists in HPC areas is the Administration. Secured by another own firewall, those components and databases are deployed within this domain which provide and proceed information of general system/infrastructure capabilities, accounting/billing tasks, etc.

With Figure 6.1 it becomes obvious that the SLA Management framework can be deployed within an existing HPC infrastructure but the different components itself should be carefully distributed within the different firewall secured domains.

6.6. Towards future resource management systems

During the evaluation of the proposed approaches it became obvious that today's resource management systems do only partially fulfil the requirements for effective

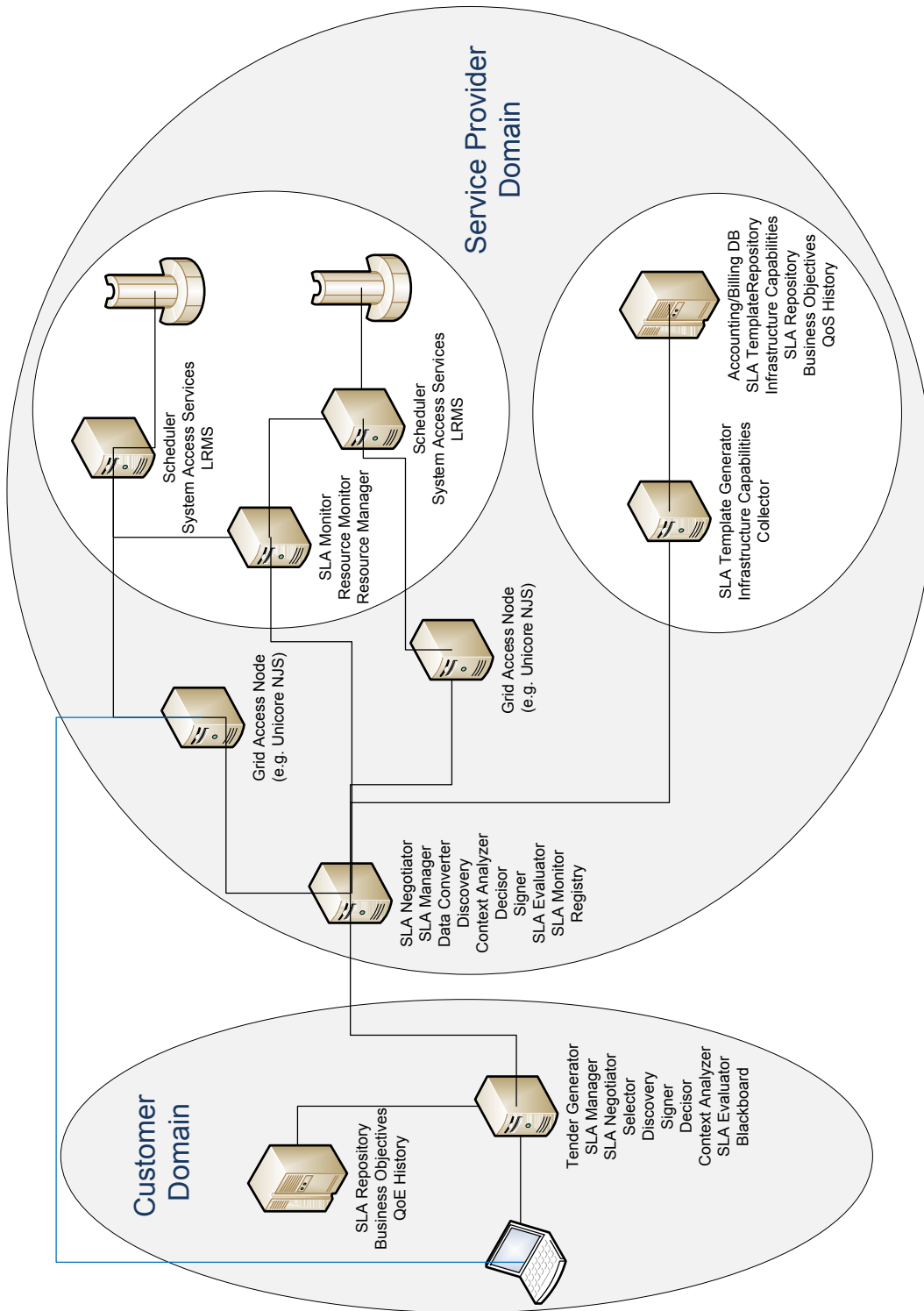


Figure 6.1.: Deploying the SLA Framework in a Real World HPC Environment

and automated SLA Management. One of the main issues with current cluster management systems is their system-centric approach leading to a weak addressing of the users requirements which are the main contents of an SLA. The support of QoS terms is only minimal so that promises will be hard to keep when relying only on the capabilities of a resource management system.

Additionally even if enhanced functionalities are already provided, then by commercial products such as PBSPro or LoadLeveler but not by the freely available ones (as e.g. Torque).

As a summary, the main requirements on future resource management systems are:

- **Integration of Business Relationship Management functionalities**

Current means of prioritizing jobs are rather simple and do not take into account the business policies of the respective service providers. Thus customers which may have all assigned a high priority are handled equally, which may be against the Providers business policies. Such a business policy could for instance be that customer X is always preferred when he has a job to be started. Future resource management systems have to be able to adapt the prioritization on base of those business policies to allow for an effective, business driven

- **Licence Management**

Traditional cluster management assumes that licenses are either not needed or used in a closed environment. However, with the upcoming VO approach, there are means needed to support licenses handling beyond the borders of a single cluster within a single domain. As mentioned before, this is already a research focus of the SmartLM and the BEInGRID projects.

Future resource management systems have to provide enhanced license handling functionalities which allow the usage of external licenses. This allows for license authorization at the submission time of the job(s) independent of the location of the license.

- **Dynamic Pricing** Currently HPC centres offer fixed prices for pre-defined services. By using Service Level Agreements to manage the resources and by that to allow for negotiation of terms, this "fix-price" strategy becomes obsolete and prices have to vary [100].

This implies the need of dynamic pricing functionalities, which should be at least connected to (if not integrated into) the resource management systems of the future. With automated dynamic pricing, service providers will achieve greater profitability as they can then take into account current load, risks and projected supply and demand.

- **Prediction** As elaborated in Section 6.3.2.2, prediction is generally possible but needs additional components that collect single information bits from the

resource management system.

The resource management systems of the future should have integrated prediction possibilities which relies on logged historical data (e.g. by the use of a QoS history). Additionally, if the resource landscape is inhomogeneous, the prediction algorithms of the future have to take this into account.

6.7. Enhanced Business Models

Currently service providers (not only from the HPC domain) follow usually a fixed business model. Requests which do not map to this model are nearly impossible or can only be fulfilled with high effort, and therefore are not efficient.

With the provided solutions for SLA Management in the HPC domain, an adaptation of the so far followed business models of the service provider can become simpler as decision and management processes are supported to a certain degree. With the support of SLAs, outsourcing of capabilities can become easier and could allow for provisioning of composed services on demand, without needed investments inside the service provider domain.

However, the choice of the offered services should be taken very carefully. Services Offered with a high number of negotiable terms increase the complexity of negotiation and at the same time the risk of violating the service during execution.

