Generic Methods for Adaptive Management of Service Level Agreements in Cloud Computing

Dissertation
zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"
der Georg-August-Universität Göttingen

im Promotionsprogramm Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Edwin Yaqub
aus Lahore, Pakistan

Göttingen, 2015

Betreuungsausschuss

Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Jens Grabowski
Institut für Informatik, Georg-August-Universität Göttingen

Mitglieder der Prüfungskommission

Referent: Prof. Dr. Ramin Yahyapour,
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Korreferent: Prof. Dr. Jens Grabowski,
Institut für Informatik, Georg-August-Universität Göttingen

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Carsten Damm,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Dieter Hogrefe,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Konrad Rieck,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Stephan Waack,
Institut für Informatik, Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 29 Oktober, 2015

Acknowledgments

In the beginning, I thank God for His immense blessings and grace on me.

During my PhD, several people were involved at different times and locations. It is my privilege to recognize their role in accomplishing this thesis work. I would like to thank Prof. Dr. Ramin Yahyapour for giving me the opportunity to conduct research work at his chairs, first at TU Dortmund and later at the University of Göttingen. Over the years, his leadership, supervision and experienced insights helped me transform my research vision into reality. I am much thankful to my second supervisor, Prof. Dr. Jens Grabowski for his guidance. I am also grateful to Prof. Dr. Stephan Waack, Prof. Dr. Carsten Damm, Prof. Dr. Dieter Hogrefe and Prof. Dr. Konrad Rieck for participating as members of my defense committee.

Research work that led to this PhD thesis started in fall 2009 and was concluded by fall 2014. It has indeed been quite a journey and one which refreshes many memories when looking back. I wish to thank my team lead Philipp Wieder for taking care of administrative issues and assisting me in dealing with EU projects. Heartfelt thanks are due to Constantinos Kotsokalis for his mentoring, publication reviews and encouragement, and to Thomas Röblitz for his knowledgeable advices and kindness. I would like to thank all my colleagues at the GWDG for the nice work atmosphere, especially Kuan Lu and Ali Imran Jehangiri for our long discussions and late night tea times.

My friends provided moral and physical support as I moved from Düsseldorf to Dortmund and later to Göttingen. Maike Räkers, Rajmohan Shanmugavadivel, Ianthe Wijayarathna and Michael Harrington deserve special thanks for their love, trust and being there for me. May God bless you.

Support from my family has been second to none. I wish to thank my sister Edna and my brother Alfred for their motivation and prayers. I thank pastor Francis Pfister and Ann Pfister for support in a critical situation. I cannot forget my nephews Zivan and Zuriel and my niece Johanna for all the fun times. Finally,

I wish to express my deepest gratitude to my parents for their unconditional love and unwavering support. Words fall short to thank you enough for everything you did for me. I dedicate this thesis to my parents.

Abstract

The adoption of cloud computing to build and deliver application services has been nothing less than phenomenal. Service oriented systems are being built using disparate sources composed of web services, replicable datastores, messaging, monitoring and analytics functions and more. Clouds augment these systems with advanced features such as high availability, customer affinity and autoscaling on a fair pay-per-use cost model. The challenge lies in using the utility paradigm of cloud beyond its current exploit. Major trends show that multi-domain synergies are creating added-value service propositions.

This raises two questions on autonomic behaviors, which are specifically addressed by this thesis. The first question deals with mechanism design that brings the customer and provider(s) together in the procurement process. The purpose is that considering customer requirements for quality of service and other non functional properties, service dependencies need to be efficiently resolved and legally stipulated. The second question deals with effective management of cloud infrastructures such that commitments to customers are fulfilled and the infrastructure is optimally operated in accordance with provider policies.

This thesis finds motivation in Service Level Agreements (SLAs) to answer these questions. The role of SLAs is explored as instruments to build and maintain trust in an economy where services are increasingly interdependent. The thesis takes a wholesome approach and develops generic methods to automate SLA lifecycle management, by identifying and solving relevant research problems. The methods afford adaptiveness in changing business landscape and can be localized through policy based controls. A thematic vision that emerges from this work is that business models, services and the delivery technology are independent concepts that can be finely knitted together by SLAs. Experimental evaluations support the message of this thesis, that exploiting SLAs as foundations for market innovation and infrastructure governance indeed holds win-win opportunities for both cloud customers and cloud providers.

The research leading to this thesis was conducted as part of the EU FP7 SLA@SOI Integrated Project (grant agreement number 216556). It was continued at the GWDG and partially supported by the EU FP7 Project PaaSage (grant agreement number 317715).

Contents

Part I.

Introduction

Chapter 1.

Context

This chapter provides a condensed introduction to this thesis work. A motivating case for considering Service Level Agreements (SLA) in cloud computing is laid out. This is followed by a list of scientific contributions and a description of how this thesis is organized.

1.1. Motivation

The motivation behind the presented thesis is rooted in two factors. First, modern services have dependencies on other software and infrastructure level services that surpass multiple stakeholders. The primary question is that of trust between the producer and consumer of services. Service Level Agreements (SLA) are a plausible solution for parties at both ends of this dependence equation. An SLA provides a legal foundation to build trust, where contracting terms are unambiguously expressed along with agreed service levels and liabilities.

This makes SLAs indispensable for value creation scenarios where service providers collaborate to form service chains and business value networks. Here, automated negotiations can dynamically resolve SLA dependencies throughout the service hierarchy in accordance with the SLA requirements of customer. This compliments rising demands by government bodies and industry for automatic SLA (re)negotiation mechanisms, since the traditional take-it-or-leave-it SLAs do not accommodate diverse customer requirements and are designed to limit provider liabilities.

The adoption of clouds for service development and delivery has been phenomenal. However, due to the nascency of cloud markets, service offerings and business models are constantly evolving. Therefore, the markets are highly receptive to innovation opportunities. If cloud services are offered through feature-rich

negotiable SLAs, two side-effects result; i) customers get empowered in the procurement process, and ii) the unique negotiation mechanisms help to differentiate providers. Thus, SLA negotiation as a flexible business model to sell added-value services can positively disrupt cloud markets by fueling competition for adaptive service management. Additionally, this would offer a realtime possibility to dialectically maximize the business utility of SLA through intelligent negotiation strategies. However, in oder to be interoperable with multiple providers, software machinery would be needed to facilitate such collaborations in order to establish a foothold in multiple markets.

At this point, two research questions are identified for this thesis:

Question 1: Negotiations are steered by negotiation protocols. Given the fact that no single protocol can satisfy all negotiation scenarios and that protocol development must confront various complexities regarding modeling, design and implementation aspects, the first half of this question is about realizing a methodology which leads to developing multiple protocols in a standard manner. Although this helps to diversify selling mechanisms, the other half of this question is concerned about seamlessly executing the developed protocols across the two end points. Thus, an execution platform is necessary which allows loose integration with the underlying business systems.

Question 2: The objective of negotiating parties is to converge on an acceptable SLA. Viewing SLAs in light of their business utility has not received sufficient attention in prior art. This question aims at negotiation strategies in order to maximize the business utility of an SLA. It is obvious that various stakeholders may have different preferences for different SLA properties. Thus, reducing their utility or SLA-gap in short time through concession making negotiation strategies needs to be analyzed.

The second motivating factor for this thesis work is the governance role SLAs can play to reliably manage the cloud infrastructure. Cloud providers face inflated expectations from customers for always available and high performing resources. To deliver their SLA commitments, providers are constantly challenged to balance resource utilization, contention, energy use, migrations and other machine related costs. These are further complicated by service-driven constraints, which business applications impose regarding performance and availability.

Here, an SLA can be treated as a live artifact that is consulted throughout a service's lifecycle. For example, delivered quality can be frequently compared to

committed service levels and allocations can be regulated to ensure a fair share of system resources. Such operational decisions can be optimized considering SLA constraints and executed in line with provider's high level business policy.

This leads to the third research question that this thesis addresses:

Question 3: This question focuses on the service consolidation problem, where both machine level and service level constraints are considered when creating consolidation plans. Such an advanced SLA-aware resource management method is required to systematically handle large scale infrastructures and deployments.

Thus, automated SLA management can exploit the true potential of cloud by affording providers multiple operational competencies. This entails improved customer satisfaction and increased return on investment for providers. However, from the software engineering perspective, another question needs to be adressed:

Question 4: Cloud systems are inherently based on three service deployment models; the Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). These are often coupled at intra- or inter-organizational levels. It needs to be examined how business and operational planes can be cohesively related in layered cloud architectures, with the intention to integrate SLA management features such as the ones presented above.

The contributions of this thesis offer a synthesis on using SLAs as dependable tools to innovate service propositions and selling mechanisms, as well as controlling quality through conformance based management of cloud infrastructures.

1.2. Thesis Contributions and Organization

As research goals, this thesis developed methods for automating SLA management having three intrinsic qualities. First, the contributions are generic in that they are domain-independent and support wider applicability. For instance, the developed negotiation solution was demonstrated for multiple use cases of the EU project SLA@SOI and later reused in the EU project Contrail.

Secondly, the engineering and algorithmic contributions provide policy based controls at different levels of granularity. This is necessary to tailor the developed methods according to business-specific directives, particularly for personalizing negotiation sessions, associating utility to contract spaces, or making multi-criteria optimizations regarding infrastructure operations.

Thirdly, the contributions allow to adapt to changing business landscape.

Adaptiveness is delivered at many levels of SLA management; whether responding tactically to the conceding behavior of opponent during time-constraint SLA negotiations, supporting or developing new negotiation protocols to receive market traction, renegotiating SLAs to align with updated requirements, or consolidating cloud infrastructure to adjust to demand fluctuations.

This thesis is organized in three parts. The first part introduces the background, focus of this work and research problems to address. The second part presents the developed solutions and their novelty with respect to prior art. The third part concludes this thesis. As a summary, the scientific contributions and their organization in this thesis are listed below:

- **Contribution 1:** A generic methodology that leads from design to implementation of negotiation protocols as verifiable, reusable and machine executable artifacts. This is demonstrated by developing a bilateral negotiation protocol. Further, a negotiation platform is developed for point-to-point execution of negotiation protocol(s). The platform is used for chained or nested negotiations in distributed SLA scenarios with multi-provider dependencies. The protocol development methodology, the protocol and platform are presented in Chapter 5. This solves the first research question.

- **Contribution 2:** Two negotiation strategy algorithms are developed. These maximize the business utility of an SLA by adapting to the concession making behavior of negotiating partner and traversing contract spaces in short time. The strategies and their evaluations are presented in Chapter 6, providing a solution to the second research question.

- **Contribution 3:** An SLA-aware resource management solution is developed which focuses on Platform as a Service (PaaS) cloud infrastructures. The solution applies four Metaheuristic search algorithms to plan and optimize resource reallocation in a recurring "service consolidation" problem. A variety of soft and hard constraints regarding deployed services and cloud machines are considered. Simulations are performed to evaluate performance against a variety of quality metrics and solutions are ranked against high level policies. These results are presented in Chapter 7, which solve the third research question.

- **Contribution 4:** A prototype architecture based on the SOCCI [23] standard is realized. This provides a unified and extensible solution to man-

age layered cloud infrastructures and incrementally introduce selected SLA management features. This system is presented in Chapter 8 and provides a solution to the fourth question.

A summary on the output of this thesis and future prospects are presented in Chapter 9, which concludes this thesis.

Chapter 2.

Background

This chapter presents the background on service computing and cloud computing paradigms, to explain some concepts and terms referred later in this work. It also introduces Service Level Agreements (SLA) around which the main motivation of this work revolves. The following chapter builds upon this background to elaborate how SLA management in cloud computing is envisioned in this work.

2.1. Services Computing

The Information Technology Infrastructure Library (ITIL) provides a best practice guidance framework for service management which has matured over many years. ITIL defines a service as "a means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks" [1]. To deliver services to customer, a provider must have specialized management capabilities to deal with the software, its dependencies and infrastructure requirements. These capabilities must be utilized such that the business objectives of customer are achieved while the ownership of service constituents could be decoupled among service delivery stakeholders.

From the perspective of digital services, service oriented computing has been widely used as a paradigm to develop interface based services that are loosely coupled together to realize business workflows. This approach helps in building modular systems composed of diverse and replaceable services, which access data sources over the internet. Service oriented architectures (SOA) have emerged as popular solutions to build reliable systems where services can be advertised, discovered and accessed irrespective of their geographic boundaries. Towards this, web services are a successful approach to build complex systems using XML based data envelops that are exchanged between the client and the server using

application layer protocols such as SOAP. More recently, there is an increased use of representational state transfer (REST) protocol to build disparate services that exchange data in a session-less manner using lightweight formats such as JSON, plain text, atom, etc. Service oriented computing enables complex service design through two design patterns namely service composition and service aggregation.

Service composition deals with developing a service which is further composed of one or more services. The composing service's functionality is incomplete without the availability of the composed service. Service aggregation deals with bundled services which do not necessarily depend on each other. In both cases, service dependencies appear as a core design feature. When the ownership of this dependency is decoupled among various service providers or different departments of the same provider, various concerns need to be formally and legally addressed before production services are built. These relate with the functional and non-functional properties (NFP) of involved services.

Due to the various possible correlations, the price of a service may vary depending on the expected values of these properties. An important class of NFP is the quality of service (QoS) e.g., the availability and performance related metrics. Two service integration approaches are observed to realize service composition or aggregation; orchestration and choreography. Orchestration defines a centrally determined approach to access services in order to realize a business workflow. Choreography deals with coordination sequence where a service individually accesses its dependencies. QoS-aware services, their composition and aggregation has generated a lot of interest among research communities working on enterprise systems, grid computing and recently cloud computing.

2.2. Cloud Computing

In this work, the service paradigm of focal interest is cloud computing. Cloud computing provides on-demand and self-service network access to shared pool of computing resources that can be provisioned and released rapidly with minimal management effort [105]. Various features inherent to clouds raise management challenges with direct impact on the business. In essence, cloud computing applies service orientation to manage infrastructure resources through software services. These are exposed as Application Programming Interfaces (API) and have revolutionized how infrastructure is consumed and managed, from the perspec-

tives of both customer and provider.

This as-a-service paradigm is achieved by virtualization technology that provides an abstraction (hypervisor) layer on physical machines. Virtualization exists in many forms and is applied to compute, storage and network functionalities to pool heterogeneous resources. These are utilized by provisioning virtual resources e.g., virtual machines (VM), operating system level containers and software defined networks. Virtual resources can be rapidly provisioned as multiple tenants which share the underlying machines in a securely isolated manner. Another consequence is the mobility of virtual resources which allows to consolidated workloads on fewer machines to improve resource utilization and save energy.

By design, cloud computing enables elastic use of computing resources on a consumption based "pay-per-use" billing model. This fair model scales along demand and the corresponding revenue generated by the deployed service. As the technology matured, clouds have succeeded in delivering computing as a utility. Services deployed on cloud benefit from high availability as these can be spread across multiple geographic zones and regions. Further, load can be balanced through auto-scaling of resources in a horizontal or vertical fashion. On top of these powerful facilities, cloud offers consumers varying levels of management capability through three service models. These are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [105].

The service models differ in the level of control and administrative burden on the provider and consumer. IaaS consumers have maximum control to instantiate virtual machines and storage. However, the overhead lies in configuring application components and managing large amount of VMs. The PaaS targets accelerated development of applications and their lifecycle management. Consumers benefit from software environments, libraries, databases and middlewares delivered as add-ons to (provider-managed) Linux containers. On the contrary, SaaS completely abstracts consumers from the resource level view.

The economic advantage of using public cloud lies in low capital expense (Capex) and operational expense (Opex). Individuals or small and medium enterprises do not need to invest large capital to buy server capacity. Clouds facilitate in quickly proliferating innovating offerings in the market. This reduced time to market may be vital in establishing a leadership position. Generally, costs increase in the three service models in a bottom up fashion, i.e. IaaS is less costly than PaaS which is less costly than SaaS. Generally, savings also accumulate bot-

tom up. Providers who understand or foresee computing requirements of their business can buy relatively inexpensive reserved instances instead of on-demand (expensive counterparts) with an upfront cost-effective payment. The costs of on-demand instances from public cloud may exceed the amortized costs for private cloud. Hence, the decision to opt for public or private cloud requires careful financial analysis in order to maximize ROI. Sometimes, a hybrid private-public cloud is the preferred deployment model.

Amazon Web Services - a market leader and pioneer cloud computing company also offers spot instances. These are volatile resources offered through a bidding mechanism and reclaimed based on the demand and supply situation on the cloud. A market of fault-tolerant applications has resulted around spot instances. Predicting the acceptable bidding price for Amazon spot instances has spun up much research due to its cost saving potential. Spot Instances have set a precedence to sell cloud services through negotiation mechanisms, which is now an established research area.

Cloud computing is reshaping market relationships. Although cloud providers compete through differentiated offerings but simultaneously, public clouds transform past rivals into allies that collaborate to offer added-value services in win-win scenarios. As cloud-based services proliferate the market, customers are faced with new challenges of filtering the market space to procure or aggregate services that meet quality objectives and cost constraints. To deal with these complexities, the role of cloud brokers and marketplaces is gaining importance.

Providers exploit cloud computing from the business aspects of competition and collaboration. However, the customers are concerned with service customization and reliable delivery especially considering that service hierarchies or choreographies often span multiple providers and/or clouds. Therefore, we address these concerns by means of negotiated Service Level Agreements (SLA) to procure cloud services. To further elaborate our rationale, some trends regarding cloud adoption in current and emerging markets are presented next.

2.2.1. Trends: Multi-domain Synergies and Interdependencies

Various paradigms are rapidly evolving the service based markets, while new markets are forming primarily due to multi-domain synergies. Interestingly, cloud computing provides the technological basis in most of these paradigms.

An increasingly popular trend among cloud providers is the chaining of SaaS,

PaaS and IaaS providers. This is mostly because: i) the PaaS eases deployment of SaaS instances and ii) the PaaS clouds are a federation of virtual resources from multiple IaaS providers which are dynamically scaled based on the consumption by SaaS customers.

A recent IDC study predicted that by 2020, nearly 40% of the information in the digital universe will be touched by clouds, referring to data storage and processing performed in clouds [3].

Mashups are websites where data and services from various sources are composed and aggregated to provide rich graphical services e.g., by reusing third-party APIs for maps, spreadsheets or analytics. This is complimented by the fact that an increasing number of providers are exposing their data or services as open APIs to increase their customer base [5]. A popular open API is of Facebook social network that connects 1.23 billion monthly users [7]. It allows third-party developers to write and embed applications with the Facebook platform that is completely cloud based. AppStores are user-friendly lucrative channels to supply software to a large consumer audience. Garter estimated 102 billion software downloads in 2013 from mobile AppStores alone and expects more than 268 billion by 2017 [4]. Forrester notes that many AppStores are an embodiment of public cloud services [10]. Internet of Things (IoT) is an emerging paradigm to connect devices and goods with business operations. The IoT is aligning with cloud computing to perform analytics on sensed data [19].

IDC reports that clouds are enabling technologies from Social, Mobile, Internet of Things and Big Data paradigms to join forces, create value and shape new markets [2]. The US cloud computing strategy heavily argues on migrating federal government IT to clouds with upto 20 billion $ investment [15]. These trends indicate that cloud based services having software and infrastructure level interdependencies are set to flourish. This raises interest in SLAs for building reliable and adaptive systems that are appropriate for commercial purposes.

2.3. Service Level Agreements (SLA)

An SLA is a contract created as a result of negotiation between the provider of a service and its customer. SLA formally defines a service, its functional properties and agreement terms on non functional and quality aspects of the service. The provider is obliged to deliver these service levels, given that preconditions or

usage constraints are respected by the customer (if any).

Functional properties describe the capability/operations of the service along with access information such as supported protocols or endpoint references. Non functional properties deal with the QoS such as availability and performance. Service usage constraints may specify bounds on concurrent sessions, throughput or request payload. An increasing number of terms that require a legal cover are being considered for SLAs. These include security, data protection, policy rules for auto-backups and auto-scaling, parameters related to monitoring mechanism, SLA violations, reward/penalty functions, level of customer support, service ownership and continuity [98]. Thus, an SLA addresses an intersection of technical and business concerns. In order for an SLA to be enforced and tracked for compliance, SLA terms and parameters should be reasonable, attainable, enforceable, measurable and objective [17].

2.3.1. EU Research Project SLA@SOI

Work leading to this thesis started under the SLA@SOI [93] project. Therefore, its various modalities and novelties shall be referred to in various sections of this thesis. SLA@SOI envisioned, "a business-ready service-oriented infrastructure empowering the service economy in a flexible and dependable way [93]".

Project Motivation

The project was motivated by the ongoing evolution towards a service oriented economy where IT-based services can be flexibly traded as economic goods. To realize such a high degree of automation in a dependable and predictable manner, the project emphasized on dynamic provisioning of services which must be governed by a comprehensive SLA management framework. Various usecases from leading industrial partners such as SAP, Intel, Telefonica and several universities, led to the realization of an SLA management (SLAM) framework. This allowed lifecycle management of complex services based on formally specified SLAs and cohesively brought together a host of sub-systems e.g., for service design, template definition, advertisement, discovery, negotiation, provisioning, monitoring, adjustment and termination. The project is accredited with substantially extending state of the art and also influencing contemporary research projects in similar direction. Here it is considered worthwhile to present some core require-

ments which SLA@SOI identified and addressed to achieve its goals.

Project Requirements

Multi-level SLA management: Providers are keen on offering differentiated services and this usually translates to value networks among multiple providers. The top level business perspective demands that software services can be offered according to unique functional and non-functional requirements of the customer. This requires business services and their component level dependencies to be predictable so service customizations can be performed by negotiating top level SLAs. However, even within a single provider domain, IT stacks are usually composed of multiple layers. This effects the delivery capability of a service provider. Hence, an SLA management layer is needed at different tiers of a provider's IT landscape to enable a dependable mapping of functional and non-functional properties from the software level down to infrastructure level metrics.

Negotiation driven adaptations: These capabilities need to be systematically designed and integrated at runtime to offer QoS-aware services that can be customized through runtime (re)negotiations. Service level dependencies may exist at intra- and inter-organizational levels as value added services are usually created by realizing synergies. The SLA Management framework used models to i) represent a service landscape which is usually internal to a provider, and ii) service definitions as a negotiable SLA template. The latter is used to tailor SLA offers which are exchanged between customer and provider framework instances during the negotiation process. To enable market competition, the framework supports providers to sell differentiated service offerings using privately or publicly developed negotiation protocol(s), which can be executed using the negotiation platform provided by the framework.

Provisioning, Monitoring and Adjustments: SLAs are legal artifacts and must be enforced. Thus, service dependencies (both at the software and infrastructure levels) need to be monitored at service execution time. This requires that the formal definition of SLA terms be processed in alignment with the monitoring sub-system of the provider. This is further necessary to build advanced analytics and adjustment functionalities so that SLA violations can be either proactively avoided by readjusting resource/service instance(s), or the root cause of violations is determined to avoid such failings in future. For this, the framework leveraged feedback control loops and complex event processing

techniques, while components for planning, optimization and adjustment were realized to make autonomic decisions and perform corrective measures in line with provider's business policies.

Formalizing SLA: A key innovation was a machine readable SLA model, which served as the basis for automation and interfacing the different modules of the SLAM framework. This is briefly presented next.

The SLA Model

Past works such as the WSLA and WSAG [55, 56] tightly coupled the modeling capability to the XML format. Considering these shortcomings, the SLA model in SLA@SOI was designed as an abstract syntax which can be rendered in different concrete formats e.g., Java, BNF (Backus Norm Form) or XML. The model provides domain-independent constructs to create an SLA template (SLAT) for a service. The SLAT serves as a blueprint where QoS terms are listed along with broad value ranges in order to increase contracting possibilities. Specific SLAs are created from SLAT by fixing ranges to single values as a result of negotiation between the customer and provider.