Therefore a well defined mixture will be needed of service offers which could target certain categories such as *Guaranteed Environment*, *Prioritized Access* or *Timed Access* (as described in the PhD thesis from Wesner [101]) but also of so-called *Best Effort* SLAs which may have only a small number of parameters and are non-critically to execute. By that, service provider could maximize the utilization of their resources whilst being able to shift best-effort jobs when a more important service should be fulfilled.

Chapter 7.

Conclusions

Current solutions for resource management of High Performance Computing environments were mainly developed neglecting the business needs and goals of service providers and customers. With the introduction of Service Level Agreements, a valuable tool was presented which can be used to express exactly those (important) business-related terms in a way, allowing enhanced and automated management of offered HPC services.

So far the use of Service Level Agreement in the management of HPC resources has only found minor uptake and the possibilities provided with respect to define the contents of an SLA are rather simple. Specifications and standards for Service Level Agreements and their handling have been produced, but none of them is in a state which would allow for easy uptake.

The approach within this thesis assumes that Service Level Agreements need a dedicated management framework which may be embedded in a middleware which is used to steer an HPC environment. By that, also further evolution towards a simplified participation in Virtual Organizations and by that an extension of the business fields could be achieved, opening completely new business opportunities. Whilst other approaches prefer a start from scratch, this thesis has based its developments as far as possible on existing technologies and by that concentrated on the gaps which need to be filled in future work.

Thereby the proposed solution is split in two parts: on one hand concentrating on the needed functionalities with respect to Service Level Agreement management (leading to an SLA framework architecture), on the other hand also providing an SLA Schema supporting the definition and handling of Quality of Service parameters in HPC.

The proposed framework for SLA Management supports the end users (customer and service provider) in all phases of the lifecycle, from Development over Creation up to Provisioning, Execution and Assessment and Termination. Thereby it was designed to provide the best possible business flexibility in terms of creating an SLA but also its execution. By that, it puts no longer the burden of discovery solely on the

customer but allows reverse discovery (of customers by service providers) through mechanisms such as invitations to tender. Steered by the context of the respective requests and services, the creation (namely Discovery and Negotiation) can follow different strategies which allows both, service provider and customer to initiate the creation process. In this way it is clear that the framework itself cannot be isolated but needs interactions with "external" components such as Accounting, Business Relationship Management or the Resource Management Systems of the Provider. Aligned with the architecture design, an HPC SLA Schema was produced, which covers the information about the business context of the respective partners but also the terms with respect to Quality of Service. Being settled on different levels, the guarantee terms, cover system oriented terms such as resource or application descriptions and the associated prices, penalties and much more. Nonetheless the outcomes can not be fully applied to today's HPC management environments, as there are severe functionalities missing, which have to be covered by further evolutions of these environments, mainly in terms of the resource management systems. However, from what has been reported in this thesis, it should be obvious that a SLA Management system for HPC could be released as a basic implemented framework already today. The work in this thesis is aligned with recent outcomes of research projects and based its developments on real-world requirements gained from three chosen use-cases. By that, the needs of the user communities are exactly mirrored in the developments and the solution, when fully functional realized will support High Performance Computing Centres also to open new business opportunities, such as participating as providers in the IAAS (Infrastructure as a Service) movement.

7.1. Future Work

7.1.1. Enhanced Prediction of the Service Boundaries

As already stated in chapter 6.6, prediction of service runtime or system behaviour is with today's resource management systems nearly impossible. Even though approaches towards the prediction exist, they are only in a minority of cases correct and by that provide no basis for reliable estimations.

Future work should concentrate on evolving prediction mechanisms that allow for an enhanced estimation of the satisfiability of requested service parameters and by that to support the decisions on whether to accept a Service Level Agreement request or not.

7.1.2. Intelligent Context analysis and automated choice of protocols

In this work, the context in terms of creation constraints was very simple and consisted of a set of pre-defined values. At the same time this limits the flexibility of the approach of course. Whilst it is hard to predict if overall flexibility in term definitions will ever be possible (so far it seems that technologies supporting this are not existent), at least enhancements of this approach should be possible.

Figure 7.1 shows an enhanced architecture for the creation of Service Level Agree-

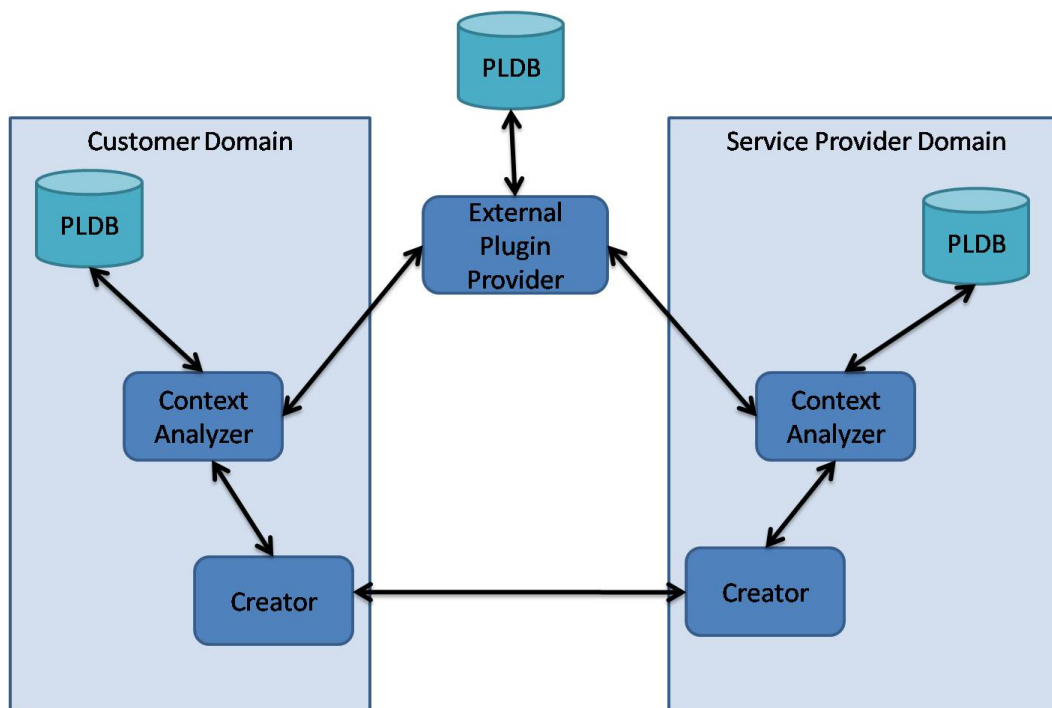


Figure 7.1.: Probable extension with a plug-in concept

ments which is based on embedding a plug-in concept into the presented solution. This architecture is intentionally symmetric for both the customer and the service provider domain, so as to keep the ability of both to trigger a discovery or a negotiation process. Instead of a component "Discovery" or "Negotiator" there will be one component taking over the functionality of both, depending on the plugged-in logic. This logic can be manifold covering discovery and negotiation protocols and offering the possibility for development of new, enhanced protocols which may be easily tested with an enhanced framework. The new component is designated the "Creator". Furthermore every side owns a component called Context Analyzer, responsible for

analyzing the context node and deciding based on this which plug-in to load. Place for retrieval of the plug-in is either an internal plug-in database (PLDB) or an external database offered by a third party provider, which would also open the market for professional offerers of those plug-ins.