The model provides semantics to represent primitives, expressions and descriptions. These are used to specify a service's interface, terms for QoS, events (such as violation of term) and actions (for reporting or recovery). Third party vocabulary libraries can be plugged-in to foster a common understanding of domain-specific QoS terms. The technical details of the model are presented in [58]. Fig. 2.1 shows a high level view how the model structures an SLAT (or SLA). The UUIDs refer to the unique identifier for the SLA and that of the corresponding SLAT. The validity period of an SLA represents the time when it is effective and the service is in use. The section on parties documents contact information of obliged parties. The variable declarations are used to represent expressions. An expression can be seen as a function with parameters. Variables allow to share expressions in subsequent sections. The interface declarations specify the functioning interface of a software service or a resource e.g., a virtual machine.

Agreement terms define the QoS states to be guaranteed. These are expressions specified as logical constraints on service level objectives (SLOs). An SLO is a quality metric such as availability or response time. Additionally, any precondition or usage constraint regarding the agreement term can also be specified e.g., response time is guaranteed provided a certain throughput is not exceeded.

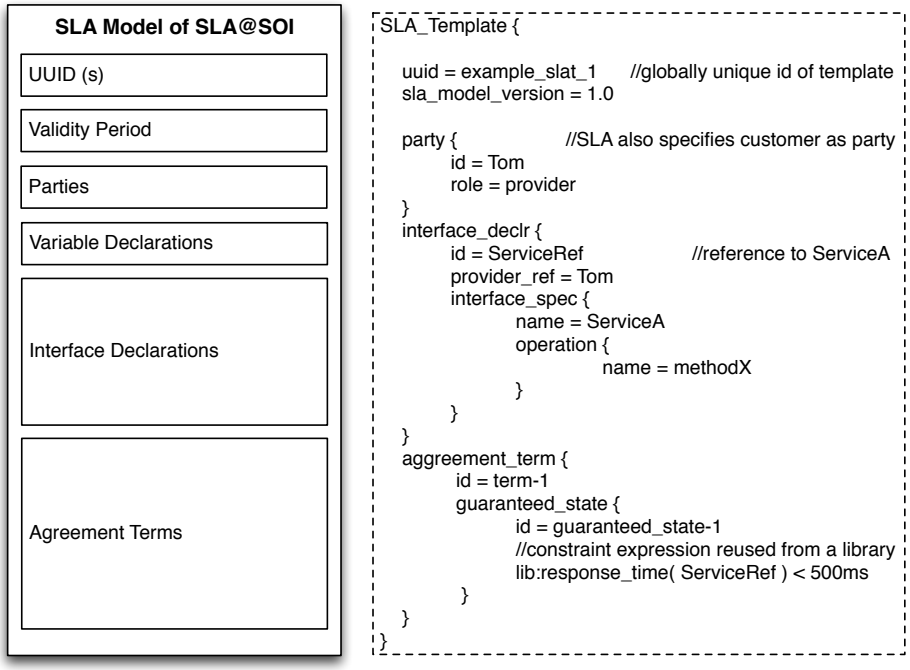


Figure 2.1.: SLA Model and an SLA Template defined using its BNF Syntax

Finally, terms can also define penalties for violation of a guaranteed term.

The SLAM framework was realized using SOA architecture, which standardized various interfaces and components to holistically address the mentioned features. Its architecture would be presented in line with author's work in Chapter 5. The next chapter deals with the role of SLA management in cloud computing.

Chapter 3.

SLA Management in Cloud Computing

This chapter introduces SLA management in cloud computing. The motivation and rationale for this is strengthened by market trends and the two perspectives identified on the SLA lifecycle. These reveal opportunities for maximum exploitation of cloud infrastructures through SLAs and the challenges faced. This helps to formulate research problems presented in the next chapter, along with an overview of related work and the novelty of own contributions.

3.1. SLA Lifecycle

Establishing and complying with SLAs is non-trivial in large scale dynamic systems such as clouds. Besides technical issues of automation and control, organizations require additional SLA management processes. These have been thoroughly studied in ITIL volumes on service management. The Telemanagement Forum (TMF) has addressed SLA management in telecommunications industry [11], which has since been applied to cloud computing [13, 17].

Based on these works, SLA management is defined as systematic activities that are linked together in a management lifecycle called an SLA lifecycle. The SLA lifecycle governs a service instance from its inception to provisioning, adjustments over time and termination. Fig. 3.1 shows the SLA lifecycle given in [11] and used as a reference in this work. Earlier works by Sun, IBM and HP also structured similar lifecycles [37]. In our context, SLA management implies automated instrumentation of SLA lifecycle phases. This is a challenging task since there exist various research and engineering problems in different phases. More recently, the lifecycle management of cloud services is being reshaped around SLAs. To this, the TMF report on cloud SLA management highlights that, "the SLA definition, SLA policy and SLA negotiation management must be flexible

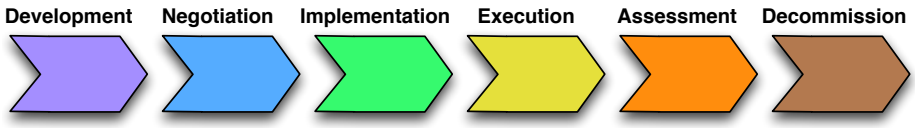


Figure 3.1.: SLA Lifecycle [11]

enough to support the cloud ecosystem operation; static and dynamic SLA management, SLA negotiation and re-negotiation must be considered " [12]. This provides an industrial impetus to the direction of this thesis work.

As shown in Fig. 3.1, the SLA lifecycle consists of six phases which are explained next. In development phase, a service offering is planned and its SLA template is created. This requires benchmarking service levels considering software and infrastructure resource dependencies. Performance engineering methods are employed to translate expected service quality to non functional and quality requirements from its dependencies. The quality terms offered to the end customer are structured along with their value ranges in an SLA template. SLA templates can be advertised over a public repository or a publish/subscribe advertisement system. Interested parties subscribe or search for the kind of services based on metadata or semantic annotations provided on templates.

Once discovered, the SLA template is used to conduct negotiation during which the customer and provider exchange SLA offers to converge on a mutually acceptable SLA. In the implementation phase, the established SLA determines the provisioning of appropriate infrastructure and/or platform resources for the procured service. The SLA enters the execution phase, when the service goes in effect. During execution, the service instance is regularly monitored to avoid SLA violations. If necessary, corrective actions may be preemptively taken to maintain seamless delivery of quality guarantees. Assessment allows to revise service offerings or update delivery capabilities for future improvements. This may also take into account marketing factors, such as competition with other provider offerings, adapting own negotiation strategy. Finally, an SLA is decommissioned upon maturity or if an SLA offering is to be revoked. As a result, the allocated resources are released. Even during service, an SLA may be renegotiated if required by changes in demand and supply or business policy.

This thesis makes scientific contributions towards implementing SLA management, whereas ITIL and TMF provide the necessary theory and organizational

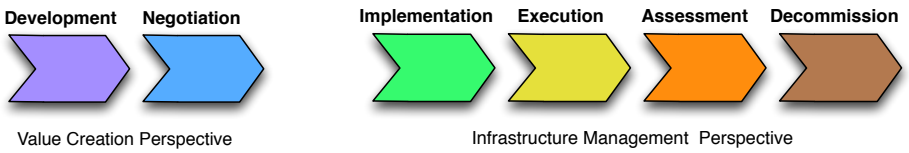


Figure 3.2.: Two perspectives on the SLA lifecycle

processes. Thus, many gaps between theory and practice are reduced with due consideration to prior art.

3.2. Opportunities and Challenges

Trends (presented in Section 2.2.1) indicate that clouds are set to trigger competition among providers to create value-added service propositions, novel selling mechanisms and intensify demands for reliable service delivery. In this context, clouds incarnate a self-reinforcing Krebs cycle, where they catalyze innovation which in turn increases their demand even more [8]. Growth in such large dynamic systems is best managed through automated and autonomic techniques that control the entire service lifecycle [9]. Management of cloud based services is finely related with the management of cloud infrastructure. This duality holds many innovation opportunities, which nevertheless also embody challenges. We expose these opportunities and challenges by presenting two key perspectives on the SLA lifecycle as shown in Fig. 3.2, namely:

- Value Creation Perspective.
- Infrastructure Management Perspective.

3.2.1. Value Creation Perspective

This perspective deals with the marketing opportunities for SLA-based cloud computing and comprises the first two phases i.e., service development and SLA negotiation. Much focus in prior works has been paid to technical aspects such as the ones we address in the infrastructure management perspective, yet it is the somewhat neglected value creation perspective that determines a provider's

position or share in the cloud market. Value is usually derived when service providers collaborate in a service chain, which may have a complex hierarchy. This leads to business value networks where providers act as producer or consumer of a service. NESSI (a European consortium of over 300 ICT companies and research community focusing on networked services) highlights value networks as viable business models for ICT infrastructures [14]. ICT providers such as cloud providers are keen to explore sustainable business models to maximize infrastructure use, so profits can be accumulated over amortized investments.

Added-value service propositions represent composite or aggregated services spanning multiple providers and administrative domains. The fundamental issue is to configure the service instance by resolving SLA dependencies throughout the service chain according to the SLA requirements of the customer. Here, negotiation is an acclaimed business model to dynamically engage a customer and the string of provider(s), in order to resolve SLA dependencies among background services and infrastructure resources. Besides collaboration, negotiation mechanisms serve to differentiate providers and may lead to a monopoly in certain market or market segment, such as enjoyed by Amazon incase of spot instances.

In refined terms, value creation demands methods to develop and investigate negotiation mechanisms that engage stakeholders in accommodating interactions to customize a service, resolve QoS dependencies or preference conflicts. Such methods would be fundamental in nascent cloud markets where IaaS, PaaS and SaaS based providers are already chaining together to quickly proliferate added-value services. Current offerings however lack the notion of negotiable SLAs, but the prevalent take-it-or-leave-it SLAs do not meet demands to customize services by mutually agreeing on service level guarantees and liabilities [15, 98].

Automated negotiations, steered by negotiation protocols and complimented by intelligent negotiation strategies can address this challenge. Therefore, in this work, value is created by means of negotiation protocols and negotiation strategies. Together, these tools can sustain competitive advantage for cloud providers. Negotiation protocols not only facilitate collaboration between parties, but can also fuel competition among providers by diversifying selling mechanisms. This gives providers the much coveted differentiating factor and can lead to disruptive new service propositions with negotiable SLAs replacing the rigid take-it-or-leave-it SLAs. However, since a single protocol may not be used in all scenarios, therefore the challenge is to develop and operate shared protocols in

a seamless manner at all ends of the service chain. Similarly, no single strategy outperforms others in all negotiation scenarios and test-and-trial is the usual approach to estimate outcomes in terms of individual utility and social welfare of the overall market.

Negotiation driven value creation ultimately envisions an open, distributed and inter-operable system of marketplaces. Ideally, a party should have the means to establish a foothold in multiple markets and create SLAs whose business worth is more than the best alternative to no agreement (BATNA) [25].

3.2.2. Infrastructure Management Perspective

The infrastructure management perspective comprises the phases of implementation, execution, assessment and decommission of SLAs. Arguably, the most important objectives of cloud providers are cost savings and customer satisfaction. Upon closer inspection, infrastructure management perspective reveals a blend of technical, ecological, economical and customer relationship aspects.

Infrastructure management primarily aims at improving machine utilization. It is well known that data centers commonly utilize only 10-20% of their server resources [124] while clouds can increase utilization upto 70% [15]. Low utilization wastes energy, which is economically not viable and raises ecological concerns. In 2010, data centers consumed between 1.1%-1.5% of global energy use [9]. However, improving utilization to save power costs risks over provisioning, which degrades availability or performance of deployed services. The International Working Group on Cloud Computing Resiliency (IWGCR) reported that in period 2007-12, major cloud providers served an average of 99.9% availability, which amounts to 7.5 hours of unavailability per service per year. Many industry adopters remain unaware of this fact [16] and demand more (performance related) QoS guarantees such as response times, latency or throughput [73].

Thus, utilization needs to be improved considering the vital notion of SLA violations, which must include performance degradation due to contention on resources as well as due to migrations. The former can assess if system resources such as CPU, memory, disk or network bandwidth are being utilized beyond a safe limit as this leads to machine failures. The latter can estimate unavailability or performance compromises caused by migration of deployment units belonging to a service e.g., virtual machines or containers. Moreover, modern clouds are built to target multiple availability zones. However, such large scale of cloud,

machine and software heterogeneity, a blackbox view of deployments and usage fluctuations add to the complexity of infrastructure management.

State of the art cloud stacks such as OpenStack (IaaS) and OpenShift (PaaS) provide automated capabilities for provisioning, auto-scaling and migration of deployment units, but advanced control features for consolidating resources from the perspective of SLA aware services and the business policy of provider are not available to date. For SLA management to be realistically introduced on top of a specific cloud stack, it needs to beware of technical underpinnings so resources are (re)allocated with due consideration to topology, configuration and capacity constraints of machines as well as placement constraints among services.

The infrastructure management perspective demands optimization methods and tools using which cloud infrastructures are optimally operated and SLA commitments are satisfied. Such capabilities enable cloud providers to save costs, enhance profits and maintain good reputation with customers. A cumulative effect of thoroughly investigating and addressing this perspective shall determine the eventual return on investment (ROI) for cloud providers.

3.3. Related Work

This section presents recent SLA related projects, highlighting their key features.

The 4CaaSt project [26] considered a marketing view of business services and their dependencies, which can be represented in a "blueprint" document. For this purpose, a description language is developed. Provisioning requirements regarding multi-tenancy and scaling can also be expressed. Services are exposed through a marketplace environment. Based on an end customer's high level business requirements, a simulation tool maps them to low level resource parameters. This helps to realize SLA requirements or template definition.

Project Cloud4SOA [27] developed semantics based approach so PaaS developers can express QoS requirements of applications and cloud providers can express their infrastructure capabilities. This allows matchmaking and WS-Agreement [56] based negotiation is employed to create an SLA.

Project IRMOS [28] addressed configuration and adaptation concerns of real time interactive applications deployed on PaaS and IaaS clouds. IRMOS uses an extended version of WSAG negotiation protocol [56], which repeatedly requests new SLA offers from a dependent provider by submitting a proposal. It thus

adds a quotation submission interaction as in contract net protocol (CNP) [59] to circumvent the take-it-or-leave-it shortcoming of WSAG protocol [29]. The project emphasized on SLA adaptation through renegotiations, SLA translation problem which maps application QoS levels to low-level resource requirements [30], and SLA enactment through its monitoring framework.

The Optimis [31] project focused on engineering, deployment and control of service lifecycle on clouds. It extended WSAgreement protocol to negotiate SLA based resources in a host of procurement scenarios including private-public cloud bursting, federated clouds and multi-cloud deployments. Functional and non functional requirements of a service are expressed as a description language called the manifest [32]. This includes virtual machine specification from preferred infrastructure providers, affinity constraints, legal terms and quality levels.

Project Contrail [33] developed a cloud management platform which uses resources from a federation of IaaS cloud providers. The project used the SLA model of SLA@SOI project but extended SLA terms to link to resources e.g., virtual machines, which can be scaled up when specified thresholds are reached. Contrail also used the SLA negotiation system of SLA@SOI to implement its federation layer. This allows Contrail customer to negotiate an SLA with the federation which acts as a broker and negotiates SLAs with appropriate provider(s) in the federation on customer's behalf.

The SLA@SOI project [93] thoroughly researched SLA lifecycle management in service oriented infrastructures such as clouds, based on formally specified negotiable SLAs [58]. A key outcome of the project was a generic SLA management framework that allows to introduce SLA management on IT stacks [63]. It comprised of generic and extensible components for domain-specific adaptations. The framework demonstrated SLA modeling, template based discovery, multilevel and multi-domain SLA negotiations, provisioning, monitoring and adjustment of resources in a wide range of industrial use cases.

Chapter 4.

Requirements and Problem Description

This chapter first presents three motivating use cases, which provide main requirements for the research questions identified earlier in Chapter 1. Based on this, problem descriptions are presented in a refined manner. Finer granularity requirements for each of these problems, and the novelty of proposed solutions with respect to prior art is presented in subsequent chapters.

4.1. Use cases

The use cases highlight SLA scenarios where dependencies need to be resolved at multiple levels or multiple domains. The latter creates added-value propositions with devolved ownership of risk. Two use cases from the SLA@SOI project and one use case originating from the author's work at GWDG are described.

Retail as a Service

This SLA@SOI use case was developed as an open reference case to establish the feasibility of the SLA framework in retail chains. The main proposition here is a transaction-heavy retail solution offered as a service by an IT provider who specializes in supermarket operations. The service is composed of an inventory control and a payment processing sub-service. These are invoked when customers buy goods at the cash desk of a supermarket store.

A supermarket having a chain of stores negotiates an SLA with the service provider to procure an instance of retail service tailored to its non functional requirements e.g., request load at stores and transaction completion time. The scenario is shown in Fig. 4.1. The software components of this service are developed and managed by the software department of the provider. Thus, the

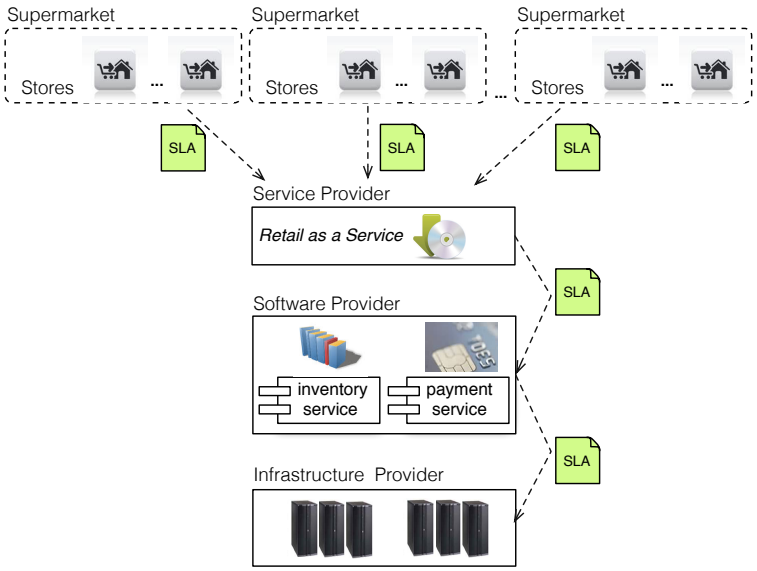


Figure 4.1.: SLA Hierarchy in Retail Chain Scenario

customer-facing business department depends on the software department to translate the requested quality levels onto the software configuration. Software performance also depends on the infrastructure e.g., the capacity or number of virtual machines used to deploy the software. Thus, a further dependency on the infrastructure department exists to provide the required infrastructure resources. This reveals a multi-level SLA hierarchy, where provisioning decisions nevertheless lie in the same provider domain. The SLA framework deployed at each level of this hierarchy is used to conduct SLA negotiations to affix dependencies as SLAs and provision resources. The use case is applicable to service procurement in many network industries such as hotels, railways, airlines or electric grids.

Aggregated Services in Telecommunication

This SLA@SOI use case was led by Telefonica and Telecom Austria. With the advent of wireless networks and internet based telephony services from new and more agile market players, traditional telecommunication companies face the challenge of creating new services that maximize the utilization of their existing infrastructure capabilities. A viable solution is found in aggregating these capabilities to create new offerings. This is achieved by writing web service wrappers for services such as voice over IP (VoIP), short messaging service (SMS) and

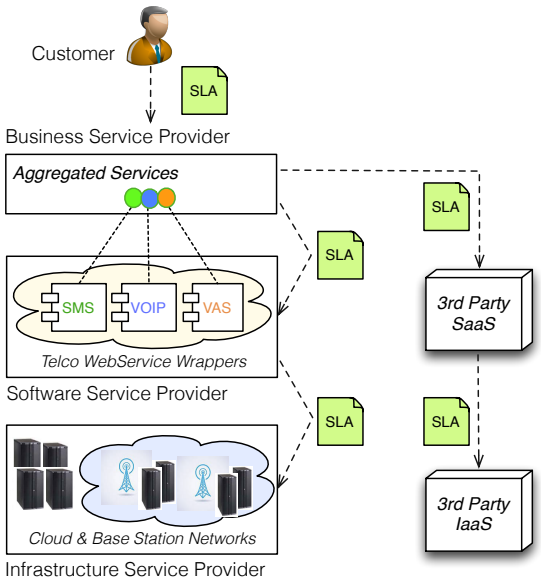


Figure 4.2.: SLA Hierarchy in Aggregated Services Scenario

voice application services (VAS). However, these aggregations need to be flexibly established together by means of SLA negotiation. This empowers a customer to configure an added-value business product, while remaining blisfully unaware of the complex interdependencies that exist between software and infrastructure.

As shown in Fig. 4.2, the use case represents a multi-domain SLA scenario, where providers implement specific extensions of the SLA framework to deal with local provisioning and external outsourcing decisions. This way, business-to-customer (B2C) negotiations trigger nested business-to-business (B2B) negotiations among dependent providers, including possible third party SaaS and IaaS providers. To control frequent (re)negotiation triggers, negotiations need to be personalized against customer profiles. This enables the customer-facing provider to assign negotiation time based on customer priority or refuse to (re)negotiate, as determined by centrally defined policies on billing history, renegotiation attempts, obligations, or infrastructure readiness. Thus, commercial or non-commercial customers are treated in a standard manner. When policies change, subsequent (re)negotiations reflect these with agility.

Cloud Value Chains

This use case highlights a multi-domain SLA scenario spanning three deployment models of cloud. In Chapter 7, requirements for SLA-aware infrastructure management are considered with focus on the PaaS model of this use case.

This use case reflects the popular chaining trend among SaaS, PaaS and IaaS providers as highlighted in Fig. 4.3. IaaS clouds generally compose of a large number of hosts in distributed data centers. Similarly, PaaS clouds are built using virtual resources e.g., VMs from multiple IaaS providers spread across the globe. Cross-zone distribution allows to comply with regional regulations, risk mitigation incase of disasters and better quality of experience for customers.

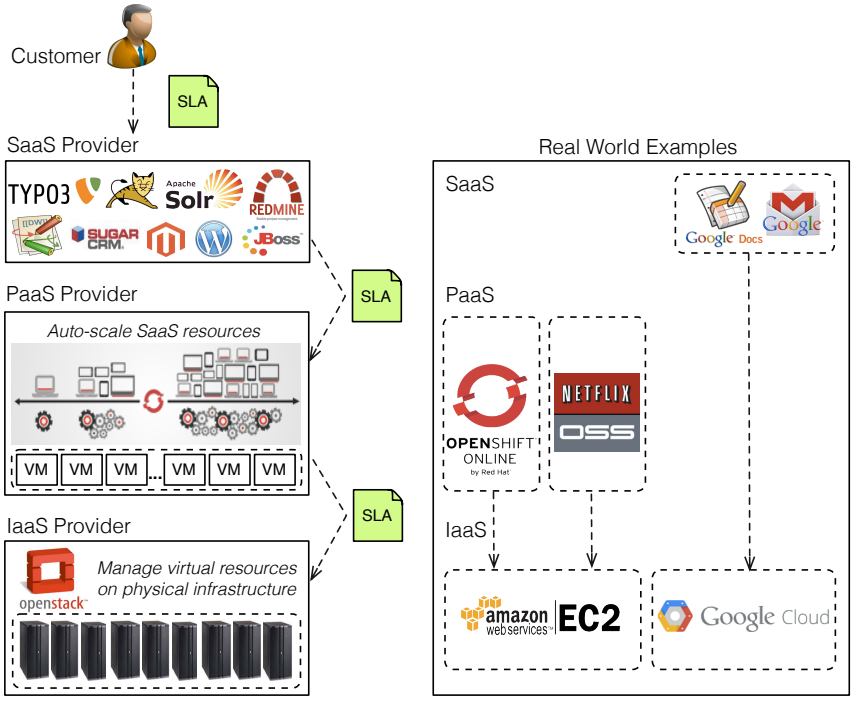


Figure 4.3.: SLA Hierarchy in Cloud Value Chain Scenario

PaaS integrates quality concerns of SaaS layer with the IaaS layer while hiding administration complexity. This not only eases development and delivery of SaaS, but the PaaS cloud itself can scale dynamically i.e. VMs from different IaaS providers can be added (or released) in the PaaS federation. In this way, PaaS efficiently utilizes IaaS resources using multi-tenant containers to securely host SaaS instances on shared VMs, unless explicitly disallowed by a SaaS customer.

Further, PaaS stacks allow to cloud enable off the shelf software and create add-on based composite services with low operational expense, little capital and short time to market. This enables true exploitation of cloud computing [21].

The real world relevance of this use case is also depicted in Fig. 4.3 with three examples. There are over 2.5 million applications provisioned on OpenShift Online [94] (public PaaS by RedHat), which is itself deployed on Amazon EC2 [20]. Netflix serves 50 million subscribers through its PaaS, also deployed on EC2. Google launches over 2 billion containers across its global data centers every week to power its SaaS offerings like gmail and docs [107].

Cloud value chaining is ideal for SLA based end to end procurement. The end (SaaS) customer is concerned with QoS. As depicted in Fig. 4.3, the customer-facing SaaS provider has SLA dependencies (for quality and quantity) on PaaS resource containers. The PaaS provider has SLA dependencies on IaaS resources on which the PaaS Cloud is deployed. At each level, negotiations are conducted to resolve SLA dependencies and to avoid over and under procurement of resources.

4.2. Problem Description

This section builds upon the SLA scenarios in stated use cases and presents a refined description of the three research problems or questions identified for this thesis. These problems are an embodiment of the two SLA management perspectives identified in Section 3.2 i.e., they help to create value and manage infrastructure concerns.

4.2.1. Development and Execution of Negotiation Protocols

Unlike most of the prior art on negotiation protocols, this work views negotiation protocols as mechanisms to sell customizable service propositions. To maintain competitive advantage, selling mechanisms therefore need to be diversified. Owing further to the fact that no single protocol can satisfy all selling/negotiation scenarios [35,36], the primary aim is to realize a generic methodology to develop negotiation protocols. Nevertheless, operating multiple protocols is another open challenge [37,38]. This serves as the secondary aim of this problem. Upon closer inspection, the problem is decomposed into sub-problems as described below.

First, a generic method should lead from specifying a protocol design, formally verifying its functional and non-functional properties, to implementing it

as a machine executable artifact that can be shared. Such a holistic approach would eliminate critical issues in protocol adoption that arise from dissimilar interpretation of the same specification, or different implementations.

Second, business policies often intermingle with protocol design and limit reusability [26, 40]. Thus, business and engineering concerns need to be separated. The implementation format of the protocol should be flexible to allow domain-specific hooks. This is needed to configure protocol parameters according to the business domain, without intermingling with its interaction semantics.

Third, consistent execution of a protocol at each negotiating party is hard to ensure [38], therefore, a common negotiation platform is needed for point-to-point execution of shared protocols. If protocols are encoded in a standard scheme, the platform could leverage this uniformity to process newly authored protocols with little or no modification of processing functionality.

To establish feasibility of the proposed method, a concretely developed protocol should serve as adequate proof. This needs to be executed in chained negotiations (as seen in the use cases) using the negotiation platform. Thus, the platform provides the foundation to roll out service offerings negotiable using a specific protocol, while also facilitating inter-operability with other provider protocols.

This multi-faceted problem is very challenging due to its broad scope. However, it enables high value creation possibilities. To solve this problem requires amalgamation of key concepts from prior art in a coherent fashion. These have been previously partially or solely attempted, with split or case based emphasis.

4.2.2. Utility-optimizing Negotiation Strategies

This problem regards the ever increasing SLA gap between the customer and the provider. The desired feature-rich SLA templates comprise of multiple quality of service (QoS) properties, but preferences for these vary from customer to customer. Therefore, cloud service providers currently offer restricted QoS guarantees in the form of take-it-or-leave-it SLAs to limit their liabilities [24]. Automated agents using a negotiation strategy can reduce this SLA gap by tactically compromising on SLA utility - a process which discovers win-win offers in subsequent negotiation rounds. Nevertheless, to converge on an SLA, agents can exchange millions of negotiation rounds in few minutes, which is computationally expensive and prone to network instability. Thus, negotiation decision making is a difficult problem because time window is short, multi-issue contract spaces are

usually large, objectives are not shared and preferences often conflict [62, 82].

Under this context, this problem constitutes of two aspects. First, utility-optimizing negotiation strategies are needed to create feature-rich SLAs in domains ranging from small to very large contract spaces, while adapting to the conceding behavior of opponent and time. Second, considering that several strategies have already been proposed in prior art, a holistic evaluation of state of the art strategies is necessary to understand negotiation dynamics. This should include learning-based, non learning and mixed strategies to simulate diverse market compositions. Evaluations should also consider varying degrees of preference conflicts to resemble the real world differences between customer and provider roles. Further, as in the real world, individual objectives should be kept confidential and opponent strategies should be previously unknown to each other.

Mindful that no single strategy can outperform others in all negotiation domains, the aim is to determine robustness of developed strategies and approximate payoffs. The latter should consider multiple notions of utility e.g., utility, social welfare and Pareto-optimality of SLAs to derive meaningful conclusions.

4.2.3. SLA-aware Cloud Resource Management

In general, cloud schedulers over-provision resources to improve machine utilization because not all services are simultaneously active. Monitoring tools collect resource usage data, which allows to estimate build up of potential SLA violations. Based on this information, resources can be proactively reallocated to preempt violations. Before services with advanced SLA guarantees can be offered in production, it is vital to investigate SLA-aware resource management in simulated cloud environments. Prior art on resource reallocation has focused on machine level concerns, without paying due attention to service related concerns such as SLA dependencies among services. However, both concerns stem together in the "service consolidation" problem, which recurs in PaaS clouds.

The problem exacerbates because on-demand procurement, unpredictable workloads and auto-scaling result in rapid provisioning and release of cloud resource containers. The latter host a service's deployment units e.g., database, load balancer, application server, etc. This causes undesired utilization of machine resources and energy wastage that can be avoided with real time planning.

Hence, the main aim of this problem is to plan and optimize the placement of containers on cloud machines, such that SLA commitments are fulfilled and

resource utilization is maximized using minimum machines. However, multi-resource based allocations and machine heterogeneity qualify this as a multi-dimensional variable sized bin packing problem. In addition, service-driven constraints regarding containers and machine related constraints regarding capacity, topology, location and migrations make SLA-aware resource management non-trivial. The problem needs to be assessed for clouds of varying scales, configurations and workloads as the variability effect helps resemble real world dynamics.

The problem thus requires a solution that would bring the cloud from an unconsolidated to a consolidated state by respecting all hard constraints and maximally satisfying soft constraints in short times. The performance of consolidation algorithms must be evaluated against formal quality metrics. These include objective function score which reflects consolidation quality, machines used, utilization, resource contention, SLA violations, migrations and energy use of proposed consolidated (reallocation) solutions.

The multiplicity of quality metrics and consolidated solutions by different algorithms pose a decision problem as to what solution to prefer. This requires alignment of business strategy with IT. Thus, a policy based decision needs to be formulated to automate solution selection according to provider's business objectives. Finally, in order to establish a good relevance between theory and practice, the problem should ideally be framed considering a real cloud stack.

Part II.

SLA Negotiation, SLA Planning and Optimization

Chapter 5.

A Generic Approach to Develop and Execute Negotiation Protocols

This chapter presents a generic approach as a solution to the first problem described in Chapter 4, i.e., development and execution of negotiation protocols. The contents of this chapter make use of author's publications [45, 46].

As emphasized in part I, there is market interest in QoS based added-value cloud services. This raises the importance of Service Level Agreement (SLA) negotiations, which allow to tailor service dependencies and procure appropriate resources throughout the value chain.

Currently, providers offer restrictive take-it-or-leave-it SLAs to limit their liabilities. The European cloud computing strategy [98] has criticized take-it-or-leave-it SLAs as undesirable for users and highlighted terms like data integrity, confidentiality, ownership and service continuity to be part of SLA. Similar demands are made by the cloud computing strategy of the US government [15].

In this work, negotiations are treated as flexible business models to exploit the dynamic nature of clouds for selling added-value customizable services. A negotiation protocol thus acts as an automated mechanism to sell a specific value proposition. The assumption is that such flexible mechanisms can dialectically address SLA concerns from both sides of the dependence equation, because i) these facilitate coordination among depending and dependent parties, and ii) can be accompanied with intelligent negotiation strategies to amicably resolve conflicting interests. European Commission's 2013 exploitation report on cloud computing SLAs also recommends providers to "support runtime adaptability through dynamic automatic SLA (re)negotiation mechanisms" [73]. In this context, SLA negotiations compliment on-demand procurement of cloud services.

Negotiated SLAs provide additional benefits. For instance, adapting to de-

mand fluctuations or change in business policy can be enforced by renegotiating existing SLAs. As business models, negotiation mechanisms also serve to differentiate providers. For instance, Amazon created a unique cloud market for fault-tolerant applications by granting spare resources (called spot instances) through a bidding mechanism [20]. Similarly, Ebay is an auction based marketplace for trading goods. Currently, these mechanisms lack the notion of SLA as negotiations are limited to price. This provides motivation to apply multi-round SLA negotiations as a viable niche to sell cloud or cloud-based services.

Protocol governed electronic markets target specific segments e.g., spot instances do not replace on-demand or reserved instances market. Furthermore, it is a fact that no single protocol satisfies all negotiation scenarios [35]. In [36], it is argued that instead of a single protocol or central marketplace, a system of marketplaces using different protocols will eventually emerge.

However, developing and operating multiple protocols is an open challenge [37, 38]. Prior works have largely limited themselves to using a single protocol [40, 41, 53, 67, 68, 76]. This rigidity hinders diversification and is exacerbated by three issues. First, protocol development research has been highly subjective, with split or case based emphasis on specifying, verifying and implementation phases. Secondly, business configurations often intermingle with protocol design and limit its reuse [26, 40]. Thirdly, same implementation of protocol and its consistent execution across all parties is hard to ensure [38].

Along these lines, this chapter presents a general purpose methodology to spawn and execute negotiation protocols. Based on a thorough review of prior art and requirements from the SLA@SOI use cases, our approach attempts to coherently amalgamate key concepts from design, verification and implementation phases of protocol development. The following aspects are specifically addressed:

- Designing protocol specifications which can be verified for functional and non-functional properties.
- Flexible implementation that allows domain-specific hooks for parametric configuration of protocol in order to personalize negotiation sessions.
- Seamless execution of protocols across parties.

To address these concerns, following concrete contributions are made, which bridges several gaps in the current art:

1. A generic approach leading from design to implementation of negotiation protocols as verifiable and machine executable artifacts. This is illustrated by presenting the *Simple Bilateral Negotiation Protocol* (SBNP) - a multi-round and configurable protocol deployed in SLA@SOI use cases and exemplified here for the cloud value chain usecase (see Section 4.1).

2. A common negotiation platform for *same* execution of shared protocol among distributed parties.

3. Verification based evaluation proving the efficiency of proposed protocol, if value chains are orchestrated on a centralized marketplace.

5.1. Related Work

Negotiations have been successfully applied to problems such as service composition, scheduling resources in computational grids and procuring resources in logistic service chains [44, 53, 76, 85, 99, 100]. Application of negotiation has surged in recent cloud based projects [73]. However, diversifying market mechanisms through protocols, execution machinery and decision models in order to maximize business utility of SLAs, while facing conflicting interests and unknown opponents remain open problems. As lifecycle management of cloud services is being reshaped around SLAs, the role of negotiations becomes more important [12]. In this section, prior art relevant to presented work is highlighted.

Iterative Contract Net Protocol (ICNP) [59] allows iterative submission of proposals. The process is initiated when a customer invites a provider to submit a proposal against a call for proposal (cfp) message. Provider returns a binding proposal and the customer can accept/reject proposal or request new proposal. This sequence is iterated till agreement occurs or negotiation time ends.

The Web Services Agreement (WSAG) [56] is a specification which provides an XML based SLA standard to define SLA templates. It further provides a basic take-it-or-leave-it protocol to create an SLA. Its extension WSAGN [57] provides a multi-round negotiation protocol with additional (but fixed) interactions.

Unlike ICNP, the WSAGN is well suited to cloud service providers as non-binding offer(s) and counter offer(s) can be exchanged between parties in request-response style. This is conducive to resolve conflicts without the necessity to book resources or make reservations in each round, which can be counter productive

for small providers [100]. The customer submits a binding offer in a separate (final) message, which is converted to SLA if the provider accepts.

Although SLA standard is a welcome contribution to solve the interoperability problem, providers cannot be bound to a single negotiation protocol. The fact remains that no single protocol can satisfy all negotiation scenarios [35,37]. Further, we view negotiation protocols as selling models to differentiate providers. Thus what is needed is a generic methodology that allows to develop differing protocols for value creation.

On a more technical level, both ICNP and WSAGN protocol specifications rely on informal methods [57, 59] like sequence diagrams to model interactions. This raises concern whether applying them in specific negotiation scenarios, especially with hierarchical negotiations would be safe or efficient. For instance, the WSAGN allows signaling (interactions) where both parties may simultaneously send messages, which may cause deadlock. A formal view of interacting processes where all possible interaction states can be examined is missing.

Therefore, we model negotiation protocols using communicating finite state machines (CFSM), which precisely describe states of customer and provider processes and their messaging interaction. The CFSM model can be represented in Process Meta Language (PROMELA), which provides built-in primitives to describe a process's state machine, interaction sequence and hierarchical formations. The model checker Spin can process PROMELA model to formally verify non functional aspects and functional compliance of a protocol.

Unlike WSAGN, we do not model states like solicited, advisory, accepted or rejected on exchanged offers. This is because feature-rich SLA templates can contain very large contracting spaces [62]. For instance, our experiments used templates containing upto 10^{30} possible offers. Maintaining states per offer is computationally infeasible in multi-round protocols, where hundreds, thousands or even millions of offers are exchanged to facilitate convergence. Here again, verification is needed to determine state space growth of a protocol.

Among other works, Yeung has used Hoare's CSP to model a contract net protocol for service chains in manufacturing control [41]. Kraenke et al. [53] used a variant of ICNP for logistic service chains and verified it using Spin. In [38], specification and consistent interpretation of protocols is addressed in open systems. The authors confirm that statecharts provide intuitive graphical modeling than Petri-nets or AUML diagrams. They propose verification using

propositional dynamic logic (PDL) language. However, unlike them, our scope is not limited to mere correct specification, rather also to implement the protocol and provide an execution machinery.

For implementation, we argue that an implementation format must be configurable so a protocol can be conveniently adapted to wider business contexts. At the same time, it must be machine executable and expressive. This is a difficult balance to meet. The WSAG4J framework [69] implements the WSAGN specification and its negotiation protocol in Java code. Such imperative implementations are not easy to modify or extend. [54] argues for an XML based implementation of protocols. Though XML provides the benefits of an explicit and expressive format, protocol-generic execution is not possible without extending the execution platform to parse and process each new protocol. This highlights the difference between machine readable and machine executable formats.

Our concept of a protocol is a self-contained, configurable and machine executable artifact that can be publicly shared to avoid dissimilar interpretation and implementation of the same protocol specification. Hence, our solution uses declarative rules to implement protocols. Rules amenably map finite state machines for customer and provider processes as executable (precedence-consequence) logic blocks. This separation prevents cluttering of protocol code with that of execution platform, eases configuration and promotes reusability.

Haifei has used rules [40] to map high level policy to low level decision actions during negotiations e.g., to process an offer or generate a counter offer. In contrast, we use rules only to implement the protocol and consider decision making actions as part of negotiation strategy, which is loosely related with the protocol.

Project SeCSe [75] has addressed protocol development using rules. However, SeCSe specifically targets marketplace-assisted negotiations. Thus, a three party protocol is used, where users must provide their objectives to a centralized marketplace, which mediates by proposing mutually satisfying offers to buyer and seller agents. Among other negotiation support systems, [68] presents a comprehensive broker architecture, which performs matchmaking and SLA negotiation for web services on behalf of requested and offered QoS levels. In this way, the broker selects the best service provider. [76] has developed a broker framework, which uses preferences submitted by customer and provider to launch bilateral negotiations using agents within its platform. These frameworks use Heuristic [68, 76] or Metaheuristic search [75] to create mutually satisfying SLAs. Un-

like these works, our negotiation platform does not require participants to submit business objectives to a third party since the SLA framework integrating it allows negotiations directly among distributed parties. However, our design is flexible and can be reused to implement central marketplace or brokerage system among trusted parties. In fact, our platform was adopted by the EU project Contrail to implement brokered SLA negotiations in federated cloud scenarios [33].

5.2. SLA Management Framework

The SLA@SOI project [93] developed a Generic SLA Management (GSLAM) framework [63]. GSLAM automates SLA management on top of IT stacks. A simplified architecture of the GSLAM framework is shown in Fig. 5.1. The components in dashed boundary are generic components, while others need to be adapted to the provider domain by implementing the provided interfaces.

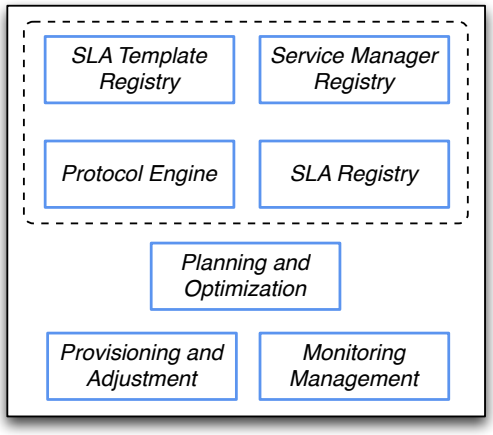


Figure 5.1.: Generic SLA Management Framework

The negotiation platform called the Protocol Engine is integral part of GSLAM implementation [71]. Its design and functionality is explained in Section 5.4. The SLA Registry component provides a store for established SLAs. The Service Manager Registry stores information regarding the provider's service landscape, licenses, dependencies and resource requirements for different service levels. The Planning and Optimization component acts as the executive controller that implements decision models e.g., for negotiation or resource consolidation/management. The Provisioning and Adjustment component is responsible for provision-

ing a service on the underlying cloud infrastructure. Monitoring Management provides low level monitoring of deployed resources to enact SLAs throughout the lifecycle.

The framework enables distributed parties to come together in order to effectively negotiate, collaborate and manage a service agreement platform. For this, various modules need to discover and interact with each other through their corresponding SLA Managers. The SLA Template Registry provides a publish/-subscribe based advertisement system, which uses a broker component to discover and disperse a provider's SLA templates to interested parties. This creates an open cluster of distributed SLA Managers as shown in Fig. 5.2.

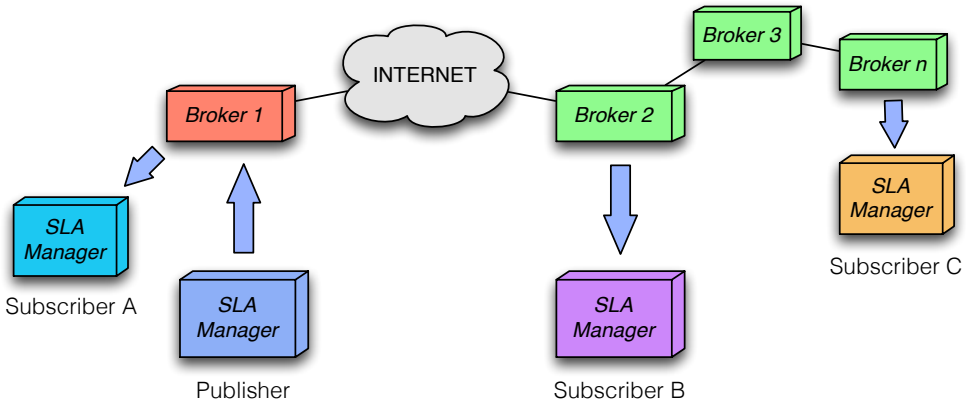


Figure 5.2.: SLA Template Discovery by SLAM Advertisement System

5.3. Development of Negotiation Protocols

Based on our usecases and thorough review of relevant literature, we state following requirements for protocol design.

R1 Negotiation scenario and interaction: Service propositions may contain SLA dependencies e.g., software or infrastructure provisioning on other providers. This constitutes a negotiation scenario and an interaction is designed to engage stakeholders.

R2 Description: Visual description is easy to comprehend but a formal description removes ambiguity.

R3 Non-functional and functional aspects: A protocol must be verified for non-functional properties such as state space growth and inconsistent states. Similarly, correctness criteria must be verified.

R4 Implementation: A verified protocol implements the interaction in a machine executable form.

To address these requirements in a generic fashion, we propose a systematic protocol development lifecycle, comprising four distinct phases namely modeling, verification, implementation and generic execution. We illustrate these phases by introducing the SBNP protocol for the negotiation scenario.

Negotiation Scenario: Negotiations depend on the characteristics of a scenario [35, 68]. Our scenario is inspired from the cloud value chain use case (see Section 4.1). This popular chaining trend among SaaS, PaaS and IaaS providers eases procurement of SaaS for the customer. On other other hand, the PaaS provider can dynamically scale PaaS cloud e.g., virtual resources from different IaaS providers can be added (or released) in the PaaS federation [47, 48].

Cloud value chains provide a realistic negotiation scenario for end to end SLA based procurement. The end (SaaS) customer is concerned with QoS. As depicted in Fig. 5.3, a customer-facing SaaS provider has SLA dependencies (for quality and quantity) of PaaS resource containers. The PaaS provider has SLA dependencies on IaaS resources on which the PaaS cloud is deployed. At each level, negotiations are conducted to resolve SLA dependencies and to avoid over and under procurement of resources.

5.3.1. Modeling

Protocol specifications for ICNP and WSAG/N have used sequence diagrams for modeling interaction behaviors [57, 59]. The problem with sequence diagrams is that they capture partial behaviors only. Visualizing an interaction is necessary but not sufficient. A modeling approach that can be formally expressed allows i) to eliminate ambiguity, and ii) is amenable to automated verification to rule out inconsistencies and incorrect behaviors.

We therefore use Communicating Finite State Machines (CFSM) [61], which cater for these requirements at varying degrees of abstraction. We are particularly interested in bilateral protocols that are composed of "sender" and "receiver" process roles. We now illustrate how interaction between these processes

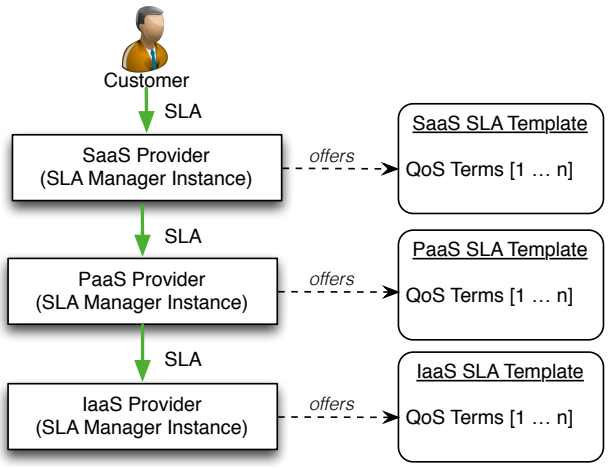


Figure 5.3.: Negotiation scenario and SLA dependencies

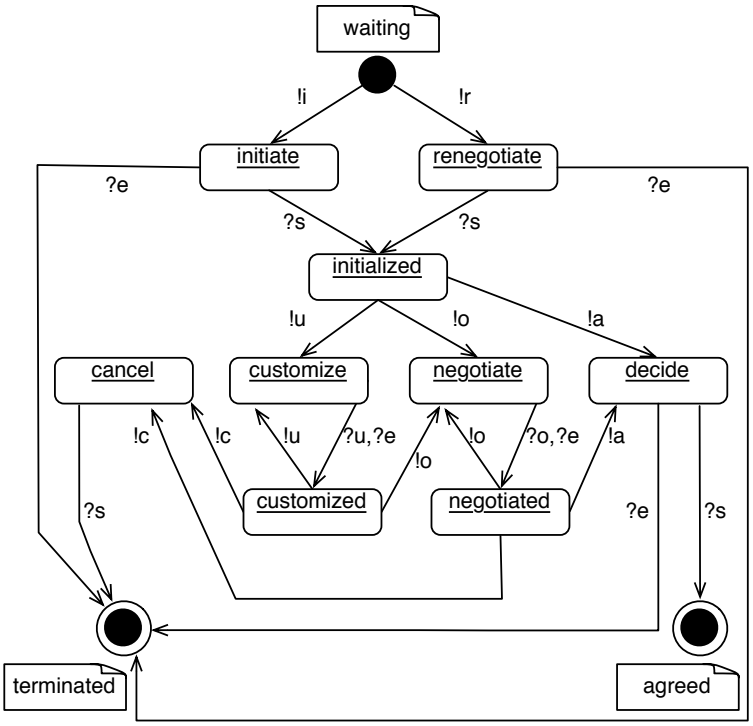
is modelled using CFSM.