7.1.3. Enhanced Terminology Mapping

The BREIN project addressed already with its SA-SLA specification (Semantic Annotated Service Level Agreements) the problematic of term mappings in different languages. A customer saying X could mean the same which is known to a service provider as Y, but as the one doesn't know the other, they will not come to an agreement for this term.

This is not the only level where mappings are necessary. A simple classification of those levels is shown in Figure 7.2.

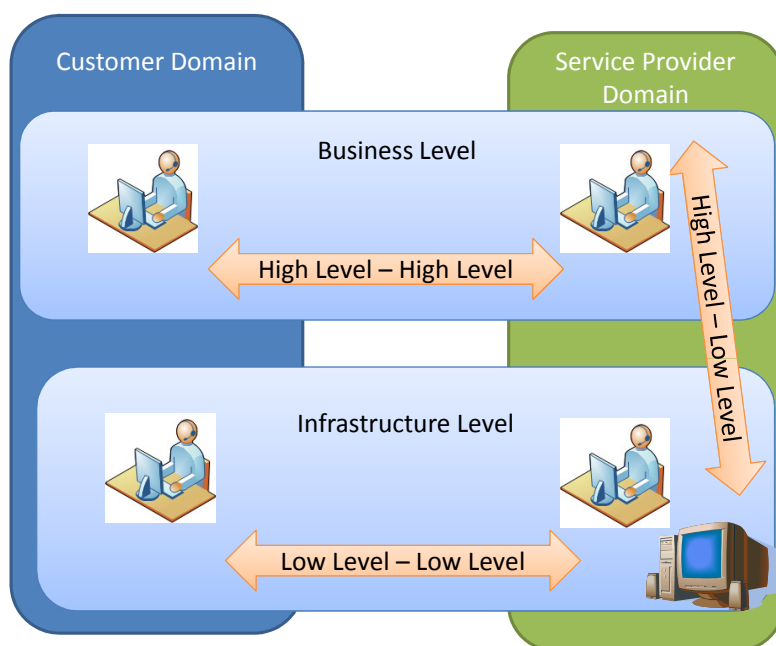


Figure 7.2.: Variation of the terminology according to the interaction level

- **High Level to High Level:**
In this case, the level of terms should be the same, but usage of the same terms for the same parameter is not ensured by that. Whilst someone would talk

about "duration" the business opponent could talk about "estimated execution time", meaning the same. This is some sort of translation problem, which could be solved by using Ontologies

- **High Level to Low Level:**

This case is settled within the service provider domain. Having a customer asking for Service XYZ with parameter "completion time = 1 hour" is not directly convertible as it concerns time required for deployment and provisioning of the service in the service provider infrastructure. This is a mapping problem from the business world requirements to the lower level infrastructure parameters which is currently performed by system administrators.

- **Low Level to Low Level:**

This case occurs, when both, customer and service provider are operating/negotiating on the plain infrastructure level but the terminology is different. A typical example is the outsourcing of capabilities of a service provider to a Cloud Provider. In this case, the service provider gets into the role of the customer and the Cloud Provider is the service provider. This could be solved as well by infrastructure level Ontologies.

Looking at the different levels, it becomes obvious that approaches like the SA-SLAs are only a small step forward. Project initiatives like e.g. SLA@SOI are also currently researching this area, but with small dedicated effort.

Further evolution of this sector would lead to a powerful and nearly fully automated solution.

Appendix A.

The XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:hpcwsag="http://schemas.bako.org/phd/2009/10/hpcws-agreement"
xmlns:wsla="http://www.ibm.com/wsla"
xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace=
  "http://schemas.bako.org/phd/2009/10/hpcws-agreement"
elementFormDefault="qualified" attributeFormDefault="qualified">
  <xs:import namespace=
    "http://schemas.ggf.org/graap/2007/03/ws-agreement"
    schemaLocation="C:\Users\R_Steele\Documents\PhD\Needful
    Things\ws-agreement.xsd"/>
  <xs:import namespace="http://schemas.ggf.org/jsdl/2005/11/jsdl"
    schemaLocation="C:\Users\R_Steele\Documents\PhD\Needful
    Things\jsdl.xsd-18.xsd"/>
  <xs:import namespace="http://www.ibm.com/wsla"
    schemaLocation="http://www.research.ibm.com/wsla/WSLA093.xsd"/>
  <xs:element name="SLA Agreement"
    type="hpcwsag:SLA AgreementType"/>
  <xs:element name="PreSLADocument"
    type="hpcwsag:PreSLADocumentType"/>
  <xs:element name="Context"
    type="hpcwsag:AgreementContextType"/>
  <xs:element name="PreContext"
    type="hpcwsag:PreContextType"/>
  <xs:element name="Terms"
    type="hpcwsag:TermType"/>
  <xs:element name="AgreementTerms"
```

```
    type="hpcwsag:AgreementTermType"/>
<xs:element name="ExclusionTerms"
  type="xs:string"/>
<xs:element name="ServiceTerms"
  type="hpcwsag:ServiceTermType"/>
<xs:element name="ResourceTerms"
  type="jsdl:Resources_Type"/>
<xs:element name="GuaranteeTerms"
  type="hpcwsag:AgreementGuaranteesTermType"/>
<xs:element name="NonNegotiableGuarantees"
  type="hpcwsag:GuaranteeTermType"/>
<xs:element name="NegotiableGuarantees"
  type="hpcwsag:GuaranteeTermType"/>
<xs:element name="ApplicationTerms"
  type="jsdl:Application_Type"/>
<xs:element name="BusinessTerms"
  type="hpcwsag:BusinessValueListType"/>
<xs:element name="AvailabilityTerms"
  type="hpcwsag:AvailabilityTermType"/>
<xs:element name="ResponseTime"
  type="hpcwsag:ResponseTimeType"/>
<xs:element name="CreationConstraints"
  type="hpcwsag:ConstraintType"/>
<xs:element name="CreationContext"
  type="hpcwsag:CreationContextType"/>
<xs:element name="ParameterConstraints"
  type="wsag:ConstraintSectionType"/>
<xs:element name="DataTreatmentTerms"
  type="hpcwsag:DataTreatmentType"/>
<xs:element name="DataBackup"
  type="hpcwsag:DataBackUpType"/>
<xs:element name="Price"
  type="hpcwsag:PriceType"/>
<xs:element name="FixedPrice"
  type="hpcwsag:FixedPriceType"/>
<xs:element name="VariablePrice"
  type="hpcwsag:VariablePriceType"/>
<xs:element name="Item"
  type="wsag:OfferItemType"/>
<xs:element name="DocumentBase"
  type="hpcwsag:DocumentBaseType"/>
<xs:element name="Party"
  type="hpcwsag:PartyType"/>
```

```

<xs:element name="ThirdParty"
  type="hpcwsag:ThirdPartyType"/>
<xs:complexType name="AgreementType">
  <xs:sequence>
    <xs:element ref="wsag:Name" minOccurs="0"/>
    <xs:element ref="hpcwsag:ExclusionTerms" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SLAAgreementType">
  <xs:complexContent>
    <xs:extension base="hpcwsag:AgreementType">
      <xs:sequence>
        <xs:element ref="hpcwsag:AgreementTerms"/>
        <xs:element ref="hpcwsag:Context"/>
      </xs:sequence>
      <xs:attribute name="AgreementId" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="PreSLADocumentType">
  <xs:complexContent>
    <xs:extension base="hpcwsag:AgreementType">
      <xs:sequence>
        <xs:element ref="hpcwsag:Terms"/>
        <xs:element ref="hpcwsag:PreContext"/>
        <xs:element name="CreationConstraints"
          type="hpcwsag:ConstraintType" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="Id" type="xs:string"/>
      <xs:attribute name="DocType" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ContextType">
  <xs:sequence>
    <xs:element name="ThirdParty"
      type="hpcwsag:ThirdPartyType" minOccurs="0"/>
    <xs:element name="DocumentBase"
      type="hpcwsag:DocumentBaseType" minOccurs="0"/>
    <xs:element name="Validity"
      type="wsa:PeriodType" minOccurs="0"/>
    <xs:element name="Other"
      type="xs:anyType" minOccurs="0"/>
  </xs:sequence>

```

```
</xs:sequence>
</xs:complexType>
<xs:complexType name="AgreementContextType">
  <xs:complexContent>
    <xs:extension base="hpcwsag:ContextType">
      <xs:sequence>
        <xs:element name="ServiceProvider"
          type="hpcwsag:PartyType"/>
        <xs:element name="Customer"
          type="hpcwsag:PartyType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="DocumentBaseType">
  <xs:sequence>
    <xs:element name="Id" type="xs:string"/>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Type" type="hpcwsag:DocumentType"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="DocumentType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Template"/>
    <xs:enumeration value="Tender"/>
    <xs:enumeration value="Offer"/>
    <xs:enumeration value="CounterOffer"/>
    <xs:enumeration value="Request"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="PreContextType">
  <xs:complexContent>
    <xs:extension base="hpcwsag:ContextType">
      <xs:sequence>
        <xs:element name="ServiceProvider"
          type="hpcwsag:PartyType" minOccurs="0"/>
        <xs:element name="Customer"
          type="hpcwsag:PartyType" minOccurs="0"/>
        <xs:element name="AgreementInitiator"
          type="hpcwsag:AllPartyType" minOccurs="0"/>
        <xs:element name="AgreementResponder"
          type="hpcwsag:AllPartyType" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```

    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="PartyType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="ContactPerson" type="xs:string"/>
    <xs:element name="Street" type="xs:string"/>
    <xs:element name="City" type="xs:string"/>
    <xs:element name="Country" type="xs:string"/>
    <xs:element name="Phone" type="xs:string"/>
    <xs:element name="Fax" type="xs:string" minOccurs="0"/>
    <xs:element name="AdditionalInfo" type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ThirdPartyType">
  <xs:complexContent>
    <xs:extension base="hpcwsag:PartyType">
      <xs:sequence>
        <xs:element name="Role" type="hpcwsag:RoleType"/>
        <xs:element name="Owner" type="hpcwsag:SimplePartyType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="RoleType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="EvaluatorService"/>
    <xs:enumeration value="NotaryService"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SimplePartyType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Customer"/>
    <xs:enumeration value="ServiceProvider"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="AllPartyType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Customer"/>
    <xs:enumeration value="ServiceProvider"/>
    <xs:enumeration value="Third Party"/>
  </xs:restriction>

```

```

    </xs:restriction >
</xs:simpleType>
<xs:complexType name="AgreementTermType">
  <xs:sequence>
    <xs:element name="ServiceTerms "
      type="hpcwsag:ServiceTermType"/>
    <xs:element name="GuaranteeTerms "
      type="hpcwsag:AgreementGuaranteesTermType "
      maxOccurs="unbounded"/>
    <xs:element name="BusinessTerms "
      type="hpcwsag:BusinessValueListType "
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="TermType">
  <xs:sequence>
    <xs:element name="ServiceTerms "
      type="hpcwsag:ServiceTermType"/>
    <xs:element name="GuaranteeTerms "
      type="hpcwsag:GuaranteesType " maxOccurs="unbounded"/>
    <xs:element name="BusinessTerms "
      type="hpcwsag:BusinessValueListType " maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="GuaranteesType">
  <xs:sequence>
    <xs:element name="NonNegotiableGuarantees "
      type="hpcwsag:GuaranteeTermType " minOccurs="0"/>
    <xs:element name="NegotiableGuarantees "
      type="hpcwsag:GuaranteeTermType " minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="GuaranteeName" type="xs:string "
    use="required"/>
  <xs:attribute name="Obligated" type="hpcwsag:AllPartyType "
    use="required"/>
  <xs:attribute name="ServiceName" type="xs:string "
    use="required"/>
</xs:complexType>
<xs:complexType name="AgreementGuaranteesTermType">
  <xs:complexContent>
    <xs:extension base="hpcwsag:GuaranteeType">
      <xs:sequence>
        <xs:element name="ResourceTerms "

```

```

        type="jsdl:Resources_Type"
        minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="ApplicationTerms"
    type="jsdl:Application_Type"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="AvailabilityTerms"
    type="hpcwsag:AvailabilityTermType"
    minOccurs="0"/>
<xs:element name="DataTreatmentTerms"
    type="hpcwsag:DataTreatmentType"
    minOccurs="0"/>
<xs:element name="Other"
    type="xs:anyType" minOccurs="0"/>
    </xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="GuaranteeType">
    <xs:attribute name="GuaranteeName"
        type="xs:string" use="required"/>
    <xs:attribute name="Obligated"
        type="hpcwsag:AllPartyType" use="required"/>
    <xs:attribute name="ServiceName"
        type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="ServiceTermType">
    <xs:sequence>
        <xs:element name="ServiceDescription"
            type="wsag:ServiceDescriptionTermType" minOccurs="0"/>
        <xs:element name="ServiceReference"
            type="wsag:ServiceReferenceType" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="GuaranteeTermType">
    <xs:sequence>
        <xs:element name="ResourceTerms"
            type="jsdl:Resources_Type" minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="ApplicationTerms"
            type="jsdl:Application_Type" minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="AvailabilityTerms"
            type="hpcwsag:AvailabilityTermType" minOccurs="0"/>

```