The first step in modeling is to identify possible states. We propose a finite set S={*waiting, initiate, renegotiate, initialized, customize, customized, negotiate, negotiated, decide, cancel, terminated, agreed*} of states, which can be uniquely arranged to define an interaction. Figs. 5.4(a)&(b) show the CFSM for the sender and receiver side of SBNP. States are distinguished as *request* and *response* states and messages trigger a transition from the former to latter.

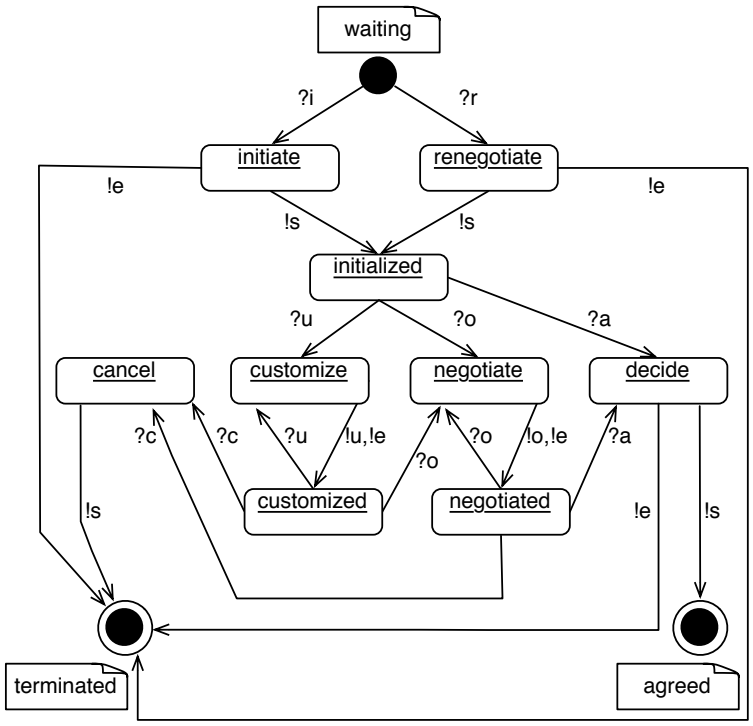
The second step is to identify the messages that can be invoked on the states. To this, we propose a list of messages which are described along with their alphabetic abbreviation in Table 5.1. Messages have a crucial link with the negotiation interface that results (discussed in Section 5.4). The argument is that proposed states and messages can be reused or extended to design additional protocols.

The third step deals with the parametric configuration of protocols. We identified six parameters as shown in Table 5.2. In this way, different parameter values may be set for individual negotiations (see Section 5.3.3).

Now, we present a non-exhaustive CFSM formalism for SBNP - a bilateral (client-server like) protocol. Given the set of states S and a non-empty finite set of negotiating agents $A = \{a_0, a_1, ...a_n\}$, we define SBNP as a network of CFSM(s) $\mathcal{A}$ such that one machine in this network represents one negotiating agent. In this context, the term agent and machine can be used interchangeably. Then, for all $a \in A$
$\mathcal{A} = (\mathcal{A}_a, I, F)$



(a) Sender



(b) Receiver

Figure 5.4.: Simple Bilateral Negotiation Protocol (SBNP)

Table 5.1.: Protocol States, Messages and Alphabets

State	Message	Alphabet	Description
waiting	-	-	Wait for negotiation commencement request
initiate	initiateNegotiation	i	Request to initiate negotiation
renegotiate	renegotiateAgreement	r	Request to renegotiate an existing SLA
initialized	-	-	Response state for *initiate* and *renegotiate*
customize	customizeParameters	u	Request to modify protocol's default parameters
customized	-	-	Response state for *customize*
negotiate	negotiate	o	Request that provides an SLA offer
negotiated	-	-	Response state for *negotiate*
cancel	cancelNegotiation	c	Request to gracefully cancel negotiation session
decide	createAgreement	a	Request to create an SLA of proposed (final) offer
terminated	-	-	Response state that ends negotiation unsuccessfully
agreed	-	-	Response state that ends negotiation successfully
-	successful response	s	A positive response message
-	unsuccessful response	e	A negative response message

Table 5.2.: Protocol Parameters

Parameter	Description
Process_Timeout	Life time of negotiation process
Customization_Rounds	Rounds for fixing protocol parameters
Negotiation_Rounds	Rounds for exchanging offers
Max_Counter_Offers	Offers sent as response to received offer
Optional_Critique_On_QoS	Critique on term value e.g., *increase*, *decrease*, *change* or *acceptable*
Quiescence_Time	Inactivity time among negotiating agents
Chain_Length	Allowed length of negotiation service chain

$\mathcal{A}_a = (S_a, \rightarrow)$ is a finite state machine
S_a is a set of local states
$\rightarrow \subseteq S_a \times Act_a \times S_a$ is a set of local transitions
$Act_a \subseteq Act$ is a set of local actions
I is a non-empty set of global initial states, and
F is a non-empty set of global final states

The set $Act = \{send, receive\}$ is a set of communicative actions in $\mathcal{A}$ and used by the local transitions that take $\mathcal{A}_a$ from one state to another. Agents communicate solely by passing messages over first-in-first-out (FIFO) channels. Agent a_i and a_{i+1} communicate over a channel $c = (a_i, a_{i+1})$ and $c' = (a_{i+1}, a_i)$ where $1 \leq i < n$. The action *send* is defined for agent a to pass message m to agent a_{i+1} as $Act^{!}_{a} = (c!m)$. Its corresponding action is *receive* defined at agent a_{i+1} to receive the passed message as $Act^{?}_{a_{i+1}} = (c?m)$. Similarly, *send* is defined for agent a_{i+1} as $Act^{!}_{a_{i+1}} = (c'!m)$ and its corresponding *receive* for agent a is defined as $Act^{?}_{a} = (c'?m)$. Then, $Ch = \{c, c'\}$ is a set of outgoing and incoming channel per agent-pair. The message $m \in \sum$ where $\sum = \{i, r, u, o, c, a, s, e\}$ is a

set of input alphabets used by *send* and *receive* actions.

Interactions and Marketing Enablements

Figs. 5.4(a)&(b) show how SBNP interactions are organized. Generally, the receiver process (provider) resides in the *waiting* state until a sender process (customer or provider) invokes the initiateNegotiation or renegotiateAgreement message, resulting in the *initialized* state which establishes a negotiation session. The negotiate message allows to exchange (non binding) offers in multiple rounds.

By default, a provider's parameter values are enforced by the negotiation platform during negotiation. SBNP additionally provides an optional interaction through the customizeParameters message to adjust parameter values between the parties prior to offer exchange. Negotiation can be cancelled using cancelNegotiation message. The createAgreement message is used to submit a final (binding) SLA offer. If accepted by receiver, an SLA is created and also returned.

Fig. 5.5 shows how SBNP enables a negotiation (value) chain in our stated scenario. The customer-facing SaaS negotiation triggers a nested negotiation carried out by the SaaS provider with the PaaS provider, which in turn launches a nested negotiation with the IaaS provider. In so, the SaaS and PaaS providers play the dual role of a receiver and a sender in the same negotiation session.

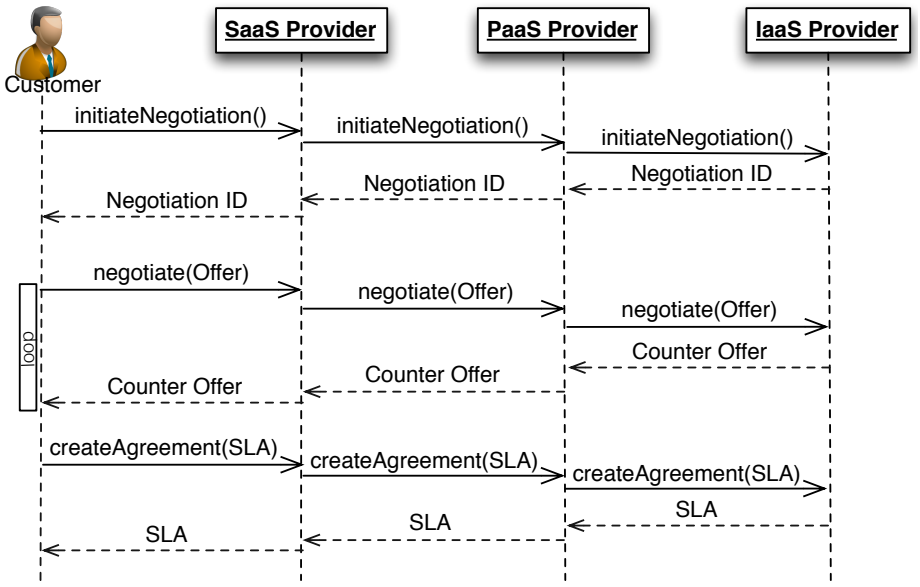


Figure 5.5.: SBNP enabled Negotiation Chain

A single protocol may contain multiple interactions e.g., SBNP allows single offer, multi-round and customized multi-round interactions. Interaction design reflects on service proposition. SBNP is ideal for single stage propositions. However, protocols enabling multi-stage propositions can be designed similarly. Consider an IaaS-SaaS proposition. At first stage, IaaS resource such as storage is negotiated, followed by an optional second stage to negotiate QoS for file synchronization and management service e.g., ownCloud (SaaS) to be provisioned over procured storage. Thus, based on the optional or mandatory nature of each negotiation stage, unique service propositions can be designed. In [70], SLA aggregation patterns have been proposed to create service propositions. These can be ingested to design further negotiation protocols, so aggregation points can be resolved by engaging both the customer and provider(s).

5.3.2. Verification

As pointed out already, protocols must be verified before implementation. In [38], propositional dynamic logic is used whereas [53,64] have used Spin model checker [50]. As input, Spin requires a protocol model to be encoded in PROMELA abstract syntax. PROMELA provides built-in primitives to encode processes, states, channels and message passing between arbitrary many processes. Spin creates the protocol graph from PROMELA representation, and exhaustively searches all its paths to determine conformance or violation of correctness behaviors expressed in linear temporal logic (LTL) properties. These features provide strong basis to use Spin for verifying not just SBNP but any CFSM protocol model. To contain a complex topic in limited space, we restrict ourselves to highlighting three contributions which lack attention in prior art.

1. Error detection in design or intended runtime use of protocol.
2. Evaluating state space growth.
3. Formulating correctness properties in LTL.

Error Detection

Spin verification run of SBNP confirms that inconsistencies like invalid end states, acceptance cycles and unspecified receptions do not exist. Cycles for customization and negotiation are marked as *progress* cycles, which means a bound will be

in place in real use and therefore Spin does not raise cyclic errors. By definition, no deadlock or livelocks exist which make SBNP a safe protocol.

However, some control actions in negotiation systems lie at controller component, which drives the negotiation platform. A common practice at this level is to retransmit a message if a sent message is not timely responded to, provided the number of negotiation rounds or negotiation time is not exceeded. Such control situations can be emulated in PROMELA/Spin based verification through non-determinism. This very expressive feature allows to detect errors in intended runtime use of protocol. Using this non-determinism, a transition (named as timeout) was introduced in the SBNP verification program. If executed, this transition resends the message as a timeout would prompt in the real system. Fig. 5.6 shows such a timeout transition applied at the *negotiate* state of sender process where a response is awaited.

When Spin verified the updated protocol model, it instantly detected deadlocks. Spin also pointed out the exact error. The new transition can cause a cyclic retransmission of message, which builds a queue at the receiver. The sender may meanwhile transit to the decide state, where a single (late arriving) error response of an earlier offer may further transit it to the terminated state while the receiver is still active. Moreover, the late arriving agreement request could be accepted by the receiver creating a one-sided SLA while the sender has already aborted. Such inconsistencies (invalid end states) must not happen. Hence, in this work, a safer choice was adopted by using "Process_Timeout" with

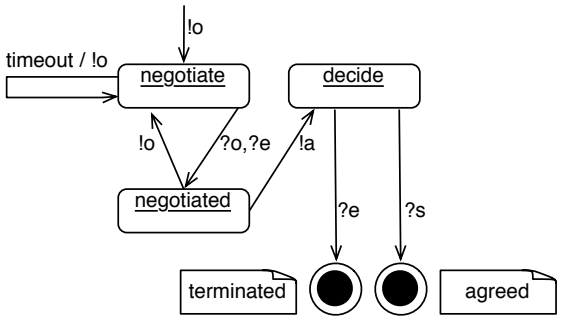


Figure 5.6.: Timeout at Sender

optional request level quiescence at the negotiation states. If a request or a response is lost, the machine remains quiescent i.e., blocked in its current state until a "Quiescence_Time" parameter times out, after which, each side voluntar-

ily relinquishes the session and maintains system consistency.

Evaluating Protocol Scalability

Although the SLA@SOI framework is aimed at distributed negotiating parties, the proposed negotiation platform (Section 5.4) which is integral part of the framework may also be used independently to create centralized marketplaces e.g., similar to Ebay or Amazon. However, unlike these platforms, the presented work focuses on chained negotiations where a customer facing provider agent is expected to instigate nested negotiations with other provider agents registered with the marketplace. In this case, estimating how a protocol scales is important to address memory requirements. Recall that a CFSM network is made of sender or receiver FSMs (representing customer and provider agents) and channels between them. In this context, SBNP scalability refers to increase in states of the CFSM network as more agents partake in negotiation chain (see Fig. 5.5). The worst case state space growth [65] is obtained by:

$$\prod_{i=1}^{N}(|\#\text{FSM states}|\prod_{y}^{y\in V}|dom(y)|)\prod_{j=1}^{K}|dom(c_j)|^{cap^{c_j}} \tag{5.1}$$

The variables in this equation are set in accordance with our experiment using SBNP's PROMELA program [66]. This is used by Spin to simulate chained negotiation scenario. The program implements the protocol automatons in PROMELA abstract syntax and may use a variable set V for this purpose. Since these variables are for verification use only, they need not to be represented in the graphical (CFSM) model. N represents the number of agents in a chain. "FSM states" refers to the states in SBNP's sender and receiver automatons i.e., 12. We used a single variable $y \in V$ to send or receive a message.

For the agent at the start of negotiation chain, the $dom(y) = 2$ since it only concerns with storing successful or unsuccessful response. For agents between the start and end of chain, the $dom(y) = 8$ to store the six messages possible in SBNP and two responses. Finally, the $dom(y) = 6$ for the agent at rear end of chain as it only needs to store messages. $dom(c_j)$ is the domain of each of the K channels. There are two channels between each agent-pair and $dom(c) = 6$ for outgoing channels carrying the 6 possible messages and $dom(c) = 2$ for the incoming channels carrying the 2 possible responses. cap^{c_j} refers to the amount of messages that may be sent at one time e.g., in one negotiation round.

Spin distills state space by using partial order reduction to generate only reach-

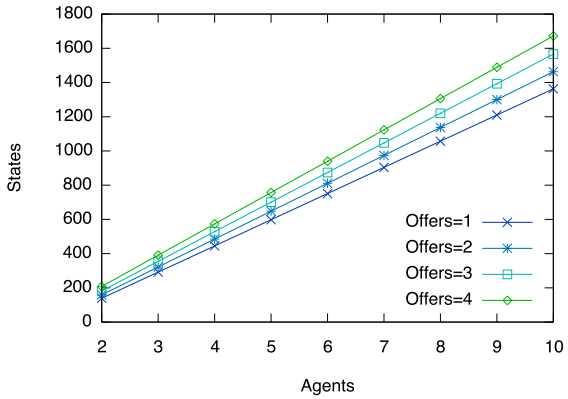


Figure 5.7.: Growth in States

able protocol states. In our experiment, Spin generated from 139 to 1363 states for $N = \{2, ..., 10\}$ agents exchanging a single offer. These increase from 159 to 1463 states, 182 to 1566 states and 208 to 1672 states when exchanging 2, 3 and 4 offers respectively. This is a manageable growth and the graph in Fig. 5.7 reveals that increase is linear. This establishes SBNP as an efficient protocol. Further, it shows that protocol parameters like "Chain_Length" and "Max_Counter_Offers" be fixed to minimum values. As negotiation strategies add to the computational load, linearly or sub-linearly scaling protocols are preferred in negotiation chains.

Functional Correctness

Functional requirements are derived from negotiation scenario(s). Formalizing requirements is non trivial [51]. Hence we elaborate this by example which provides a practical way to also verify other protocols. For our scenario (Figs. 5.3, 5.5), there are two functional requirements for SBNP:

1. SLA creation with customer requires that dependent SLAs throughout the negotiation chain are created to avoid over commitment. Conversely, if SLA is not created with customer, dependent SLAs should not be created to avoid over procurement.

2. Exactly one or no SLA must be created between all negotiating pairs in a negotiation chain.

LTL formalizes requirements as safety or liveness behaviors. Behaviors are temporally ordered sequence of states succinctly expressed as LTL properties.

To begin, occurrence of a certain state is captured as a flag in the verification program. A proposition is a boolean condition expressed using flag(s). Search determines if a proposition holds true. Proposition may be decorated as safety property (preceded by $\Box$ symbol, which tests its universality) or as liveness property (preceded by $\Diamond$ symbol, which tests its eventual permanence). Propositions can be combined together to examine occurrence of complex sequences of states using logic and modality operators such as conjunction ($\wedge$), disjunction($\vee$), negation ($\neg$), implication ($\rightarrow$) and until ($\cup$). This daunting task is eased by Dwyer's property specification pattern templates [51, 52] that can be reused to map well-understood conception of system behaviors as LTL properties.

The patterns fundamental to our requirements are those of Precedence and Response. Precedence specifies that occurrence of a certain state called the cause is a necessary pre-condition to the occurrence of another state called the effect. Response specifies that occurrence of a cause acts as a stimulus that must be followed by effect. Precedence Chain pattern applies cardinality to Precedence by relating one cause to more effects and vice versa. Similarly, Response Chain pattern applies cardinality to Response by relating one stimulus to more responses (effects) and vice versa. An important step to use these patterns is to identify the cause and effect states. Cause is usually the state where the first agreement creation takes place. In SBNP negotiation chains, the prerogative of first agreement creation lies with the provider at the end of chain, which serves as the cause.

In two party SBNP negotiations, agreement created at receiver is the cause state, referred by the flag l_agr while agreement created at the sender is the effect state referred by the flag s_agr. For SBNP chained negotiations, cause is determined if the IaaS provider creates agreement requested by the PaaS provider and marked by the flag l_agr. The flag m_agr marks the agreement created by PaaS provider upon request by SaaS provider. Finally, the flag s_agr marks the agreement creation at SaaS provider upon customer request. This leads to our LTL properties.

$\Diamond s\_agr \rightarrow (\neg s\_agr \cup (l\_agr \wedge \neg s\_agr))$: This Precedence property applies to two party negotiation. It states that an agreement reached eventually at the sender (s_agr) is preceded by an agreement at the receiver (l_agr). The proposition $\neg s\_agr$ remains true i.e. sender does not reach agreement until the cause i.e. the l_agr occurs, after which the sender also reaches agreement. Its scalable variant $\Diamond s\_agr \rightarrow (\neg s\_agr \cup ((l\_agr \wedge \neg s\_agr) \wedge \Diamond(\neg s\_agr \cup m\_agr)))$ extends the behavior

to our 3 tier negotiation chain, by implementing the Precedence Chain (2-cause, 1-effect) pattern. It states that agreements are reached at last (IaaS) and medium (PaaS) tiers before the start tier (SaaS) reaches agreement. Thus, SBNP rules out contracting dependent SLAs without ever making SLA with customer.

$\Box(l\_agr \rightarrow \Diamond s\_agr)$: This Response property applies to two party negotiation. It states that always if an agreement is reached at the receiver (l_agr), eventually an agreement at the sender (s_agr) will follow. Its scalable variant $\Box(l\_agr \rightarrow \Diamond((m\_agr \wedge \neg s\_agr) \wedge \Diamond s\_agr))$ extends this behavior to our 3 tier negotiating chain by implementing the Response Chain (1-stimulus, 2-response) pattern. The property reads that always an agreement at the last tier (IaaS) is eventually followed by a sequence of states where the middle tier (PaaS) reaches an agreement and the start tier (SaaS) has not reached agreement, but eventually a state follows where the SaaS also reaches agreement. Thus, SBNP ensures that customer-facing SLA is supported by all dependent SLAs in procurement chain.

$\Diamond\Box(one\_or\_no\_sla)$: This property states that eventually, it is always the case that negotiation resulted in one SLA or no SLA. A flag is incremented in the agreed state to mark successful SLA creation. The proposition *one_or_no_sla* checks it in all parties, ensuring that exactly one or no SLA is created between all negotiating pairs.

In LTL theory, Precedence does not tie each cause to an effect. It merely states that a cause precedes effect which lets causes to occur without subsequent effects. Response allows effects to occur without cause. To relate cause and effect, Response is used as converse to Precedence. Together, Precedence and Response properties compliment each other to determine a precise behavior.

5.3.3. Implementation

Protocol development culminates with implementation. The challenges here are to use a format that is descriptive, easy to encode and can be generically executed with little or no recompilation. There is dearth in existing literature on implementing negotiation protocols. The WSAG4J framework [69] provides a Java implementation for the WSAGN specification. Here, the protocol is implemented as imperative code which is not easy to modify or extend. [54] argues for an XML based implementation of protocols. While XML provides the benefits of an explicit and expressive format, protocol-generic execution cannot be achieved without extending the platform to parse and process each new protocol.

We argue that a protocol should exist as a self-contained, executable artifact that clearly defines finite state machine (FSM) interactions. This entails that a protocol can be publicly shared as a file, which avoids dissimilar implementations of the same protocol specification. This necessity is heavily argued for in [38]. Further, this leads to a protocol-generic foothold in multiple markets by virtue of executing multiple protocols in a common execution environment.

These technical requirements are sufficiently met by declarative rules. Rules amenably map FSM semantics such as transitions and guard conditions into executable logic easily encoded as "*IF-THEN*" clauses. We emphasize on the executable nature of rules because a common scheme to pass information to the rules and fetch the results back can delegate most of the processing complexity out of the platform and into the rules. This avoids cluttering the protocols in the platform code or creating XML documents which get difficult to read. It is this flexibility of rules that we leverage to achieve generic execution.

Specifically, we used the Drools rule syntax, which is highly expressive and allows to compose Java objects in a declarative manner [42]. Our rule encoding scheme binds an input event to a state in the *IF* part of rule. This acts as a pre-condition. If true, an action is performed in the *THEN* part. A feedback instruction (setProcessed) signals if the event processed by the current state produced a positive or negative response. The event is retracted from working memory at the end of rule. This way, rules match allowed events with states and implement transitions among state objects within the working memory.

We now present three simplified rule snippets to convey how SBNP is implemented using rules [66]. Fig. 5.8 shows a rule where *initialized* state transits to *negotiate* state upon receipt of an offer. Fig. 5.9 shows a rule that implements

Rule Initialized_To_Negotiate_Transition

```
1: IF
2: initializedState : State(name == INITIALIZED);
3: event : Event(name == OfferArrivedEvent);
4: THEN
5: initializedState.remove();
6: insert(new State(name=NEGOTIATE));
7: event.setProcessed(true);
8: retract(event);
```

Figure 5.8.: Transition Rule

a guard condition on the *negotiate* state when an offer is received. The rule is

processed if negotiation rounds have not exceeded allowed limit. It then increments the counter for negotiation rounds. The rule shown in Fig. 5.10 enforces negotiation lifetime. It uses the built in evaluation function to test if allocated time has exceeded. If so, it sends a negative feedback and a descriptive reason which the platform uses to end negotiation session. This rule is not bound to a particular event, rather it applies to all events (line 4). Since multiple rules may share an event and also qualify for execution simultaneously, a *salience* (priority) value can be specified on rules to fix execution order. Here, the highest salience (5 in our case) is used to preempt all other rules as negotiation time has ended.

Rule Guard_Negotiation_Rounds
1: *IF*
2: negotiateState : State(name == NEGOTIATE,
rounds ≤ maximumNegotiationRoundsAllowed);
3: event : Event(name == OfferArrivedEvent);
4: *THEN*
5: negotiateState.rounds++;
6: negotiateState.update();
7: event.setProcessed(true);
8: retract(event);

Figure 5.9.: Guard Condition Rule

Rule Control_Negotiation_Time
1: salience 5
2: *IF*
3: params: Parameters(eval(currentTime() ≥ (startTime + processTimeout)));
4: event : Event();
5: *THEN*
6: event.setProcessed(false);
7: event.setReason('Negotiation timed out');
8: retract(event);

Figure 5.10.: Negotiation Time Rule

In addition to implementing the protocol, domain specific business rules are written e.g., to compliment the customization states or set protocol parameter values when negotiation is initialized. This allows to personalize protocol parameters through business rules. One such rule can compute a rank using the negotiator profile object, which is loaded in rule engine at negotiation initialization. Based on her profile, provider's business policy rules can assign values to protocol parameters such as negotiation time or number of negotiation rounds. This allows to tradeoff negotiation complexity and chances of convergence in

negotiator-specific manner. Fig. 5.11 illustrates such a business rule. Similarly, whitelisting and blacklisting rules can be written to favor a customer or refuse to negotiate altogether [66].

Rule Customize_Protocol_Parameters

```
1:  IF
2:  initializedState : State(name == INITIALIZED);
3:  event : Event(name == CustomizationArrivedEvent);
4:  THEN
5:  initializedState.remove();
6:  insert(new State(name=CUSTOMIZE));
7:  if( partnerProfile.getRank == DESIRED)
8:     insert(new Parameter(NEGOTIATION_ROUNDS=100));
9:  else
10:    insert(new Parameter(NEGOTIATION_ROUNDS=10));
11: event.setProcessed(true);
12: retract(event);
```

Figure 5.11.: Business Rule

5.4. Negotiation Platform for Protocol Execution

The SLA Managers use SLA template to directly negotiate with each other using their individual negotiation platform i.e. the Protocol Engine component. The Protocol Engine receives messages over a negotiation interface exposed as a webservice end point reference (EPR). The negotiation interface is shown in Fig. 5.12 and consists of methods identified during modeling (see Table 5.1). Their functionality is briefly phrased below.

initiateNegotiation: Request to start a negotiation session. A session ID is generated and returned.

renegotiateAgreement: Request to renegotiate an existing SLA. A session ID is generated and returned.

customizeParameters: Submit a proposal suggesting parameter values to configure the used protocol.

negotiateOffer: Submit an SLA offer. The platform forwards the offer to the POC component. POC can create counter offer(s) which are returned.

createAgreement: Request to accept the submitted offer and create and SLA if acceptable. The platform forwards the offer to the POC component which makes the final decision.

cancelNegotiation: Request to cancel negotiation.

```
UUID initiateNegotiation ( SLATemplate offer )
UUID renegotiateAgreement ( SLATemplate offer )
Parameters customizeParameters ( UUID negotiationID, Parameters params )
SLATemplate[] negotiateOffer ( UUID negotiationID, SLATemplate offer )
SLA createAgreement( UUID negotiationID, SLATemplate offer )
cancelNegotiation( UUID negotiationID )
```

Figure 5.12.: Negotiation Interface

We made certain design choices regarding separation of concerns in order to provide maximum flexibility. These visibly distinguish the Protocol Engine (shortly PE) from other platforms [69, 75, 76]:

- Loose coupling of protocols with the platform.
- Separation of protocol from negotiation strategy.
- Separation of platform from specific SLA standard(s).

Fig. 5.13 shows the internal components of the PE. Upon the initiate call,

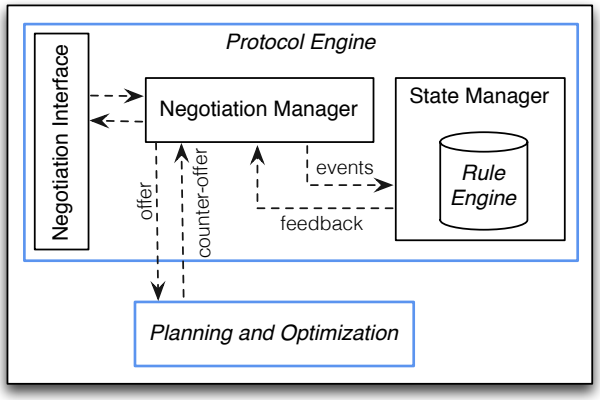


Figure 5.13.: Protocol Engine Design

the Negotiation Manager identifies the protocol to use from the SLA template parameter and asks the State Manager component to establish a negotiation session in memory. The State Manager uses an instance of the Drools rule engine

and loads the protocol file into its working memory. This bootstraps the corresponding state machines. As mentioned in the last section, the rule encoding scheme uses events passed by PE to the State Manager, which inserts them into the working memory of Drools. These events correspond to the incoming/outgoing messages. Drools then fires all qualifying rules when an event arrives, or an object is created or manipulated as a result of rule execution(s); a mechanism also known as forward-chaining. In this manner, the platform builds upon the comprehensive rule management capabilities of Drools. This loose coupling of protocol with the platform allows the protocol to regulate negotiation to the extent that even specific error reasons are returned to the caller as coded in rules.

Generic execution also required a strict separation of protocol and strategy. Hence, we do not overload rules to process an offer and generate counter offer(s). This activity is delegated to a loosely coupled Planning and Optimization (POC) component (Fig. 5.13) which implements a negotiation strategy. This way, multiple combinations of protocols and strategies are possible, which means that services can be sold using different negotiation protocols and strategies.

As seen in our negotiation interface, we use an SLA model for representing SLA template and SLA offers. The SLA model is described in detail in [58]. However, an important consideration here was to keep the SLA Manager framework decoupled from any specific SLA standard. This supports interoperability with current or future standards through a Syntax Convertor component. For instance, the Syntax Convertor achieved interoperability with WSAG based clients by i) providing an additional webservice end point for WSAG interface, and ii) internally translating the XML representation of WSAG calls/parameters into the Java representation of our SLA model and passing these on to the PE. This interoperability mechanism is explained in [72].

In addition to the negotiation interface, the platform also exposes a control interface to configure business rules representing organizational policy. The interface comprises of following methods:
setPolicies: Set the list of policy rules. This can create or overwrite existing rule set.
getPolicies: Fetch and return the list of policy rules.
Thus, access to rule engine allows on-the-fly modification of business rules (as in Fig. 5.11) that can rank, whitelist or blacklist customers, without recompiling or restarting the negotiation platform.

5.5. Summary

In this chapter, a comprehensive negotiation solution was presented. An argument was made that SLA negotiations provide a flexible and viable business model and concrete (differing) protocols serve as provider differentiators or a nursery for creating new markets around added-value service propositions. The contributions included a generic methodology to develop machine executable negotiation protocols. A concrete bilateral negotiation protocol was developed using the same methodology. Further more, a common negotiation platform was developed which can execute protocol rules written in a standard scheme. The platform separates concerns regarding protocols, strategies and specific SLA standards, which provides for adaptation to local requirements. The protocol and the platform were deployed in various use cases of the SLA@SOI project and the Contrail project (of which the author was not a part of). This testifies to the broader applicability of the proposed solution.

With the precedence set by Ebay and Amazon EC2 spot instances, this work furthers the means to offer configurable service offerings, where both customer and provider(s) are engaged through SLA negotiations. It was argued that the given approach can establish protocol-generic foothold in a system of marketplaces, where SLA negotiations are used to procure cloud based services or resources. Although single staged negotiations (although with multiple rounds) were addressed, future prospects include multi-staged protocols which can be used to create bundled IaaS, PaaS and SaaS service propositions based on mandatory or non-mandatory stages of negotiations.

Chapter 6.

Dynamic Negotiation Strategies

This chapter presents a solution to the second problem described in Chapter 4. This regards negotiation strategies to dynamically create SLAs of high business utility. The contents of this chapter make use of author's publication [47].

As argued earlier, QoS preferences of customer and provider may vary widely and even conflict. This is regarded as the "SLA-gap" problem, and is another reason why providers restrict their offerings to take-it-or-leave-it SLAs. Addressing diverse preferences through very large service catalogues is burdensome for the customer and can be counter-productive when offerings change. Negotiations reduce the SLA-gap in a dialectic session. As shown in Chapter 5, a negotiation protocol plays an essential mechanical role to bring the negotiating parties together. However, it must be complimented with a decision model. This chapter addresses negotiation time decision making through negotiation strategies.

In this work, the focus of a negotiation strategy is to optimize the business utility of an SLA for the customer or provider entity. This is a difficult problem because feature-rich SLA templates are composed of multiple QoS or non-functional issues having wide value ranges to select from. This means that contract spaces encapsulated by such templates can be very large. In real world, customers and providers do not share their objectives and their preferences on issues often conflict. For these reasons, such SLAs are not negotiable by humans in a couple of minutes. However, computer agents equipped with intelligent negotiation strategies provide a scalable solution to autonomously resolve conflicts in short times. To make the best deal for oneself, different strategies use different conceding tactics to adapt to opponent's conceding behavior and remaining time.

Thus, the aim in this work is to develop and analyze utility optimizing negotiation strategies, using which agents may efficiently produce feature-rich and high utility SLAs for themselves. Many strategies exist, as for instance seen

in [82, 83, 89–92]. However, performance evaluation of negotiation strategies has not been exhaustively studied in terms of estimating individual and global utilities. Therefore, in this work, the following aspects are duly considered:

- Negotiation scenarios that reflect broad kind of services.
- Varying degrees of conflict between customer and provider preferences.
- Simulating diverse markets composed of state of the art strategies, where an agent has no knowledge of opponent's strategies, objectives or preferences.

The following concrete contributions are made:

1. Two negotiation strategies (algorithms) are presented, which maximize an agent's gain (utility) by adapting to opponent behavior and time.
2. Experimental evaluations that prove the robustness of proposed strategies in a variety of markets composed of top performers in past Automated Negotiating Agents Competition (ANAC) [79, 80].

6.1. Related Work

The popularity of service oriented infrastructures such as grids and clouds has led to an active interest in negotiating service SLAs. To date, SLA negotiations have been mostly studied in relation to resource allocation and service composition problems in grids, web services and eCommerce domains, but getting much attention in recent cloud based projects [73].

In [85], a Q-learning based negotiation strategy is developed for scheduling grid resources. [103] reduces negotiation time decision making to determine feasibility of requested resources in IaaS clouds by means of binary decision diagrams. Agent FSEGA [88] uses Bayesian learning to negotiate SLAs for computational grid scenarios. [84] has developed a Game Theory based strategic negotiation model and applied it to data and resource allocation problems. However, dynamic negotiation of cloud SLAs in chained scenarios (as considered in Section 4.1) differs from these approaches because resource limitation is generally not the restraining factor. The important consideration in this work is to assess the business utility of cloud SLAs from the perspective of both customer and provider in a domain-independent manner.

In his seminal work [82], Raiffa has systematically studied various aspects of negotiation strategies including utility functions, deciding under uncertainty and the role of time. In [83], parameterized polynomial and exponential functions are given to model conceding behavior of agents along time. These led to Boulware, Conceder or Linear Conceder tactics. These definitions have since become mainstream in negotiation literature and many variants have been developed.

Opponent modeling through learning techniques has received special attention in adaptive strategy design. In [86], kernel density estimation is used for eliciting issue preferences of opponent. Similarly, in [87], a scalable Bayesian algorithm is used for similar purpose. Negotiation strategy algorithms proposed in prior works are often hard to reproduce and apply in different settings. This gap is adequately filled by GENIUS - a simulation platform to create negotiation scenarios and benchmark strategies [77] against various utility metrics. GENIUS is used for ANAC [79, 80] competitions and contains a rich repository of strategies [78] so evaluation results can be reproduced. It has so far been used to simulate SLA negotiations in computational grids [88], while the author has used it for SLA negotiation in cloud scenarios to analyze learning and non learning strategies [47].

Some recent strategies disseminated as part of the GENIUS platform are briefly described here. AgentK [89] analyzes the mean and variance of opponent's bids to target a time varying utility value, accepting a bid if improvement is not expected. HardHeaded [90] models an opponent's utility function by analyzing the frequency of bids' values. CUHK [92] uses sparse pseudo-input Gaussian processes to model the behavior of opponent. MetaAgent [91] is an interesting (meta)strategy, which uses neural network to predict performance of various strategies for a given domain. It uses the strategy predicted to outperform others. Learning based strategies usually rely on maintaining a hypothesis space relative to the size of negotiation domain, which is progressively refined with the arrival of each new bid per negotiation round. This poses a problem with very large negotiation domains. Hence, these methods require scalable variants but those may not necessarily converge towards the desired outcome [88]. These techniques also increase processing load on the negotiation platform.

The evaluation report of the second ANAC competition [80] encourages further research on Tit-for-Tat strategies as these have the property to lure opponents into concession making in order to get one. The strategies proposed in this work, namely the *Reactive Exploitation* and the *Enhanced Reactive Exploitation*

attempt this through a time and behavior dependent tactic, thus extending the art on Tit-for-Tat strategies.

Most negotiation works evaluate strategies using utility as a single metric, while we evaluate the performance of strategies against a variety of metrics including utility, social welfare, social utility and Pareto-optimal bids. Hence, we take individual, mutual and relative gain into account. Secondly, results from ANAC [80] and the author's work [47] show that no single strategy guarantees a wide success margin. Therefore, two set of experiments are devised in this work with the objective to systematically investigate how state of the art strategies fare against proposed strategies. The experiments consider markets composed of non-learning, learning and mixed strategies. Due consideration is paid to preference conflicts among negotiators, minimum acceptable and time-mutable utilities as well as preferential dependence/independence among negotiable issues.

6.2. Negotiation Domain

An SLA template consists of multiple issues (e.g., QoS) whose value ranges may encapsulate a large contract space. This forms a negotiation domain [77] and negotiation is used to create an SLA, where issue values are fixed. From the perspective of SaaS or PaaS cloud providers, we identify three issues which are vital to customers. These are Availability, Performance and Backup, defined as:
Availability: the probability that a service is up and running. Formally, it is defined as $(T-d)/T$ where T is the monitored time during which total downtime was d.
Performance: relation $< ar, t_p >$ where t_p is the maximum time to return the response of a request, received in time period p where the maximum request arrival rate (throughput) ar is not exceeded.
Backup: the ability to automatically archive code and data of an application at specific times.

Price is not a negotiable issue here as we are interested in quickly resolving conflicts on QoS issues. However, it can be treated in similar fashion as a negotiable issue. Note that for performance, throughput and response time are structured together in a vector to respect preferential independence among these issues. This being a necessary condition to use linear additive utility function (explained in the next section). Container technologies position a PaaS provider

Table 6.1.: Cloud Computing Domain 1

Negotiation Domain			
Issue	**Unit**	**Value Range**	**Selection Constraint**
Availability	percentage	51% to 99%	increment by 4 till 83% then by 1
Performance	<requests_per_second , response_time(ms)>	<10,100> *to* <90,2000>	<increment by 10 , increment by 100>
Backup	time	{never,daily,weekly, bi-weekly,monthly, bi-annually,annually}	choose one

to treat performance as a superfluous issue [82] to create conceding illusions during negotiation. These issues together formulate an SLA template shown in Table 6.1 . This serves as the first cloud computing domain used for the experiments in Section 6.5.

6.3. Utility Spaces

Given a negotiation domain, the next step is to create customer and provider profiles (roles). This associates a utility to each possible bid.

6.3.1. Preference Profiles

For a bid $b = \{x_1, ..., x_N\}$, where x is the chosen value of N issues, the utility value $u(b)$ is given by the linear additive utility function:

$$u(b) = \sum_{i=1}^{N} w_i V_i(x_i) \tag{6.1}$$

where w_i is the weight for issue x_i such that $\sum_{i=1}^{N} w_i = 1$ and

$$V_i(x_i) = eval(x_i)/max(eval(x_i)) \in [0, 1] \tag{6.2}$$

is the normalized evaluation (real) value for the chosen value x_i of the i^{th} issue. *eval* is an evaluation function which assigns a utility to an issue value. Providers and customers compute V per own objectives. Thus, *eval* provides a hook for domain specific interpretation of an issue value while V is its normalized value.

For instance, a customer wanting the highest throughput and lowest response time may associate a best evaluation value of 290 to issue value $< 90, 100 >$,

Table 6.2.: Preference Profiles for Cloud Computing Domain 1

Issue	Customer Evaluation			Provider Evaluation		
	Worst	**Best**	**Weight**	**Worst**	**Best**	**Weight**
Availability	50	99	0.45	51	99	0.33
Performance	0	290	0.28	14	100	0.34
Backup	3	9	0.27	2	10	0.33

while a worse evaluation of 0 to $< 10, 2000 >$, and so on. Issues are assigned weights that reflect relative priorities of each issue. Evaluation function and weights together constitute a preference profile, which holds a utility space for customer and provider. The provider and customer profiles of cloud computing domain 1 are shown in Table 6.2. Here, the best and worst column show the value ranges used by the customer and provider *eval* functions.

6.3.2. Preference Conflicts

In the cloud computing domain 1, preferences modeled as priority weights are assigned considering standard interests of customer and provider. Customers value highly available and high performing service, while a provider would require significant investment to offer the same, hence it associates a lower priority to higher evaluations of availability or performance and vice versa.

Let $\omega \in \Omega$ and Ω is the utility space representing all outcomes or bids. Customer and provider's utility functions are used to map each ω to a real value in range [0,1]. This value is plotted as a single red dot on the negotiation chart as shown in Fig. 6.1, where the proposed strategy (ReactiveExploiter - presented in Section 6.4) negotiates with the CUHK agent [92]. Each point represents a possible SLA while the red curve is the Pareto-frontier of this negotiation domain. $\bar{\omega}$ is the highest (but impossible) payoff point at position (1,1). The relatively extended frontier (spanning 0.27 to 0.31 along x and y axis) and the large gap from $\bar{\omega}$ indicates a strong opposition between the opponents. The blue and green symbols show the bids exchanged by respective agents. The bigger red square shows the actual agreement (SLA).

6.3.3. Negotiation Dynamics and Metrics

An agreement struck on the Pareto-frontier is one measure of optimality, however as shown in Fig. 6.1, it may be highly biased in favor of one agent, as opposed to

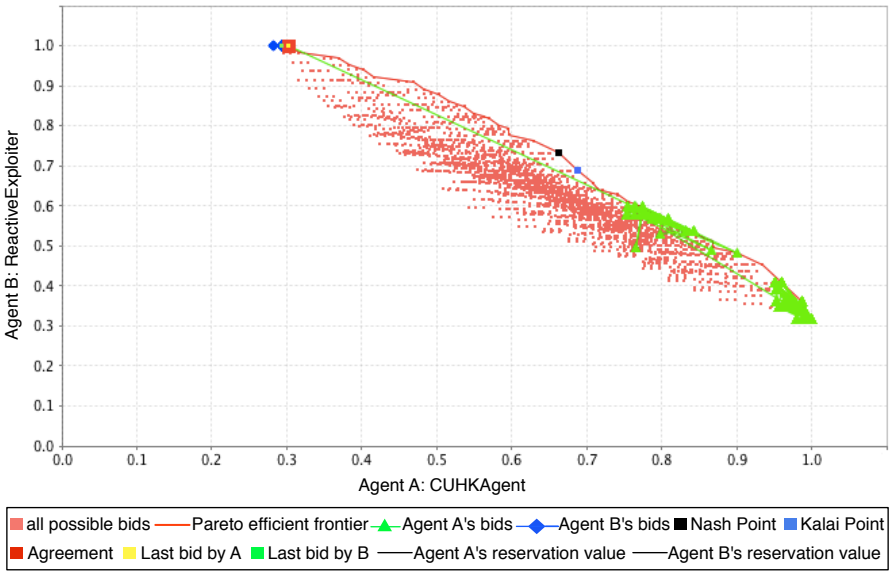


Figure 6.1.: ReactiveExploiter vs CUHK

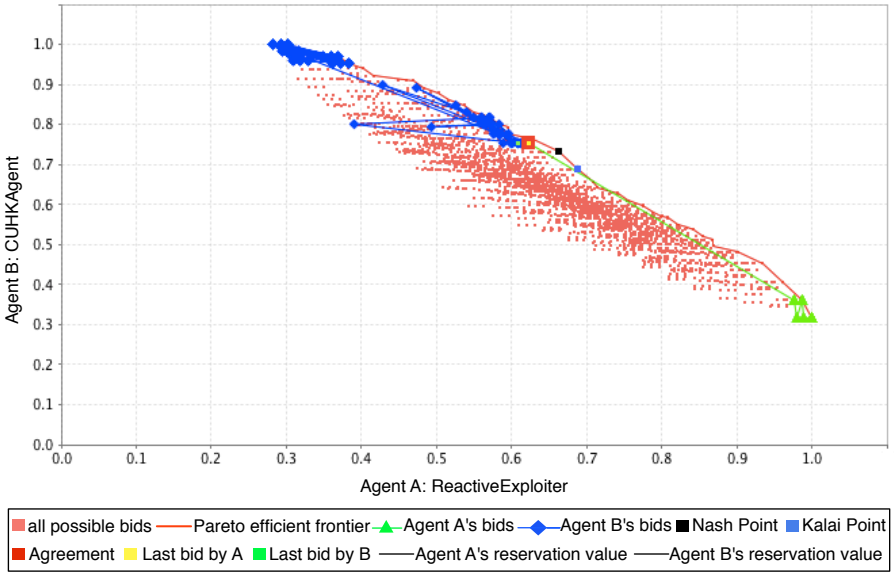


Figure 6.2.: ReactiveExploiter vs CUHK

Fig. 6.2, where it is rather close to the mid-section. The Nash point, shown as black square represents an agreement where the product of utilities of both agents is maximum. The Kalai-Smorodinsky point shown in blue square represents an agreement with maximum fairly proportioned utilities for both agents.

For our experiments, we consider the utility scored by an agent in a tournament as the measure of competitiveness. Cloud computing negotiation scenarios hold non zero-sum domains, because an agreement reached is a win-win for both customer and provider. In such domains, the importance of mutual benefit cannot be neglected. Therefore, to determine cooperativeness, we consider social welfare, which is the sum of utilities of two agents. We consolidate utility and social welfare by adding them into a single social utility metric, which is considered to loosely determine fairness. Finally, the percentage of Pareto-optimal bids (referred as Pareto bids) is considered to analyze relative optimality.

6.4. Reactive Exploitation Negotiation Strategy

Generally, a negotiation strategy S can be modeled abstractly as a 3-tuple:
$S = (B, M_o, A)$
where B is the bidding function that generates a counter offer in response to opponent model and time, M_o is the opponent model based on received bids and/or opponent moves, and A is the acceptance function which decides to accept a bid and conclude negotiation. Prior art shows a depth of variety and complexity induced by these individual functions (Section 6.1). Consequently, this adds to the computational load on negotiation platform. One of our goals is to investigate if computationally inexpensive yet reasonably adaptive functions can also produce effective results. This led to the design of Reactive Exploitation negotiation strategy. Its algorithm is shown as a flow chart in Fig. 6.3. The detailed description of algorithm is provided in Appendex A.1.