```

    <xs:element name="DataTreatmentTerms"
      type="hpcwsag:DataTreatmentType" minOccurs="0"/>
    <xs:element name="Other" type="xs:anyType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="AgreementGuaranteeTermType">
  <xs:sequence>
    <xs:element name="ResourceTerms"
      type="jsdl:Resources_Type" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="ApplicationTerms"
      type="jsdl:Application_Type" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="AvailabilityTerms"
      type="hpcwsag:AvailabilityTermType" minOccurs="0"/>
    <xs:element name="DataTreatmentTerms"
      type="hpcwsag:DataTreatmentType" minOccurs="0"/>
    <xs:element name="Other" type="xs:anyType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="BusinessValueListType">
  <xs:complexContent>
    <xs:extension base="wsag:BusinessValueListType">
      <xs:sequence>
        <xs:element name="Price"
          type="hpcwsag:PriceType" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="Name" type="xs:string" use="required"/>
      <xs:attribute name="GuaranteeName"
        type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="AvailabilityTermType">
  <xs:sequence>
    <xs:element name="AvailabilityRate"
      type="hpcwsag:AvailabilityRateType" minOccurs="0"/>
    <xs:element name="ResponseTime"
      type="hpcwsag:ResponseTimeType" minOccurs="0"/>
    <xs:element name="MaximumRunTime"
      type="xs:duration" minOccurs="0"/>
    <xs:element name="StartTime"
      type="xs:dateTime" minOccurs="0"/>
  </xs:sequence>

```

```

    <xs:element name="EndTime"
      type="xs:dateTime" minOccurs="0"/>
    <xs:element name="InvocationLimit"
      type="xs:int" minOccurs="0"/>
    <xs:element name="Other"
      type="xs:anyType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="Reference" type="xs:string"/>
</xs:complexType>
<xs:complexType name="ResponseTimeType">
  <xs:sequence>
    <xs:element name="MaximumResponseTime" type="xs:int"/>
    <xs:element name="MinimumResponseTime" type="xs:int"/>
    <xs:element name="Unit" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="AvailabilityRateType">
  <xs:sequence>
    <xs:element name="MinimumRate"
      type="xs:float" minOccurs="0"/>
    <xs:element name="MaximumRate"
      type="xs:float" minOccurs="0"/>
    <xs:element name="Description"
      type="xs:string" minOccurs="0"/>
    <xs:element name="Period"
      type="xs:duration" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="CreationContextType">
  <xs:sequence>
    <xs:element name="Urgency"
      type="hpcwsag:LevelType" minOccurs="0"/>
    <xs:element name="TimeLimit"
      type="xs:duration" minOccurs="0"/>
    <xs:element name="Reliability"
      type="hpcwsag:LevelType"
      minOccurs="0"/>
    <xs:element name="Quality"
      type="hpcwsag:QualityType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="PriceType">
  <xs:sequence>

```

```

    <xs:element name="FixedPrice"
      type="hpcwsag:FixedPriceType" minOccurs="0"/>
    <xs:element name="VariablePrice"
      type="hpcwsag:VariablePriceType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="FixedPriceType">
  <xs:sequence>
    <xs:element name="Complete"
      type="hpcwsag:FixedPriceCompleteType"
      minOccurs="0"/>
    <xs:element name="PerUnit"
      type="hpcwsag:FixedPricePerUnitType"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="FixedPriceCompleteType">
  <xs:sequence>
    <xs:element name="Value" type="xs:float"/>
    <xs:element name="Currency" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="VariablePriceType">
  <xs:sequence>
    <xs:element name="Complete"
      type="hpcwsag:VariablePriceCompleteType"
      minOccurs="0"/>
    <xs:element name="PerUnit"
      type="hpcwsag:VariablePricePerUnitType"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="VariablePriceCompleteType">
  <xs:sequence>
    <xs:element name="MinimumValue" type="xs:float"/>
    <xs:element name="MaximumValue" type="xs:float"/>
    <xs:element name="Currency" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="FixedPricePerUnitType">
  <xs:sequence>
    <xs:element name="Unit" type="xs:string"/>
    <xs:element name="UnitValue" type="xs:float"/>
  </xs:sequence>
</xs:complexType>

```

```

    <xs:element name="Currency" type="xs:string"/>
    <xs:element name="Description" type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="VariablePricePerUnitType">
  <xs:sequence>
    <xs:element name="Unit" type="xs:string"/>
    <xs:element name="UnitValueMinimum" type="xs:float"/>
    <xs:element name="UnitValueMaximum" type="xs:float"/>
    <xs:element name="Currency" type="xs:string"/>
    <xs:element name="Description" type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DataTreatmentType">
  <xs:sequence>
    <xs:element name="DataAccess"
      type="hpcwsag:AccessType" minOccurs="0"/>
    <xs:element name="Confidentiality"
      type="hpcwsag:LevelType" minOccurs="0"/>
    <xs:element name="DataBackUp"
      type="hpcwsag:DataBackUpType" minOccurs="0"/>
    <xs:element name="DataStorage"
      type="hpcwsag:DataStorageType" minOccurs="0"/>
    <xs:element name="Other" type="xs:anyType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="LevelType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="High"/>
    <xs:enumeration value="Medium"/>
    <xs:enumeration value="Low"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DataStorageType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Exclusive"/>
    <xs:enumeration value="Shared"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="AccessType">
  <xs:restriction base="xs:string">

```

```
<xs:enumeration value="Unlimited"/>
<xs:enumeration value="SpecialGroup"/>
<xs:enumeration value="Limited"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="QualityType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Gold"/>
    <xs:enumeration value="Silver"/>
    <xs:enumeration value="Bronze"/>
    <xs:enumeration value="Best Effort"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="ConstraintType">
  <xs:sequence>
    <xs:element name="ExpirationTime"
      type="xs:dateTime" minOccurs="0"/>
    <xs:element name="ParameterConstraints"
      type="wsag:ConstraintSectionType" minOccurs="0"/>
    <xs:element name="CreationContext"
      type="hpcwsag:CreationContextType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DataBackUpType">
  <xs:sequence>
    <xs:element name="FullBackup"
      type="xs:boolean" minOccurs="0"/>
    <xs:element name="DiffBackup"
      type="xs:boolean" minOccurs="0"/>
    <xs:element name="BackUpServer"
      type="xs:string" minOccurs="0"/>
    <xs:element name="TimeInterval"
      type="xs:duration" minOccurs="0"/>
    <xs:element name="Other"
      type="xs:anyType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Listing A.1: "The SLA Schema"

Bibliography

- [1] I. T. Foster and C. Kesselman, *The Grid, Blueprint for a New Computing Infrastructure*. San Francisco: Morgan Kaufmann Publishers, 2002. (document)
- [2] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Lecture Notes in Computer Science*, vol. 2150, 2001. (document), 1
- [3] J. Cope and H. M. Tufo, "Adapting Grid Services for Urgent Computing Environments," in *ICSOFT (PL/DPS/KE)* (J. Cordeiro, B. Shishkov, A. Ranchordas, and M. Helfert, eds.), pp. 135–142, INSTICC Press, 2008. (document)
- [4] O. of Government Commerce (OGC), "Official introduction to the ITIL service lifecycle," 2007. (document)
- [5] A. Andrieux, H. Ludwig, *et al.*, "Web Services Agreement Specification (WS-Agreement)," tech. rep., Open Grid Forum - Grid Resource Allocation and Agreement Protocol Working Group, 2007. (document)
- [6] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, "Job Submission Description Language (JSDL) Specification, Version 1.0," tech. rep., JSDL Working Group, Open Grid Forum, 2005. (document), 3.2.6
- [7] H. Herenger, R. Heek, R. Kuebert, and M. Surridge, *Operating Virtual Organizations using Bipartite Service Level Agreements*. Springer, 2007. 1
- [8] S. Crompton, M. Wilson, A. Arenas, L. Schubert, D. Cojocarasu, J. Hu, and P. Robinson, "The TrustCoM General Virtual Organisation Agreements," in *6th UK e-Science All Hands Meeting*, 2007. 1
- [9] ITIL, "IT Infrastructure Library." Website: <http://www.itil.org>, 2009. 1
- [10] D. Snelling, M. Fisher, A. Basermann, F. Wray, P. Wieder, and M. Surridge, "NextGRID Vision and Architecture White Paper," tech. rep., NEXTGRID, 2007. 1, 3.5.2.3, 4.2.2.1