Reactive Exploitation is a *quasi* Tit-for-Tat strategy which produces concessions if the opponent is perceived to be doing the same and vice versa. The *quasi* part stems from how lenient or strict its behavioral parameters (α and β) are configured, which may lead to wide spectrum of behaviors. For our experiments, we set $\alpha = 2$, $\beta = 0.6$ and $u_r = 0.6$. These settings afford it a Boulware [83] attitude, which is a property of demanding agents. The agent which implements the strategy is referred to as the ReactiveExploiter or shortly as RE.

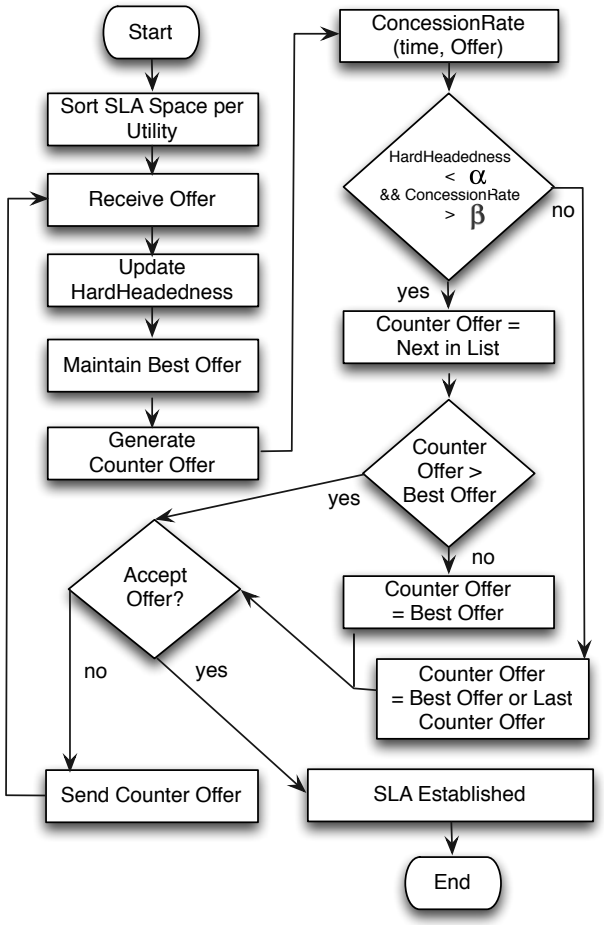


Figure 6.3.: Reactive Exploitation Strategy

6.5. Experimental Evaluations

Using SLA negotiations for service procurement means that stakeholders with better performing strategies would meet their business objectives better than others. Qualitative analysis of negotiation simulations in service markets composed of formidable opponent strategies helps to competitively position oneself.

Before conducting agent-to-agent negotiations, it was of interest to validate how a human would perform if negotiating with an agent. With only three issues, the cloud computing domain is not too small and consists of 16,380 possible SLAs (or bids). The author (who is also the creator of the domain) negotiated with the CUHK agent [92] for 2 minutes. The result showed that this indeed

poses a challenge for human as each received bid can be mesmerizing. This burden on a human is quantified in Fig. 6.4. Despite complete information and subsequent concessions by human indicated by the drop in red line from 1.0 to 0.7, only 83 negotiation rounds were exchanged. This is mostly due to human think-time or emotion. On the other hand, the agent can deal with complexity far more efficiently, indicated by the blue line which lacks reciprocity to human concessions until the very end. Repeated attempts made negligible improvements and negotiations broke off i.e. no agreement was reached in time. This starkly differs from agent to agent negotiations, where rounds may scale to thousands or even millions - a property which influences chances of convergence.

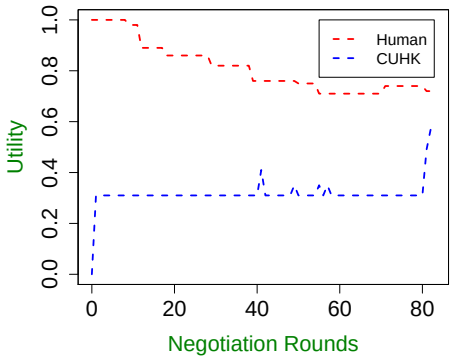


Figure 6.4.: Agent vs Human Negotiation - Utility for Human

6.5.1. Tournament Design

Tournament based experiments were devised to evaluate the robustness of Reactive Exploitation strategy against state of the art competitors. Tournaments were conducted using GENIUS, where strategies written by various researchers are available as agents. In a tournament, all strategies compete against each other using both the customer and the provider preference profiles. A tournament thus represents a market composed of diverse strategies. A tournament ran each negotiation session (i.e. between specific agent-pairs) four times; where an agent acted twice as customer and twice as provider. GENIUS orchestrated negotiations between agents using Rubinstein's alternating offers bilateral protocol over the cloud computing domain 1. Each negotiation session was held for 2

minutes. Mean values for utility along with standard deviations were collected. The values for social welfare, social utility, percentage of Pareto-optimal bids and number of negotiation rounds exchanged are also collected. It was of interest to investigate how non-learning, learning and mixed strategies would impact the SLA utility achieved by an agent/strategy. Thus, four tournaments were conducted as following.

Tournament 1

This tournament includes Boulware, Linear Conceder and TheNegotiator agents, all of whom do not employ learning to elicit an opponent's preferences. These strategies rely on time to generate concessions. TheNegotiator [104] - a rather recent strategy acts hardheaded till a predetermined time, after which it turns more concessive. Fig. 6.5(a) and (b) show a two-sided X-Ray view of concessions given and received respectively when RE negotiates with opponents as a provider.

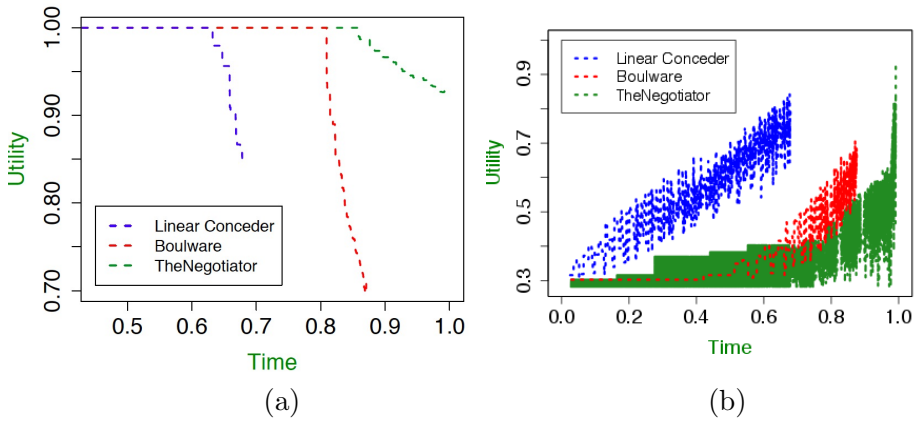


Figure 6.5.: (a) Concessions given by RE (b) Concessions received by RE

Fig. 6.5(a) exhibits a tough Boulware-like behavior, where RE concedes only after more than half the time has elapsed. A cautious tit-for-tat response is observed where RE does not give in too quickly. Fig. 6.5(b) confirms that Linear Conceder concedes linearly till it approaches its reserved utility, while Boulware and TheNegotiator follow exponential rates. The latter gets highly active but only after 80% of time has elapsed. The behavior of RE is similar when acting as

a customer. The final results are shown in Table 6.3 and Fig. 6.7(a). RE emerges as the winning strategy from tournament 1. TheNegotiator ranks second.

Table 6.3.: Tournament 1 Results

Strategy	Utility	Std.Dev.	Social Welfare	Social Utility	Pareto Bids (%)	Rounds
Boulware	0.7213	0.0	1.3802	2.1015	0.6054	6127754
ReactiveExploiter	0.7943	0.0017	1.3781	2.1724	0.9767	5386467
TheNegotiator	0.7286	0.0023	1.3760	2.1047	0.3194	1255169
LinearConceder	0.5114	0.0012	1.3771	1.8885	0.0663	5539499

Tournament 2

This tournament includes learning-based agents. The participating agents are top performers of ANAC competitions and hence considered state of the art. These include AgentK (winner 2010), HardHeaded (winner 2011), CUHK (winner 2012) and the runner up of 2013 - the MetaAgent[1]. These agents employ complex opponent learning and/or prediction models as well as mixed conceding tactics, making negotiations very competitive. The final results are shown in Table 6.4 and Fig. 6.7(b). CUHK takes the first position, while HardHeaded ranks second.

Table 6.4.: Tournament 2 Results

Strategy	Utility	Std.Dev.	Social Welfare	Social Utility	Pareto Bids (%)	Rounds
AgentK	0.6274	0.0246	1.3748	2.0022	0.1711	1136971
HardHeaded	0.6926	0.0135	1.3745	2.0672	0.4110	2745242
CUHK	0.7402	0.0453	1.3664	2.1067	0.4839	2441479
MetaAgent	0.6829	0.0336	1.3705	2.0534	0.4778	1845314

Tournament 3

This tournament represents a mix of best non-learning and learning strategies for our domain. The top two contestants from previous two tournaments, which ranked best for utility (competitiveness measure) and social utility (fairness measure) were the criteria to qualify for this tournament. Fig. 6.6(a) and (b) reveal the concessions given and received respectively, as RE negotiates with opponents as a customer. A notable difference from Fig. 6.5(a) and (b) is that now the

[1]We chose MetaAgent since negotiations broke off with the winner TheFawkes. The difference in their performance is statistically negligible.

agents negotiate hard with major concessions appearing after 85% of time has elapsed. The amount of concessions produced is much higher seen by an almost overlap of colors. TheNegotiator extracts concession from RE earlier than

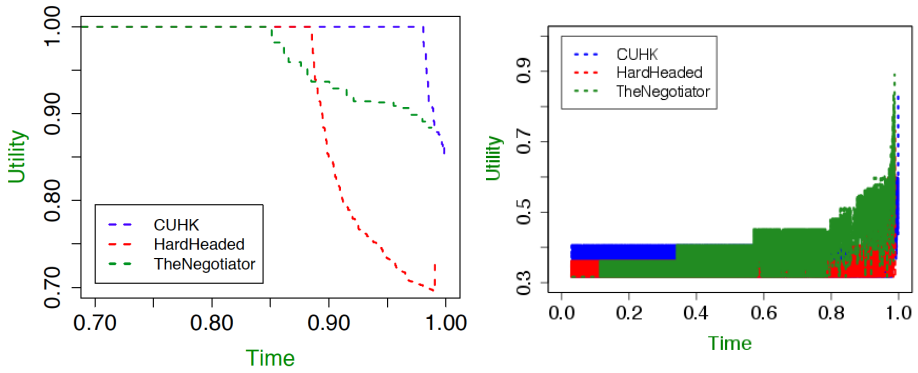


Figure 6.6.: (a) Concessions given by RE (b) Concessions received by RE

HardHeaded or CUHK, but only manages to minimally lower RE's utility. Negotiation with HardHeaded provide an interesting insight. Note the red line indicating concessions extracted by HardHeaded takes a steep dive but just as the time is nearly finished, it starts an upward surge as the opponent's bids start improving RE's utility. This is indeed the result of (win-win or fortunate) bids found on the iso-curve [82]. This serves as a proof of RE's rationality as RE picks up this increase in utility, deflecting from its usual Tit-for-Tat tactic in order to maximize its payoff. Finally, we see that CUHK is comparatively the most demanding strategy which starts to give in when the time is almost ending seen by the blue line. However, the acceptance function of RE and CUHK help avoid a break off. The behavior of RE is similar when acting as a provider. The final results are shown in Table 6.5 and Fig. 6.7(c). Here, CUHK takes the first place and RE ranks second.

Tournament 4

This tournament brings all eight agents together. Hence it represents a market constituted of widely heterogeneous strategies, from the weakest to the strongest strategies. The intention here is to analyze how this diversity effects the out-

Table 6.5.: Tournament 3 Results

Strategy	Utility	Std.Dev.	Social Welfare	Social Utility	Pareto Bids (%)	Rounds
CUHK	0.7543	0.0036	1.3750	2.1293	0.0137	3208519
ReactiveExploiter	0.7149	0.0075	1.3525	2.0674	0.9701	4136798
HardHeaded	0.6882	0.0039	1.3522	2.040	0.447	3732350
TheNegotiator	0.5687	0.0113	1.3725	1.9412	0.3131	1170602

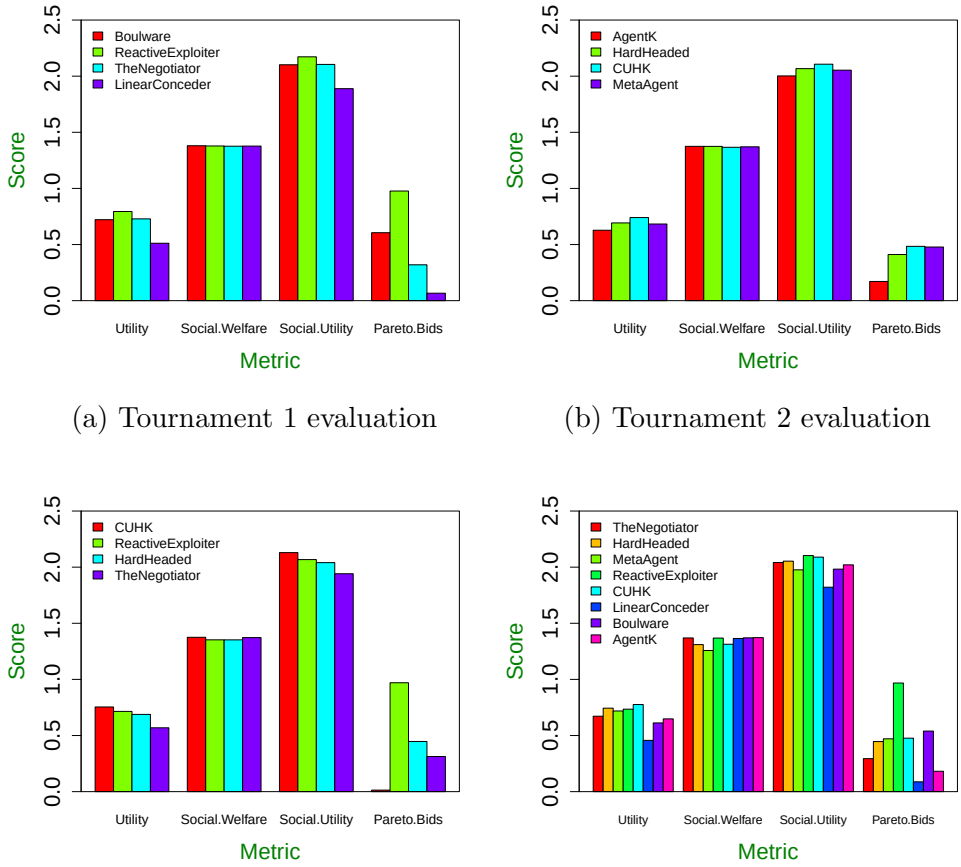
Table 6.6.: Tournament 4 Results

Strategy	Utility	Std.Dev.	Social Welfare	Social Utility	Pareto Bids (%)	Rounds
TheNegotiator	0.6719	0.0162	1.3692	2.0410	0.2938	1061222
HardHeaded	0.7437	0.0372	1.3099	2.0535	0.4463	4199962
MetaAgent	0.7186	0.0016	1.2575	1.9761	0.4708	2289329
ReactiveExploiter	0.7349	0.0021	1.3683	2.1032	0.9677	4595223
CUHK	0.7761	0.0658	1.3133	2.0894	0.4765	3048744
LinearConceder	0.4568	6.7877	1.3645	1.8214	0.0874	4051447
Boulware	0.6123	1.0261	1.3702	1.9824	0.5397	5001335
AgentK	0.6483	0.0223	1.3722	2.0205	0.1815	1067989

comes. The final results are shown in Table 6.6 and Fig. 6.7(d). The CUHK scored best in terms of utility, followed by HardHeaded and RE. For social utility, RE scored best followed by CUHK and TheNegotiator.

6.5.2. Discussion

Considering tournament results already presented as tables, market level performance can be examined. This is possible by computing the best to worst utility lead i.e., percentage utility of the best performing strategy by which the worst performing strategy lags behind it. This lag is largest in tournament 1 at 35%, compared to 15% in tournament 2, 24% in tournament 3 and 21% in tournament 4. This confirms that learning indeed increases market competitiveness. However, the mean utility of markets stand at 0.688, 0.685 and 0.681 in tournaments 1, 2 and 3 respectively, with a minimal decrease in tournament 4 at 0.67. This shows that overall, the markets remain stable even with the presence of weaker strategies. Although weaker strategies fared worse, no agent quitted or crashed in any tournament. The convergence rate was 100%. Tournament 4 also scored low for mean social welfare value at 1.34, while that of tournament 1, 2 and 3 stood at 1.377, 1.371, 1.363. This indicates that the joint gains achieved by markets of larger size and wider diversity of strategies will be lower. This is reinforced by mean social utility values, which are 2.066, 2.057, 2.044 for tournaments 1, 2 and 3 but 2.01 for tournament 4.



(a) Tournament 1 evaluation

(b) Tournament 2 evaluation

(c) Tournament 3 evaluation

(d) Tournament 4 evaluation

Figure 6.7.: Performance Results

RE uses a simplistic single parameter opponent model h representing the perceived hardheadedness (non conceding behavior). This, alongwith a Boulware-oriented utility-decreasing bidding function prevents it from freefall concessioning. Void of complex learning method, RE exploits its opponent's learning skill to retain high utility bidding. This is evident from the high percentage of generated bids that lied on the Pareto-frontier, in which terms it ranks best in all participated tournaments. The results also show that RE secures a wider utility margin over weaker opponents, while not breaking off any negotiation against stronger opponents. Opponents against whom it lost individually in terms of

utility, only won with a maximum margin of 21%, while those against whom RE won, lost with a maximum margin of 57%. RE won some matches against CUHK, while loosing all against HardHeaded and Boulware although by a maximum margin of only 16% and 18% respectively. RE won all matches against TheNegotiator, Linear Conceder, AgentK and MetaAgent. This way, it accumulated enough overall advantage to assert a dominating position. RE's strong performance in social utility shows that agreements had a fair distribution of utility, which entails partner satisfaction.

6.6. Further Domains, Reserved and Discounted Utilities

From this section onwards, evaluations are extended to include further negotiation domains. A wider spectrum of preference conflicts are considered which represent different service markets. Preference profiles having a fixed reserved utility value are also considered. The reserve utility is the minimum utility value below which an agent will not make agreement. Some preference profiles also contain discount factor. This decreases the utility of bids with time at a rate determined by the discount factor. Since negotiations can be computationally expensive, domains (or SLA templates) with discounted utilities force agents to quickly converge on an SLA because waiting only reduces the SLA's utility.

As a starting point, the cloud computing domain 1 is revised to provide fine grained values for Availability. This increases the possible SLAs to 35,280. The updated domain namely, cloud computing domain 2 is shown in Table 6.7. Its provider and two customer preference profiles are shown in Table 6.8.

Table 6.7.: Cloud Computing Domain 2

Negotiation Domain			
Issue	**Unit**	**Value Range**	**Selection Constraint**
Availability	percentage	51% to 99%, 99.9%, 99.99%, 99.999%	increment by 4 till 83%, then by 1 upto 99%
Performance	<requests_per_second , response_time(ms)>	<10,100> *to* <90,2000>	<increment by 10 , increment by 100>
Backup	time	{never,daily,weekly, bi-weekly,monthly, bi-annually,annually}	choose one

Table 6.8.: Preference Profiles for Cloud Computing Domain 2

Issue	Provider Evaluation			Customer 1 Evaluation			Customer 2 Evaluation		
	Worst	Best	Weight	Worst	Best	Weight	Worst	Best	Weight
Availability	48	99	0.33	50	100	0.45	48	99	0.3
Performance	14	100	0.34	0	290	0.35	0	290	0.4
Backup	2	10	0.33	3	9	0.2	3	9	0.3

6.6.1. Negotiating Mission Critical vs Fault-tolerant Batch Services

Customer 1's profile models a mission-critical service where the customer requires high availability and high performance. Similar to the first domain, the customer associates its best evaluation values to higher issue values of availability and performance. The priority weights for availability are 0.45, for performance 0.35 and backup merely 0.2. The provider associates worst evaluation values to such demanding issue values albeit with uniform priorities. This constitutes a high-conflict utility space shown in Fig. 6.8. Here, another proposed strategy (the eReactiveExploiter - presented in the next section) is shown against the CUHK [92] agent. The relatively extended frontier (spanning 0.27 to 0.29 along x and y axis) and the large gap from the maximum irrational payoff point $\bar{\omega}$ indicates a strong opposition between the opponents.

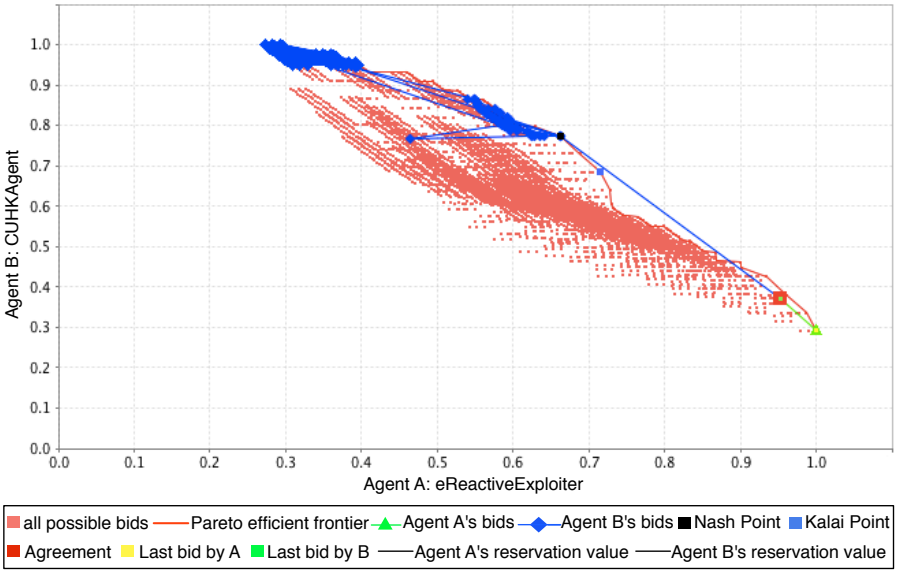


Figure 6.8.: eReactiveExploiter vs CUHK (using customer 1 profile)

Customer 2's profile models a fault-tolerant (batch) service suitable for Spot Instance markets e.g., to perform offline analysis using MapReduce jobs. These do not require high availability per se. These do however require high performance so maximum processing can be performed in the duration that resources are available. Customer 2's evaluation function associates best evaluation values to higher issue values for performance and shares the evaluation function of provider for availability issue values. Hence, the conflicts with the provider are reduced mostly to performance and the utility space has a lower conflict as seen in Fig. 6.9. Here, the Pareto-frontier is comparatively shorter (spanning 0.45 to 0.4 along x and y axis) and has a shorter gap from $\bar{\omega}$, indicating lesser opposition between the opponents. In Fig. 6.9, eReactiveExploiter is shown against HardHeaded [90] agent.