- [11] A. Arenas, I. Djordjevic, T. Dimitrakos, L. Titkov, J. Claessens, C. Geuer-Pollmann, E. C. Lupu, N. Tuptuk, S. Wesner, and L. Schubert, "Toward Web Services Profiles for Trust and Security in Virtual Organisations," in *Collaborative Networks and Their Breeding Environments* (L. Camarinha-Matos, H. Afsermanseh, and A. Ortiz, eds.), IFIP, pp. 175–182, Springer, 2005. 1
- [12] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management*, vol. V11, pp. 57–81, March 2003. 1, 3.2.1
- [13] T. T. Forum, *SLA Management Handbook, Volume 2, Concepts and Principles - GB917 v2.5, r2.5*. TM Forum, 2005. 1.3
- [14] I. Foster *et al.*, "The Open Grid Services Architecture, Version 1.5," tech. rep., OGSA, July 2006. 1.3, 3.1.1
- [15] B. Koller, D. Snelling, P. Hasselmeyer, and K. Tserpes, "NextGRID SLA Management Use Cases," tech. rep., NextGRID, 2008. 1.3.1
- [16] R. Haines, S. Manos, S. Zasada, M. Mazzeo, R. Pinning, J. Brooke, and P. V. Coveney, "Grid-enabled neurosurgical imaging using simulation," *Capability Computing, The newsletter of the HPCx community*, vol. 11, pp. 6–10, 2008. 2.1.1.1
- [17] S. Manos, S. Zasada, and P. V. Coveney, "Life or Death Decision-making: The Medical Case for Large-scale, On-demand Grid Computing," *CTWatch Quarterly*, vol. Volume 4, pp. 35–45, March 2008. 2.1.1.1
- [18] World Health Organization, ed., *The World Health Report 2008*. WHO, January 2008. 2.1.1.1
- [19] M. D. Mazzeo and P. V. Coveney, "HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries," *Computer Physics Communications*, vol. 178, no. 12, pp. 894–914, 2008. 2.1.1.2
- [20] R. Barbera, A. Falzone, V. Ardizzone, and D. Scardaci, "The GENIUS Grid Portal: Its Architecture, Improvements of Features, and New Implementations about Authentication and Authorization.," in *WETICE*, pp. 279–283, IEEE Computer Society, 2007. 2.1.1.2
- [21] G. Andronico, R. Barbera, A. Falzone, G. L. Re, A. Pulvirenti, and A. Rodolico, "The GENIUS web portal: grid computing made easy," in *ITCC*, pp. 425–431, IEEE Computer Society, 2003. 2.1.1.2
- [22] J. Bernsdorf, S. Harrison, S. M. Smith, P. Lawford, and D. R. Hose, "A Lattice Boltzmann HPC Application in Medical Physics," in *High Performance Computing on Vector Systems 2006, Proceedings of the High Performance Computing Center*

- Stuttgart, March 2006 (Resch, M. B[^]nisch, T. Tiyyagura, S. Furui, T. Seo, Y. Bez, and W., eds.), Springer, 2007. 2.1.1.2
- [23] J. Bernsdorf and D. Wang, "Non-Newtonian Blood Flow Simulation on Cerebral Aneurysms." Unpublished. 2.1.1.2
- [24] B. Raphael and I. F. Smith, *Fundamentals of computer-aided engineering*. Wiley, 2003. 2.1.2.1
- [25] D. A. Field, "Education and training for CAD in the auto industry," *Computer-Aided Design*, vol. 36, no. 14, pp. 1431–1437, 2004. 2.1.2.1
- [26] P. Hermann, T. Ting, and V. Kitsios, "Development of a fatigue analysis tool chain for automotive structural applications," *SAE transaction*, vol. 114, pp. 522–530, 2005. 2.1.2.1
- [27] S. Heimbs, F. Strobl, P. Middendorf, S. Gardner, B. Eddington, and J. Key, "Crash Simulation of an F1 Racing Car Front Impact Structure," in *Proceedings from the 7th European LS-DYNA Conference, Salzburg, Austria, 14th to 15th of May, 2009*. 2.1.2.1
- [28] P. Angeleri, D. F. Lozupone, F. Piccolo, and J. Clinckemallie, "PAM-CRASH on the IBM 3090/VF: An Integrated Environment for Crash Analysis," *IBM Systems Journal*, vol. 27, no. 4, pp. 541–560, 1988. 2.1.2.1
- [29] A. Quiroz and M. Parashar, "A framework for distributed content-based web services notification in Grid systems," *Future Gener. Comput. Syst.*, vol. 24, no. 5, pp. 452–459, 2008. 3.1.1
- [30] G. Moltó, V. Hernández, and J. M. Alonso, "A service-oriented WSRF-based architecture for metascheduling on computational Grids," *Future Gener. Comput. Syst.*, vol. 24, no. 4, pp. 317–328, 2008. 3.1.1
- [31] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," tech. rep., OASIS, 2006. 3.1.1
- [32] M. Gudgin, M. Hadley, and T. Rogers, "Web Services Addressing 1.0 - Core," tech. rep., W3C, 2006. 3.1.1
- [33] D. Erwin, "UNICORE. Eine europäische Grid Middleware zum Einsatz in produktiven Grid Infrastrukturen," tech. rep., Unicore Forum e. V., 2003. 3.1.2
- [34] H. Lederer, G. J. Pringle, D. Girou, M.-A. Hermanns, and G. Erbacher, "DEISA: Extreme Computing in an Advanced Supercomputing Environment.," in *PARCO* (C. H. Bischof, H. M. Becker, P. Gibbon, G. R. Joubert, T. Lippert, B. Mohr, and F. J. Peters, eds.), vol. 15 of *Advances in Parallel Computing*, pp. 687–688, IOS Press, 2007. 3.1.2