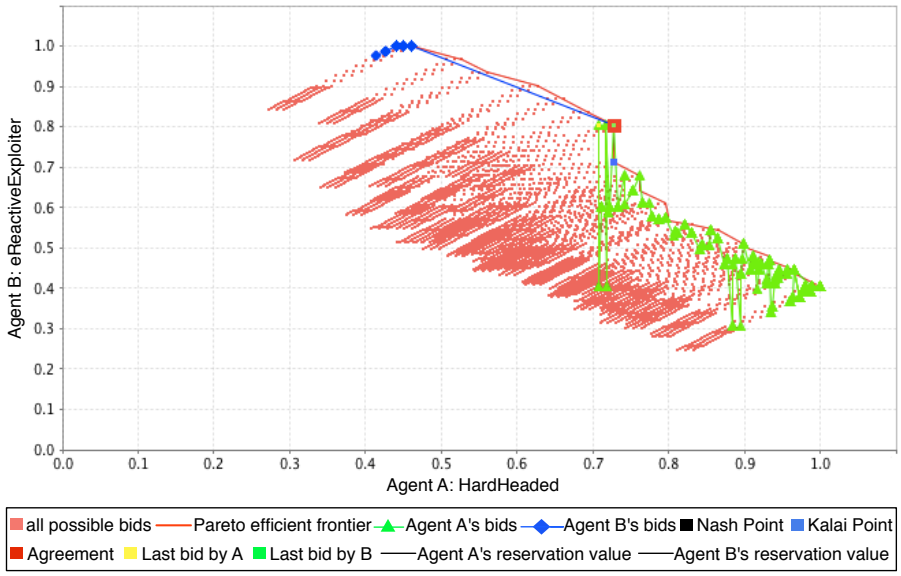


Figure 6.9.: eReactiveExploiter vs HardHeaded (using customer 2 profile)

6.7. Enhanced Reactive Exploitation Negotiation Strategy

It was of interest to investigate if computationally inexpensive yet reasonably adaptive strategies can also produce effective results in larger domains. Hence, we identify the periphery between learning and non-learning methods as a possible exploration area to simplistically model an opponent. This is coupled with

reactive tactics to exploit an opponent's learning skill such that high utility bidding is retained for oneself. Reactive Exploitation performs best under certain limitations. Reusing its exoskeleton, the *"Enhanced Reactive Exploitation"* negotiation strategy was developed. Its algorithm is shown as a flow chart in Fig. 6.10. The detailed description of algorithm is provided in Appendix A.2. Concretely, the strategy provides the following improvements:

1. It is designed for large domain spaces possessing billions, quintillions and novillions of agreements. This is done by using random search to selectively refine partial bid space which is progressively explored in negotiation rounds. Hence, it does not require a sorted bid space at negotiation startup.

2. A max-min-max method is introduced which attempts to create diverse nature of concessions (counter offers) to the opponent.

3. It uses an acceptance function which estimates remaining number of negotiation rounds. This, along with fixed time thresholds help it converge.

Like its predecessor, the strategy implements Tit-for-Tat behavior, which attempts to produce concessions if the opponent is perceived to be doing the same and vice versa. Its leniency or strictness is governed by behavioral parameters (α and β), which may lead to wide spectrum of behaviors. For our experiments, we set $\alpha = 4$, $\beta = 0.6$ which affords it a Boulware [83] attitude. The agent which implements the strategy is referred to as eReactiveExploiter or shortly as eREx.

6.8. Experimental Evaluations

Systematic experiments were designed to evaluate the robustness of eReactiveExploiter against state of the art competitors in four tournaments. The setup of the tournaments (i.e. negotiation time and performance metrics) is same as that for evaluating the first strategy, however different domains are targeted.

6.8.1. Tournament Design

The high level characteristics of all negotiation domains used are shown in Table 6.9. With the exception of cloud computing domain 2, all others are taken from prior art as available in GENIUS distribution (version 5.1). The motivation is to expand evaluations to diverse domains, with reserved utilities, discounts,

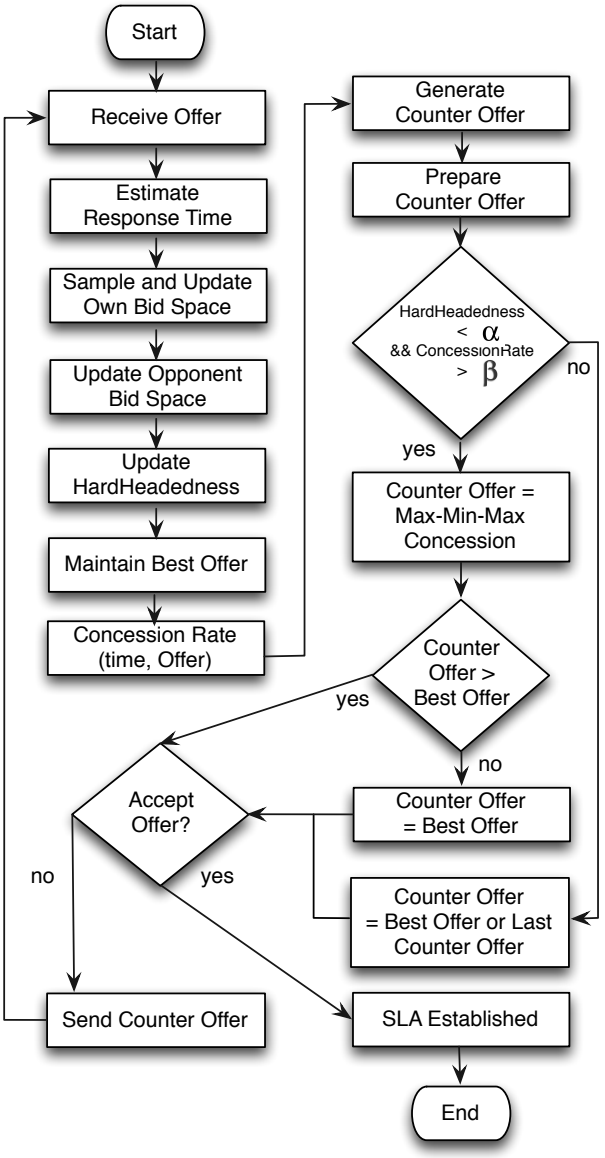


Figure 6.10.: Enhanced Reactive Exploitation Strategy

and issue dependencies. Four tournaments are designed using top performing strategies from the past ANAC competitions.

The first three tournaments include AgentK (winner 2010), HardHeaded (winner 2011), CUHK (winner 2012) and the runner up of 2013 - the MetaAgent[2].

[2]We chose MetaAgent as negotiations broke with the winner TheFawkes. The difference in their performance is statistically negligible.

Table 6.9.: Characteristics of Domains used in Experiments

Name	Negotiable Issues	Possible Contracts	Reservation Value (profile1, profile2)	Discount Factor (profile1, profile2)
Cloud Computing	3	35,280	0.0 , 0.0	1.0 , 1.0
Flight Booking	3	48	0.25 , 0.25	1.0 , 0.75
Camera	6	3,600	0.25 , 0.25	1.0 , 1.0
Smart Phone	6	12,000	0.6 , 0.6	0.87 , 0.87
S-1NIKFRT-1	10	10^{10}	0.6 , 0.5	0.95 , 0.95
S-1NIKFRT-2	20	10^{20}	0.2 , 0.1	0.9 , 0.9
S-1NIKFRT-3	30	10^{30}	0.0 , 0.0	1.0 , 1.0

For the fourth tournament, all non-linear compatible agents[3] were used. These include AgentK, AgentK2, Gahboninho and AgentKF. All these agents employ complex opponent learning or prediction models as well as varying conceding tactics, making negotiations quite hard.

Tournament 1

This tournament uses the cloud computing domain 2, where agents use the provider profile and customer profile 1 (Table 6.8). The objective in this tournament is to assess how eReactiveExploiter performs in a high-conflict market where SLAs for mission-critical services are negotiated. The final results are shown in Table 6.10 and Fig. 6.14(a). Here, the agent negotiates very competitively against its opponents as evident by its first place in terms of individual utility. However, this increased competitiveness comes at the cost of decreased cooperation as seen with low values for social welfare and social utility. AgentK ranks second for all three metrics. Finally, for Pareto-optimal bids, the eReactiveExploiter takes lead and MetaAgent lands second.

Table 6.10.: Tournament 1 Results

Strategy	Utility	Std.Dev.	Social Welfare	Social Utility	Pareto Bids %	Rounds
CUHK	0.5782	0.0266	1.2694	1.8476	0.57	1668444
HardHeaded	0.6149	0.019	1.2112	1.8261	0.5603	1683243
eReactiveExploiter	0.6759	0.0474	1.085	1.7609	0.6889	82267
MetaAgent	0.5632	0.0647	1.2714	1.8346	0.6231	1438826
AgentK	0.629	0.0353	1.2856	1.9146	0.1686	727874

[3]We excluded SlavaAgent because it threw runtime exceptions.

Tournament 2

This tournament uses the customer profile 2 and provider profile of the cloud computing domain 2 (Table 6.8). The objective here is to assess eReactiveExploiter in medium or low-conflict markets, where SLAs for fault-tolerant batch services are negotiated. The final results are shown in Table 6.11 and Fig. 6.14(b). Here again, eReactiveExploiter negotiates competitively and scores best for utility, followed by MetaAgent. It scores the least for social welfare but lands higher than AgentK for social utility. The CUHK and MetaAgent rank the best for social welfare and social utility respectively. For Pareto-optimal bids, eReactiveExploiter maintains its lead as in the first tournament.

Table 6.11.: Tournament 2 Results

Strategy	Utility	Std.Dev.	Social Welfare	Social Utility	Pareto Bids %	Rounds
CUHK	0.681	0.0162	1.4652	2.1462	0.4888	1942954
HardHeaded	0.7003	0.0134	1.4638	2.1641	0.0415	1550977
eReactiveExploiter	0.7688	0.0325	1.352	2.1208	0.6771	84985
MetaAgent	0.7287	0.0184	1.4544	2.1832	0.4962	1173326
AgentK	0.6727	0.1082	1.3677	2.0404	0.1113	658399

Tournament 3

In this tournament, we expand our evaluations to three domains that are disseminated in the GENIUS platform. These model business to customer (B2C) service propositions e.g., making a booking, buying a commodity or product driven tariff over online shops. Specifically, the domains are for booking a flight, buying a camera, and a smart phone with carrier options (see Table 6.9). The size of these domains is small, which allows to examine performance on smaller domains. Each preference profile in these domains imposes a reserved utility value u_r below which an agreement cannot be made. In flight booking and smart phone domains, a discount factor $d \in [0,1]$ is applied at current time t_c. When a discount factor is applied, the utility of an outcome ω gets discounted as:

$$u_d^t = u(\omega).d^{t_c} \tag{6.3}$$

Discounts with $d > 0 \wedge d < 1.0$ reduces the utility of bids as well as that of reserved value with time. This poses stress on agents to converge quickly. If $d = 1$, utility is not affected.

The flight booking domain (Fig. 6.11) reveals an outcome space so small that with few concessions, an agreement can be reached. Nevertheless, the Kalai point shows that one agent would always be better off than the other. The camera domain (Fig. 6.12) presents a wide outcome space with a relatively large Pareto-frontier. The Nash and Kalai points show that utility difference between opponents considering ideal agreements cannot be much biased. The smart phone domain (Fig. 6.13) shows a somewhat shorter Pareto-frontier with most outcomes located outside the region of reserved utility values. This requires agents to exhaustively explore the outcome space in order to reach mutually beneficial agreements. The opposition in all three domains is of medium nature.

The final results are shown in Table 6.12 and Fig. 6.14(c). The eReactiveExploiter takes first place for utility and HardHeaded ranks second. For social welfare, HardHeaded ascends to the first position while eReactiveExploiter ranks fourth, one step higher than CUHK. For social utility, eReactiveExploiter ranks second while HardHeaded leads. For Pareto-optimal bids, the CUHK ranks first and eReactiveExploiter ranks second in this tournament.

Table 6.12.: Tournament 3 Results

Strategy	**Utility**	**Std.Dev.**	**Social Welfare**	**Social Utility**	**Pareto Bids %**	**Rounds**
CUHK	0.7112	0.0324	1.4142	2.1254	0.9776	4585226
HardHeaded	0.7541	0.0166	1.5854	2.3395	0.9021	4417487
eReactiveExploiter	0.841	0.0033	1.4367	2.2777	0.9457	163227
MetaAgent	0.7192	0.0059	1.5278	2.247	0.7818	3270915
AgentK	0.6831	0.0015	1.4529	2.1359	0.4172	2241376

Tournament 4

The first three tournaments represented linear domains i.e., their issues were preferentially independent of each other and GENIUS used the linear additive utility function (see Equation 6.1) to compute utility spaces. The fourth tournament represents non-linear domains, where the issues have interdependencies. These are especially challenging because they represent extremely large contract spaces upto novillions (10^{30}) in size and GENIUS does not expose the used utility function(s) at the time of this writing. The non-linear domains are a recent addition to GENIUS. Unlike their linear counterparts, only a handful of agents are compatible to negotiate in non-linear domains.

The final results are shown in Table 6.13 and Fig. 6.14(d). The eReactiveEx-

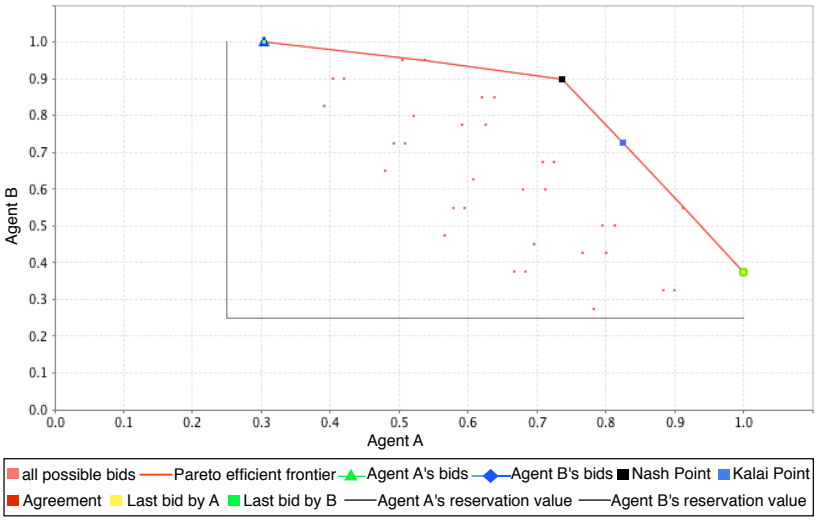


Figure 6.11.: Flight Booking Domain

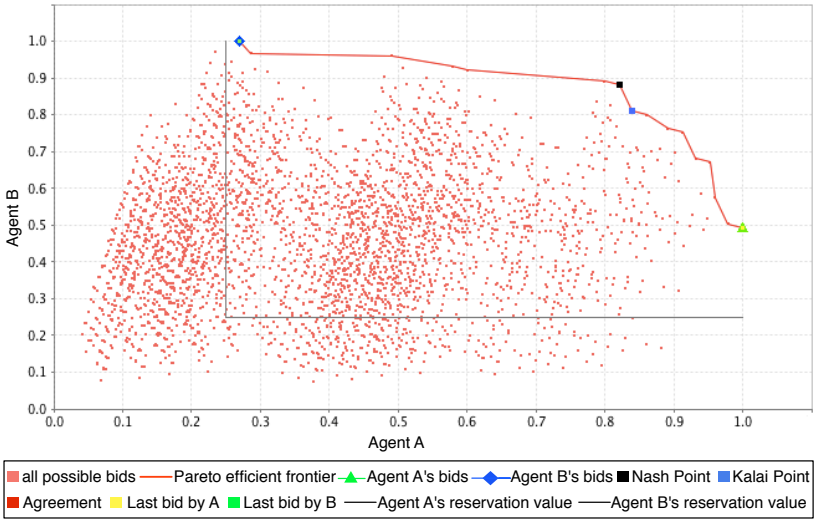


Figure 6.12.: Camera Domain

ploiter ranks best for utility while AgentK2 comes second. For social welfare, a head to head competition is seen with eReactiveExploiter leading by a fine margin while AgentK2 and AgentK respectively score second and third. For social utility, eReactiveExploiter maintains a better lead, while AgentK2 and AgentK land closely at second and third positions respectively. Due to the large size of these domains, the score for Pareto-optimal bids remained zero for all agents.

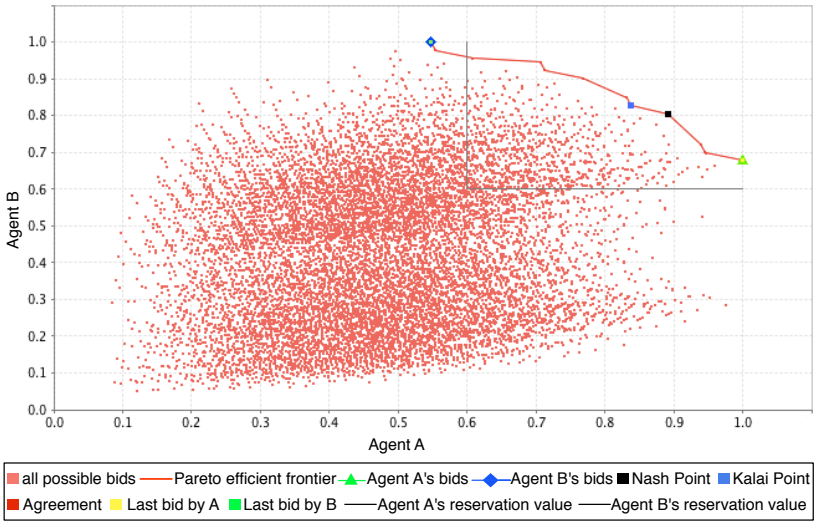


Figure 6.13.: Smart Phone Domain

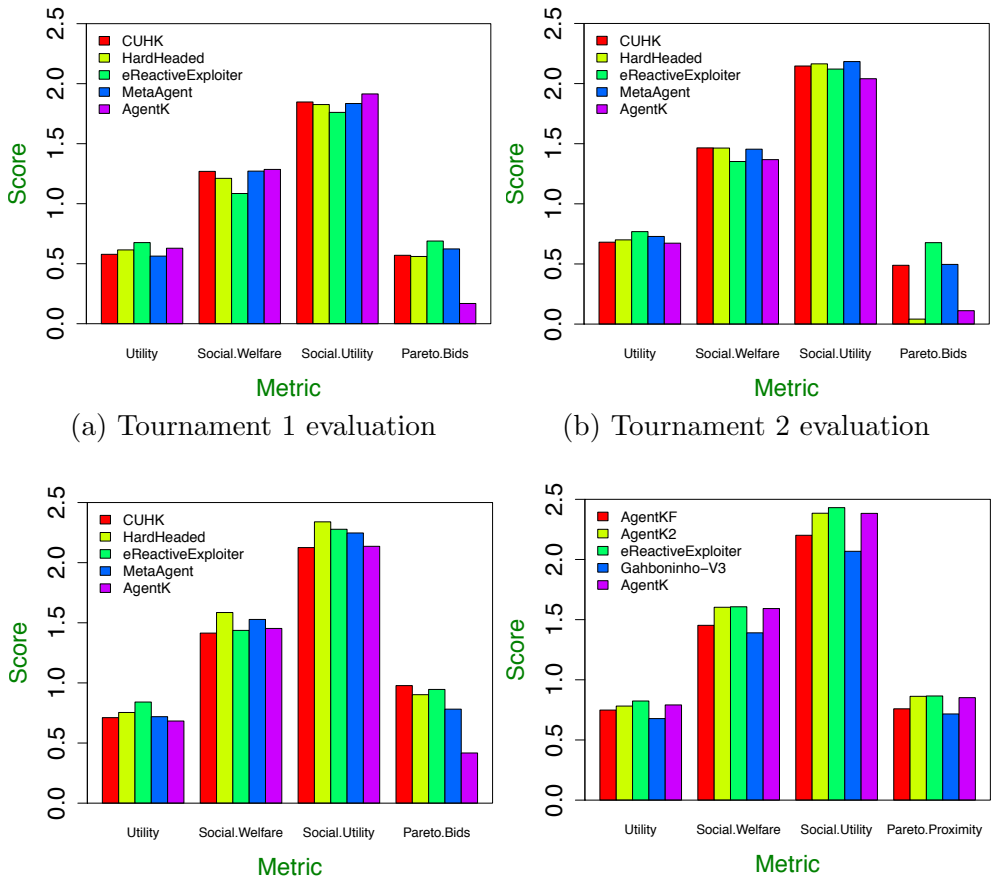
Therefore, the average Euclidean distance of each negotiated bid to a nearest bid on the Pareto-frontier (d_p) was considered. This is shown as Pareto distance in Table 6.13 and plotted as Pareto-proximity score ($1 - d_p$) for improved visibility on the barplot in Fig. 6.14(d). The eReactiveExploiter leads with a very fine margin. AgentK2 ranks second and AgentK ranks third. Hence, eReactiveExploiter can also negotiate SLAs competitively on non-linear domains.

Table 6.13.: Tournament 4 Results

Strategy	Utility	Std.Dev.	Social Welfare	Social Utility	Pareto Distance	Rounds
AgentKF	0.7484	0.0028	1.4527	2.2011	0.2412	950
AgentK2	0.7816	0.0291	1.6028	2.3844	0.1373	10577
eReactiveExploiter	0.8238	0.0386	1.6069	2.4307	0.1345	19740
GahboninhoV3	0.6776	0.0449	1.3904	2.068	0.2839	27873
AgentK	0.7910	0.028	1.5921	2.3831	0.1484	6939

6.8.2. Discussion

The best to worst utility lead in tournament 1 is 16%, which is similar to that for tournament 3 and 4 at 18% and 17% respectively. However, this lead is merely 12% for tournament 2. This confirms that competition in low-conflict markets like fault-tolerant batch services is lower than for mission critical services and even weaker strategies can strike good SLAs. This highlights the business



(a) Tournament 1 evaluation

(b) Tournament 2 evaluation

(c) Tournament 3 evaluation

(d) Tournament 4 evaluation

Figure 6.14.: Performance Results

opportunity which may exist in this market segment. The mean utility obtained by tournament 2 is at 0.71, much higher than 0.61 obtained in tournament 1, where competition is higher. The mean utility obtained by tournament 3 and 4 is 0.74 and 0.76 which are quite high values. This indicates good potential for negotiated purchases in B2C markets, despite medium opposition.

The cooperation and mutual benefit is low for tournament 1, where the mean values for social welfare and social utility stand at 1.22 and 1.83. For tournament 2, these are 1.42 and 2.13. For tournament 3, these are 1.48 and 2.25 and for tournament 4, these are 1.52 and 2.29. Hence, it can be deduced with some cautious optimism that with the exception of mission critical services (tournament 1), mutually beneficial SLAs are possible in all other market segments.

The same agents participated in the first three tournaments. Therefore, it is possible to compare mean performance of each agent in these tournaments. The eReactiveExploiter leads with a mean utility of 0.762, followed by HardHeaded at 0.69. CUHK comes next at 0.657 followed by MetaAgent at 0.67 and AgentK at 0.661. For social welfare, HardHeaded leads with mean value of 1.42, followed by MetaAgent at 1.418, CUHK at 1.383, AgentK at 1.369 and eReactiveExploiter at 1.291. Hence, eReactiveExploiter acted selfishly due to its reactive nature and values of behavior parameters used. The mean social utility for these three tournaments is led by HardHeaded at 2.11, followed by MetaAgent at 2.09, followed by eReactiveExploiter at 2.05, with CUHK at 2.04 and AgentK at 2.03. For Pareto-optimal bids, eReactiveExploiter leads with mean value of 0.77, followed by CUHK at 0.68, MetaAgent at 0.634, HardHeaded at 0.5 and AgentK at 0.232.

Overall, four tournaments were conducted that used seven opponent strategies and seven negotiation domains. Only the first domain was designed by the author while the rest were used from prior art. The cloud computing domain was used with two customer preference profiles, hence in total, negotiation strategies were evaluated for eight outcome/contract spaces.

6.9. Summary

This chapter analyzed the role of utility-optimizing negotiation strategies to establish SLAs of high business worth. Two strategies were proposed, which enable SLA (re)negotiation in an adaptive manner. Two set of holistic experiments were conducted to validate these strategies. These exposed negotiation dynamics in diverse domains and preference conflicts. The first set of experiments provided visibility into a standard cloud computing scenario, which is integrative but conflict stricken. Markets composed of learning, non-learning or mixed negotiation strategies were thoroughly examined. The second set of experiments provided further visibility into service domains where issues have dependencies and negotiator roles carry reserved and discounted utilities. The results showed that proposed strategies perform at par against state of the art opponents.