- [35] GCS, "Gauss Centre for Supercomputing." Website: <http://www.gauss-centre.eu/>, July 2009. 3.1.2
- [36] D. Box *et al.*, "Web Services Addressing (WS-Addressing)," tech. rep., W3C, 2004. 3.1.2
- [37] B. Schuller, B. Demuth, H. Mix, K. Rasch, M. Romberg, S. Sild, U. Maran, P. Bala, E. del Grosso, M. Casalegno, N. Piclin, M. Pintore, W. Sudholt, and K. Baldridge, "Chemomentum - UNICORE 6 Based Infrastructure for Complex Applications in Science and Technology," in *Euro-Par Workshops* (L. BougÈ, M. Forsell, J. L. Träff, A. Streit, W. Ziegler, M. Alexander, and S. Childs, eds.), vol. 4854 of *Lecture Notes in Computer Science*, pp. 82–93, Springer, 2007. 3.1.2
- [38] M. SurrIDGE, S. Taylor, D. D. Roure, and E. Zaluska, "Experiences with GRIA; Industrial Applications on a Web Services Grid," *e-Science and Grid Computing, International Conference on*, vol. 0, pp. 98–105, 2005. 3.1.3
- [39] S. Campana, D. Rebatto, and A. Sciabà, "Experience with the gLite Workload Management System in ATLAS Monte Carlo production on LCG," Tech. Rep. CERN-IT-Note-2007-042, CERN, Geneva, Oct 2007. 3.1.4
- [40] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005. 3.1.4
- [41] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten, "Portable batch system: External reference specification," tech. rep., MRJ Technology Solutions, November 1999. 3.1.4
- [42] U. Schwickerath and V. Lefebure, "Usage of LSF for batch farms at CERN," Tech. Rep. CERN-IT-Note-2007-018, CERN, Geneva, Sep 2007. 3.1.4
- [43] M. Wilson, A. Arenas, D. Chadwick, T. Dimitrakos, J. Doser, P. Giambiagi, D. Golby, C. Geuer-Pollman, J. Haller, S. Ketil, and ..., "The TrustCoM approach to enforcing agreements between interoperating enterprises," in *Interoperability for Enterprise Software and Applications Conference (I-ESA'06)*, (Bordeaux, France), March 2006. 3.2.1
- [44] G. Laria and other members of the BREIN consortium, "Final BREIN Architecture - D4.1.3 v2." Website: <http://www.gridsforbusiness.eu/>, July 2009. 3.2.1, 3.4.3
- [45] P. Hasselmeyer, C. Qu, B. Koller, L. Schubert, and P. Wieder, "Towards Autonomous Brokered SLA Negotiation," in *Exploiting the Knowledge Economy: Issues, Applications, Case Studies*, vol. 3, pp. 44–51, 2006. 3.2.2, 4.2.2.2

-
- [46] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour, "Semantic WS-Agreement partner selection," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*, (New York, NY, USA), pp. 697–706, ACM, 2006. 3.2.3
- [47] S. Shrivastava, "TAPAS final report," 03 2005. 3.2.4
- [48] D. D. Lamanna, J. Skene, and W. Emmerich, "SLAng: A Language for defining Service Level Agreements," in *FTDCS '03: Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, (Washington, DC, USA), IEEE Computer Society, 2003. 3.2.4
- [49] V. Tasic, B. Pagurek, and K. Patel, "WSOL - A Language for the Formal Specification of Classes of Service for Web Services.," in *ICWS (L.-J. Zhang, ed.)*, pp. 375–381, CSREA Press, 2003. 3.2.5
- [50] V. Tasic, *Service offerings for XML web services and their management applications*. PhD thesis, Carleton University, Ottawa, Ont., Canada, Canada, 2004. Adviser-Pagurek, Bernard. 3.2.5
- [51] M. S. Memon, A. S. Memon, M. Riedel, B. Schuller, D. Mallmann, B. Tweddell, A. Streit, S. van de Berghe, D. Snelling, V. Li, M. Marzolla, and P. Andreetto, "Enhanced resource management capabilities using standardized job management and data access interfaces within UNICORE Grids," *Parallel and Distributed Systems, International Conference on*, vol. 2, pp. 1–6, 2007. 3.3.8
- [52] D. Snelling, "NextGRID SLA Schema," tech. rep., The NextGRID project, 2007. 3.4.1
- [53] P. Hasselmeyer, H. Mersch, B. Koller, H.-N. Quyen, L. Schubert, and P. Wieder, "Implementing an SLA Negotiation Framework," in *Exploiting the Knowledge Economy: Issues, Applications, Case Studies*, vol. 4, pp. 154–161, 2007. 3.4.1, 3.5.2.3, 4.2.2.1
- [54] G. Laria *et al.*, "Consolidated Report on the Implementation of the Application Support Services Layer," tech. rep., The Akogrimo Project, 2007. 3.4.2
- [55] I. Kotsiopoulos, "A lightweight semantic bridge between Clouds and Grids," in *Exploiting the Knowledge Economy: Issues, Applications, Case Studies*, 2009. 3.4.3
- [56] H. Munoz, I. Kotsiopoulos, L. M. V. Gonzalez, and L. R. Merino, "Enhancing Service Selection by Semantic QoS," in *6th Annual European Semantic Web Conference (ESWC2009)*, pp. 565–577, June 2009. 3.4.3
- [57] I. Jones and other members of the BREIN consortium, "Detailed Validation Scenario Definition - D3.1.2." Website: <http://www.gridsforsbusiness.eu>, September 2008. 3.4.3

- [58] I. Rosenberg and A. Juan, "White Paper: Integrating an SLA architecture based on components," tech. rep., The BEinGRID Project, 2009. 3.4.4
- [59] P. McKee, S. Taylor, M. Surridge, R. Lowe, and C. Ragusa, "Strategies for the Service Market Place," in *Grid Economics and Business Models*, pp. 58–70, Springer Verlag, 2007. 3.5
- [60] L. Clement, A. Hately, C. von Riegen, and T. Rogers, "UDDI Version 3.0.2," tech. rep., OASIS, 2004. 3.5.1.1
- [61] T. Banks, "Web Services Resource Framework (WSRF) ñ Primer v1.2," tech. rep., OASIS, 2006. 3.5.1.2
- [62] S. Graham, A. Karmarkar, J. Mischkinsky, I. Robinson, and I. Sedukhin, "Web Services Resource 1.2 (WS-Resource)," tech. rep., OASIS, 2006. 3.5.1.2
- [63] K. Ballinger, P. Brittenham, A. Malhotra, W. A. Nagy, and S. Pharies, "Web Services Inspection Language (WS-Inspection) 1.0," tech. rep., IBM, 2001. 3.5.1.3
- [64] V. Modi and D. Kemp, "Web Services Dynamic Discovery (WS-Discovery) Version 1.1," tech. rep., OASIS, 2009. 3.5.1.4
- [65] P. C. K. Hung, H. Li, and J.-J. Jeng, "WS-Negotiation: An overview of research issues," *Hawaii International Conference on System Sciences*, vol. 1, p. 10033b, 2004. 3.5.2.1
- [66] A. Andrieux, K. Czajkowski, *et al.*, "Web Services Agreement Negotiation Specification," tech. rep., Open Grid Forum - OGF, 2007. 3.5.2.2
- [67] K. Czajkowski, I. T. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," in *JSSPP* (D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds.), vol. 2537 of *Lecture Notes in Computer Science*, pp. 153–183, Springer, 2002. 3.5.2.5
- [68] M. H. Haji, I. Gourlay, K. Djemame, and P. M. Dew, "A SNAP-Based Community Resource Broker Using a Three-Phase Commit Protocol: A Performance Study," *Comput. J.*, vol. 48, no. 3, pp. 333–346, 2005. 3.5.2.5
- [69] A. Sahai, S. Graupner, V. Machiraju, and A. P. A. van Moorsel, "Specifying and Monitoring Guarantees in Commercial Grids through SLA," in *CCGRID*, pp. 292–, IEEE Computer Society, 2003. 3.5.2.5
- [70] M. G. Nabi and A. Nadeem, "A Formal Model for English Auction Protocol," in *SERA* (R. Y. Lee, W. Du, H.-K. Kim, and S. Xu, eds.), pp. 119–126, IEEE Computer Society, 2009. 3.5.3.1

-
- [71] FIPA, *FIPA English Auction Interaction Protocol Specification*. FIPA, 2001. 3.5.3.1
- [72] L. S. Bagwell, "Dutch Auction Repurchases: An Analysis of Shareholder Heterogeneity," *Journal of Finance*, vol. 47, no. 1, pp. 71–105, 1992. 3.5.3.2
- [73] M. Yokoo, "Generalized Vickrey Auction," in *Encyclopedia of Algorithms* (M.-Y. Kao, ed.), Springer, 2008. 3.5.3.3
- [74] P. Hasselmeyer, "On Service Discovery Process Types," in *ICSOC* (B. Benatallah, F. Casati, and P. Traverso, eds.), vol. 3826 of *Lecture Notes in Computer Science*, pp. 144–156, Springer, 2005. 4.2.2.1
- [75] B. Mitchell and P. Mckee, "SLAs A Key Commercial Tool," in *Innovation and the Knowledge Economy: Issues, Applications, Case Studies*, 2005. 4.2.2.2
- [76] M. Wilson, A. Arenas, and L. Schubert, "D63 - TrustCoM Framework V4," tech. rep., The TrustCoM Project, 2007. 4.2.2.2
- [77] B. Koller and L. Schubert, "Towards autonomous SLA management using a proxy-like approach," *International Journal of Multiagent and Grid Systems*, vol. 3, pp. 313–325, 2007. 4.2.2.2
- [78] P. Hasselmeyer, P. Wieder, B. Koller, and L. Schubert, "Added Value for Business through eContract Negotiation," in *Exploiting the Knowledge Economy - Issues, Applications, Case Studies*, vol. 5, pp. 641–648, 2008. 4.3.2.2
- [79] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, pp. 4–7, 2001. 4.4
- [80] M. S. U. HPCC, "Michigan State University High Performance Computing Center Service Level Agreement." Website. 5.3
- [81] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen, "RFC3060 - Policy Core Information Model – Version 1 Specification," tech. rep., Network Working Group, 2001. 5.5.5
- [82] I. Rodero, F. Guim, J. Corbalán, and J. Labarta, "Examples of JSDL 1.0 Documents for Parallel Jobs: BSC Use Case," tech. rep., OGF, 2006. 6.2.1
- [83] C. Glasner and J. Volkert, "An Architecture for an Adaptive Run-time Prediction System," *Parallel and Distributed Computing, International Symposium on*, vol. 0, pp. 275–282, 2008. 6.2.3
- [84] O. Corcho, P. Alper, I. Kotsiopoulos, P. Missier, S. Bechhofer, and C. Goble, "An overview of S-OGSA: A Reference Semantic Grid Architecture," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 4, pp. 102–115, June 2006. 6.3

- [85] P. Hasselmeyer, B. Koller, L. Schubert, and P. Wieder, "Towards SLA-Supported Resource Management," in *HPCC* (M. Gerndt and D. Kranzlmüller, eds.), vol. 4208 of *Lecture Notes in Computer Science*, pp. 743–752, Springer, 2006. 6.3.1
- [86] P. Masche, P. Mckee, and B. Mitchell, "The Increasing Role of Service Level Agreements in B2B Systems," in *WEBIST (2)*, pp. 123–126, 2006. 6.3.1
- [87] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003. 6.3.1
- [88] H. Mersch, P. Wieder, B. Koller, and G. Murphy, "Improving Business Opportunities of Financial Service Providers through Service Level Agreements," *Grid Middleware and Services*, vol. 1, pp. 397–408, 2008. ISBN: 9780387784465. 6.3.1, 6.3.2.1
- [89] S. Aiber, D. Gilat, A. Landau, N. Razinkov, A. Sela, and S. Wasserkrug, "Autonomic Self-Optimization According to Business Objectives," in *ICAC*, pp. 206–213, IEEE Computer Society, 2004. 6.3.1
- [90] P. Hasselmeyer, M. SurrIDGE, and P. Wieder, "NextGRID Registry Use Cases," tech. rep., NextGRID, 2008. 6.3.1
- [91] I. Müller, R. Kowalczyk, and P. Braun, "Towards Agent-Based Coalition Formation for Service Composition," in *IAT*, pp. 73–80, IEEE Computer Society, 2006. 6.3.1
- [92] K. Tserpes, L. Schubert, B. Koller, D. Kyriazis, and T. Varvarigou, "Learning from the Past - SLA Composition Using QoS History," in *Expanding the Knowledge Economy: Issues, Applications, Case Studies*, vol. 4, pp. 47–53, 2007. 6.3.2.2
- [93] A. D. Stefano, G. Morana, and D. Zito, "Advanced Reservation in Grid Using PBS," *Enabling Technologies, IEEE International Workshops on*, vol. 0, pp. 216–221, 2008. 6.3.2.2
- [94] O. Wäldrich, P. Wieder, R. Yahyapour, and W. Ziegler, "Improving Workflow execution through SLA-based Advance Reservation," in *Integrated Research in Grid Computing, CoreGRID Integration Workshop 2006*, (Krakow, Poland), pp. 333–344, October 2006. ISBN: 83-915141-6-1. 6.3.2.2
- [95] J. Li, W. Ziegler, O. Wäldrich, and D. Mallmann, "Towards SLA Based Software License Management in Grid Computing," Tech. Rep. TR-0136, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, June 2008. 6.3.3

- [96] C. Simmendinger, "Support for Client-Server based License Management Schemes in the Grid," tech. rep., BEinGRID, 2009. 6.3.3
- [97] D. Kuropka, P. Tröger, S. Staab, and M. Weske, *Semantic Service Provisioning*. Berlin, Heidelberg: Springer, 2008. 6.3.3
- [98] R. Lee and M. Waldburger, "Service Level Agreements," November 2001. 6.4
- [99] D. Snelling *et al.*, "An Introduction to the NextGRID Vision and Achievements V1.0," tech. rep., NextGRID, 2008. 6.4
- [100] P. Hasselmeyer, B. Koller, I. Kotsiopoulos, D. Kuo, and M. Parkin, "Negotiating SLAs with Dynamic Pricing Policies," in *Service Oriented Computing: a look at the Inside*, pp. 28–32, 2007. 6.6
- [101] S. Wesner, *Integrated Management Framework for Dynamic Virtual Organisations*. PhD thesis, University Of Stuttgart, 2008. 6.7