Future prospects could explore further domains and performance metrics like altruism, fortunate and unfortunate moves. Dissection analysis of strategy space to identify optimal bidding, opponent modeling and acceptance functions is another promising area to reuse existing building blocks for creating new strategies.

Chapter 7.

SLA-aware Resource Management in Cloud Computing

This chapter presents a solution to the third problem described in Chapter 4, i.e., SLA-aware resource management in cloud computing. The contents of this chapter make use of author's publication [48] and definitions available in [115].

This work specifically considers the increasingly popular Platform as a Service (PaaS) cloud service delivery model. PaaS delivers software platforms as a utility by raising the level of abstraction on infrastructure resources. This is achieved by providing an ecosystem to cloud-enable software services and manage their life-cycle through runtime controls. Through advanced automation, PaaS hides the administration complexities of underlying IaaS for the user, thereby accelerating development of applications on the cloud.

Due to its vast potential, the PaaS is gaining market traction [97, 106]. However, much research to date has concentrated on efficient management of virtual machines in the IaaS model [108–110]. The deployment model in PaaS is based on densely allocated operating system (OS) level containers. This improves utilization by securely hosting different modules of multiple applications on a shared OS. PaaS has led to a new paradigm for software development called Microservices, which allows fine grained management of service components as lightweight and portable virtual containers. Unlike VMs, containers can be efficiently scaled, migrated or placed according to SLA dependencies *near* other services or data sources. This can be exploited by added-value SLA propositions.

However, the author's work on PaaS showed that on-demand procurement, unpredictable workloads and auto-scaling result in rapid increase and decrease of containers that are automatically provisioned and cause undesired utilization of the underlying machines. At global scale, this poses a huge challenge e.g., Google

recently revealed that it runs all its services from search to gmail in containers and each week, it launches more than 2 billion containers across its global data centers [107]. RedHat's public PaaS (OpenShift Online [94] - built over Amazon EC2) currently hosts over 2.5 million applications. Hence, the main challenge of a PaaS provider is to regularly plan and optimize the placement of containers on machines, such that SLA commitments are fulfilled and resource utilization is maximized using minimum machines.

On the other hand, service-driven constraints regarding containers and spatial constraints regarding machines render allocations satisfying only resource capacity requirements as infeasible. This relatively novel "Service Consolidation" problem is a variant of variable sized multi-dimensional bin-packing and hence NP-hard [112, 113]. Further, feasible solutions need to be evaluated not just against objective function score but also additional criteria e.g., total machines used, utilization, resource contention, migrations, container scaleups, estimated SLA violations and energy consumption. The latter raises serious economic and ecological concerns with data centers consuming between 1.1%-1.5% of global energy use in 2010 [111]. Well defined metrics for these criteria need to be systematically assessed for optimal trade-offs [110, 116].

Therefore, in this chapter, following concrete contributions are made:

1. Formal models are defined or reused for machine utilization, power consumption and SLA violations.

2. The Service Consolidation problem is concretely framed for a real world cloud stack by leveraging and extending the Machine Reassignment model proposed by Google for the EURO/ROADEF challenge [115].

3. An efficient consolidation solution is provided for SLA conformance by performing simulations on datasets that reflect clouds of various scales, configurations and workloads. Metaheuristic search is applied to discover (re)allocation plans (solutions), which bring the cloud from unconsolidated to consolidated state by considering service and machine level constraints.

4. Performance evaluations of solutions discovered by Tabu Search, Simulated Annealing, Late Acceptance and Late Simulated Annealing algorithms as problem properties change in datasets.

5. A utility function is used to rank solutions against various business policies.

7.1. Related Work

Resource allocation in cloud computing has focused mostly on the IaaS model. [109] presents energy-aware resource allocation using modified Best Fit Decreasing heuristic considering single resource model. The work is extended in [108] using real workloads but remains confined to CPU utilization. Our objective on the other hand is to examine consolidation algorithms under high stress imposed by multiple resources and constraints without which the feasibility of a solution remains questionable. In [110], profit maximizing models are tested using Worst Fit and Best Fit heuristics for VM based provisioning. These works used CloudSim and evaluated VM migrations, energy use and SLA violations.

In contrast, we focus on resource allocation in PaaS. We abstract over multiple resource dimensions, use high-variability datasets from Google where the base state space is exponentially large and characterize these for OpenShift. As OpenShift cloud can span multi-domain IaaS, which makes allocations hard, we solve Service Consolidation under said settings. Other popular PaaS tools are CloudFoundry, Google AppEngine, Docker and more recently project Atomic, GearD and Google Kubernetes. Kubernetes is recently inducted as a scheduler in OpenShift broker to manage large clusters of machines and container groups called pods. Similarly, Docker has been recently adopted as a container standard in OpenShift. Growing interest in these tools to exploit the untapped potential of cloud highlights associated research problems such as service consolidation.

Service consolidation is addressed in [113] using integer linear programming and compared with constraint programming. Like them, our results also confirm that latter always finds a feasible solution in short times. In [114], queuing network theory is used to generate complex data center models and a heuristic is devised to optimally allocate applications on servers. Our work is cloud-based and we use Metaheuristics instead. In [122], Simulated Annealing is shown to improve resource utilization better than First Come First Serve but very small scale datasets are used. [123] solves network-sensitive hosting of application components on multiple cloud data centers and proves the efficiency of Tabu search over mixed-integer linear programming. A high level difference from these works and the proposed solution is that the author has contextualized the service consolidation problem to be SLA aware.

7.2. System Context

The Generic SLA management framework (GSLAM) of the SLA@SOI project was presented in Chapter 5. This can be instantiated to incorporate SLA management on cloud stacks [120]. Fig. 7.1 shows a simplified architecture of OpenShift-based SLAM. The figure highlights major interaction sequences: 1) An SLA is negotiated to agree on QoS values using the Protocol Engine component as introduced in Section 5.4 [45]. It uses the Planning and Optimization (POC) component to process an offer and generate counter offer. 2) When SLA is successfully agreed, the POC triggers provisioning via the Provisioning and Adjustment component. 3) Provisioning invokes the OpenShift broker to allocate resources accordingly. 4) Monitoring Management starts monitoring the resource

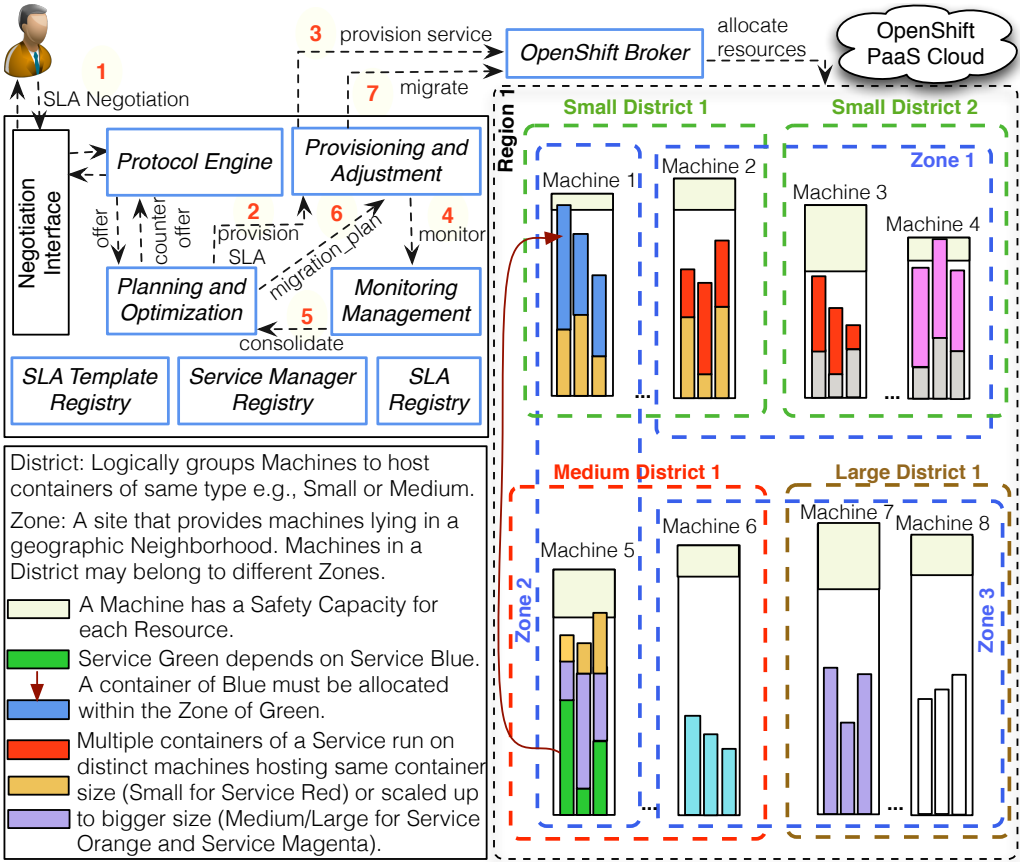


Figure 7.1.: OpenShift SLAM, OpenShift Cloud and Legend (elaborating container placement)

usage against SLA using an OpenTSDB based mechanism [119]. 5) Monitoring also triggers a Service Consolidation request to POC if underutilization is observed over time. 6) POC optimizes the usage model of allocated containers and machines to produce a migration plan. 7) Provisioning executes the migration plan in off-peak times.

OpenShift cloud comprises of broker(s) that receives API calls to control services and cloud machines (see Fig.7.1). Machines are grouped as districts and may belong to different IaaS providers. An IaaS site is tagged as a zone. Multiple zones constitute a region. Thus, OpenShift cloud can span multi-domain infrastructures (zones) in different continents (regions) to increase resilience and provide compliance with regional data protection regulations. Its plugin-in interface can integrate custom algorithms to control container placement. This can be augmented with the POC of our SLAM.

7.3. Problem Definition and Model

For the ROADEF/EURO Challenge 2012, Google presented the Machine Reassignment problem [115], whose remarkable applicability to the PaaS adds to the motivation behind this work. The datasets provided by Google represent realistic challenges and common grounds for further exploitation. Hence, we adhere much to the specification in [115] but consider OpenShift for adjusting some constraints and for dataset characterizations, without which the problem remains largely theoretical. This approach provides widely accepted definitions and abstaining the bias of self generated workloads.

The primary aim of the problem is to improve utilization of a set of machines given a set of services. A service comprises a set of processes which are assigned to machines and consume its resources. We characterize processes as containers because the minimum unit of deployment in PaaS is a container. Containers can be migrated between machines to improve utilization but moves are limited by hard and soft constraints. Since OpenShift PaaS can be developed using physical or virtual machines, both notions of machine apply. Next, we formally describe the problem and the models.

7.3.1. Notations

Consider $\mathcal{M}$ is a set of machines, $\mathcal{V}$ is a set of virtual containers, $\mathcal{R}$ is a set of resources common to all machines. $\mathcal{S}$ is a set of services and a service $s \in \mathcal{S}$ is a set of containers $\subset \mathcal{V}$. Services are disjoint. $M(v) = m$ is an assignment of a container $v \in V$ to a machine $m \in \mathcal{M}$ while $M_o(v)$ represents initial assignment of container v. $C(m, r)$ is the total capacity of resource $r \in \mathcal{R}$ for machine m and $R(v, r)$ is the requirement of resource r by container v. $SC(m, r)$ is the safety capacity of resource r for machine m.

Let $\mathcal{T} = \{small, medium, large\}$ be a set of supported container types such that $small < medium < large$ along all resource dimensions. Let $\mathcal{D}$ be a set of districts where a district $d \in \mathcal{D}$ is a set of machines. Districts are disjoint sets. $d(m)$ identifies the district of machine m, $t(v) \in \mathcal{T}$ gives the type of container v and $t(d(m))$ gives the (subsuming) container type of the district to which machine m belongs e.g., a district of type $large$ can host $large$, $medium$ and $small$ containers, a $medium$ district can host $medium$ and $small$ containers, but a $small$ district only hosts $small$ containers. Let $\mathcal{Z}$ be a set of disjoint zones where a zone $z \in \mathcal{Z}$ is a set of machines lying in a spatial neighborhood. A zone represents an IaaS availability zone. In this work, zone is equated to the neighborhood (of Machine Reassignment problem).

7.3.2. Hard Constraints

Hard constraints represent rigid placement requirements. A feasible solution is the one which satisfies all hard constraints.

Definition 1 : Capacity Constraint

For an assignment M, the usage U of a machine m for a resource r is defined as:

$$U(m, r) = \sum_{v \in \mathcal{V}, M(v)=m} R(v, r)$$

A container can run on a machine only if the machine has enough capacity available on every resource. A feasible assignment satisfies the capacity constraint:

$$\forall m \in \mathcal{M}, r \in \mathcal{R}, U(m, r) \leq C(m, r)$$

Definition 2 : Type Constraint

A container is placed on a machine whose district supports the container's type:

$$\forall v \in \mathcal{V}, M(v) = m, t(v) \leq t(d(m))$$

This allows vertical scaleup but not down scaling because the latter could violate the resource capacity guaranteed in the SLA.

Definition 3 : Conflict Constraint

Containers of a service $s \in \mathcal{S}$ must run on distinct machines.

$$\forall s \in \mathcal{S}, (v_i, v_j) \in s^2, v_i \neq v_j \Rightarrow M(v_i) \neq M(v_j)$$

Definition 4 : Transient Usage Constraint

When a container is migrated from machine m to a machine m', some resources e.g., disk or memory are consumed twice. Hence, machine m and m' must have enough capacity for the migration. Let $\mathcal{TR} \subseteq \mathcal{R}$ be resources needing transient usage, then the transient usage constraints are:

$$\forall m \in \mathcal{M}, r \in \mathcal{TR}, \sum_{\substack{v \in \mathcal{V}, such that \\ M_o(v)=m \vee M(v)=m}} R(v, r) \leq C(m, r)$$

We ignore the Spread constraint as it conflicts with our Type constraint.

7.3.3. Soft Constraints

Soft constraints model costs associated with resource contention, collocation requirements or migrations. Costs are defined in original datasets[1]. The quality of a feasible solution is determined by how much it satisfies the soft constraints.

Definition 5 : Load Cost

Let $SC(m, r)$ be the safety capacity of a resource $r \in \mathcal{R}$ on a machine $m \in \mathcal{M}$. The load cost is defined per resource and refers to the capacity used above SC.

$$loadCost(r) = \sum_{m \in \mathcal{M}} max(0, U(m, r) - SC(m, r))$$

Definition 6 : Dependency Cost

As argued earlier in this work, services may have inter-dependencies. Since dependencies may affect QoS of a service e.g., availability or performance, resource allocation should reflect dependency among two services. Concretely, a service s^a depending on service s^b requires each s^a container to run in the zone of atleast one container of s^b. Non satisfaction increments a *dependencyCost* variable.

[1]Exception is Dependency Cost, whose definition is modified and computed as described.

$$\forall v^a \in s^a, \exists v^b \in s^b \text{ s.t. } M(v^a) \in z1 \text{ and } M(v^b) \in z2 \text{ and } z1 = z2$$
$$\text{if } (z1 \neq z2) \mapsto dependencyCost++$$

On a technical note, the OpenShift API allows to migrate a service to another zone (or region). After migration, its software based networking commands allow to setup communication mechanism between containers.

Definition 7 : Balance Cost

Availability of one resource could be useless without sufficient availability of another. When migrating containers, an availability ratio between two resources is targeted for future assignments. This is modeled by a set $\mathcal{B}$ of balance triples defined on $\mathcal{N}$ X $\mathcal{R}^2$. For a triple $b =< r1, r2, target >\in \mathcal{B}$, the *balanceCost* is:

$$balanceCost(b) = \sum_{m \in \mathcal{M}} max(0, target * A(m, r1) - A(m, r2)) \text{ where}$$
$$A(m, r) = C(m, r) - U(m, r)$$

Definition 8 : Container Move Cost

This cost models the difficulty some containers pose for migrating e.g., due to dependencies, large size, lack of replicas/clones or administrative reasons. Let $CMC(v)$ be the cost of migrating a container v, then the container move cost is computed as a sum of all migrations:

$$containerMoveCost = \sum_{\substack{v \in \mathcal{V} such that \\ M(v) \neq M_o(v)}} CMC(v)$$

Definition 9 : Service Move Cost

Service move cost defines maximum number of migrated containers. This balances migrations among services.

$$serviceMoveCost = \max_{s \in \mathcal{S}}(\mid \{v \in \mathcal{S} \mid M(v) \neq M_o(v)\} \mid)$$

Definition 10 : Machine Move Cost

Let $MMC(m_{source}, m_{destination})$ be the machine level cost of migrating a container v from m_{source} to $m_{destination}$. This cost is specified for certain machine combinations and models the complexity due to spatial distance, network, machine or other technical reasons. The machine move cost is computed as a sum of all applicable MMC:

$$machineMoveCost = \sum_{\substack{v \in \mathcal{V} such that \\ M(v) \neq M_o(v)}} MMC(M_o(v), M(v))$$

Definition 11 : Objective Function

By combining costs from all soft constraints, an objective function is defined as a total of all costs, using weights as available in the original datasets:

$$\begin{gathered} totalCost = \sum_{r \in \mathcal{R}} weight_{loadCost}(r) * loadCost(r) + \\ \sum_{b \in \mathcal{B}} weight_{balancedCost}(b) * balanceCost(b) + \\ weight_{containerMoveCost} * containerMoveCost + \\ weight_{serviceMoveCost} * serviceMoveCost + \\ weight_{machineMoveCost} * machineMoveCost + \\ dependencyCost \end{gathered}$$

This value is to be minimized by searching for feasible assignments that better satisfy soft constraints e.g., increase utilization of machines, while also balancing resource contention among used machines. Conversely, constraint dissatisfaction is measured as a negative value (called score), which is maximized as improved solutions are discovered.

7.3.4. OpenShift Characterization

To contextualize the ROADEF datasets with OpenShift (version 2.0) model, each process $p \in \mathcal{P}$ given in the datasets was characterized as a container $v \in \mathcal{V}$ of type $t \in \mathcal{T}$. The classification method is shown in Fig.7.2. minR$(\mathcal{P}, r)$ and maxR$(\mathcal{P}, r)$ give the minimum and maximum requirements of resource r among all processes, while $R(p, r)$ gives the requirement of resource r by process p. Lines 7-15 record a process's usage of each resource by incrementing small, medium and large (s, m, l) flags, after comparing with the global usage of that resource. A process using any resource in large proportion is classified as a *large* container, else *medium* and finally as a *small* container (lines 16-22).

Next, each group of machines called location $l \in \mathcal{L}$ was characterized as OpenShift district $d \in \mathcal{D}$ which hosts containers of subsuming types. The classification method is shown in Fig.7.3. Here, dominanceCount shows that a location dominates another over a resource and is measured for all resources of all machines belonging to each location (lines 14-24). Using this criteria, we characterize most dominating locations as districts of type *large* and in descending order of type

class CharacterizeProcessAsContainer
method Classify($\mathcal{P}, \mathcal{R}$)
1: **for** each resource $r \in \mathcal{R}$ **do**
2: $\min_r \leftarrow \text{minR}(\mathcal{P}, r)$; $\max_r \leftarrow \text{maxR}(\mathcal{P}, r)$
3: $\text{portion}_r \leftarrow (\max_r - \min_r) / 3$
4: **end for**
5: **for** each $p \in \mathcal{P}$ **do**
6: $\text{usage}_p = (s, m, l) \in Z^3$
7: **for** each resource $r \in \mathcal{R}$ **do**
8: **if** $R(p, r) \leq (\min_r + \text{portion}_r)$ **then**
9: $\text{usage}_p.s++$
10: **else if** $R(p, r) \leq (\min_r + 2 * \text{portion}_r)$ **then**
11: $\text{usage}_p.m++$
12: **else if** $R(p, r) \leq \max_r$ **then**
13: $\text{usage}_p.l++$
14: **end if**
15: **end for**
16: **if** $\text{usage}_p.s \geq 1$ **and** $\text{usage}_p.m = 0$ **and** $\text{usage}_p.l = 0$ **then**
17: p.containerType $\leftarrow$ *small*
18: **else if** $\text{usage}_p.m \geq 1$ **and** $\text{usage}_p.l = 0$ **then**
19: p.containerType $\leftarrow$ *medium*
20: **else if** $\text{usage}_p.l \geq 1$ **then**
21: p.containerType $\leftarrow$ *large*
22: **end if**
23: **end for**

Figure 7.2.: Container Characterizer

medium and *small* (lines 27-35). Characterization is proportional to the amount of *large*, *medium* and *small* containers in the dataset (lines 1-3).

7.3.5. Utilization and Power Model

Improving machine utilization remains a challenge and an advantage for cloud providers. Reports show that data centers commonly utilize only 10-20% of their server resources [124]. Utilization can be raised by consolidating workloads, which reduces the peak-to-average utilization ratio by executing more load using fewer machines. Since most machines consume upto 70% of their peak utilization energy even when idle, consolidation can reduce energy and the associated cooling costs by turning idle machines into low-power mode or turning them off.

Power usage has been modeled on utilization of one or two resources, especially the CPU [109, 116]. However, recent advances in processor technologies have resulted in energy-efficient CPUs while memory, disk and network components

```
class CharacterizeLocationAsDistrict
    Let v_s ⊂ V : t(v_s) = small and v_m ⊂ V : t(v_m) = medium and v_l ⊂ V : t(v_l) =
    large
method Classify(L, M, R, V)
 1: smallDistrictCount ← min(1, ⌊(| L | / | V |)* | v_s |⌉)
 2: mediumDistrictCount ← min(1, ⌊(| L | / | V |)* | v_m |⌉)
 3: largeDistrictCount ← min(1, ⌊(| L | / | V |)* | v_l |⌉)
 4: Map<resource, capacity> resources
 5: Map<location, resources> locationCapacity
 6: for each location l ∈ L do
 7:    for each machine m ∈ M : l(m) = l do
 8:       for each resource r ∈ R do
 9:          resources[r].addOrUpdate(getResourceCapacity(m, r))
10:       end for
11:    end for
12:    locationCapacity[l] ← resources; resources.reset()
13: end for
14: for each location l_out ∈ L do
15:    for each location l_in ∈ L : l_in ≠ l_out do
16:       for each resource r ∈ R do
17:          r_out ← locationCapacity[l_out].resources[r].capacity
18:          r_in ← locationCapacity[l_in].resources[r].capacity
19:          if r_out > r_in then
20:             l_out.dominanceCount++
21:          end if
22:       end for
23:    end for
24: end for
25: locationsList ← locations.sortOnDominanceCountDescending()
26: largeCounter=0, mediumCounter=0, smallCounter=0
27: for each location l ∈ locationsList do
28:    if largeCounter < largeDistrictCount then
29:       largeCounter++; l.districtType ← large
30:    else if mediumCounter < mediumDistrictCount then
31:       mediumCounter++; l.districtType ← medium
32:    else if smallCounter < smallDistrictCount then
33:       smallCounter++; l.districtType ← small
34:    end if
35: end for
```

Figure 7.3.: District Characterizer

are rising contributors of total power consumption. In [117], Barroso and Hölzle show that CPU contribution to power consumption of Google servers in 2007 was less than 30%. Since our datasets resemble point-in-time system snapshots rather timeseries data, we aggregate utilization of all resources to analytically

model utilization u of a machine m:

$$u = (\sum_{r \in \mathcal{R}} U(m,r)/C(m,r))/ \mid \mathcal{R} \mid \tag{7.1}$$

where U is the usage and C is the total capacity of resource r for machine m. We use u with the power model from [109].

$$P(u) = k * P_{max} + (1-k) * P_{max} * u \tag{7.2}$$

where k is the fraction of power used by a machine in idle state and P_{max} is the maximum power consumed by a fully utilized machine. We use $k = 0.7$ and P_{max}=250 Watt which modern machines generally consume and are also used in [109]. Given mean utilization u_{mean} of all used machines, energy-efficiency e of a cloud can be loosely measured as:

$$e = u_{mean}/P(u_{mean}) * 100 \tag{7.3}$$

7.3.6. SLA Violation Model

Improving utilization to save power costs risks over provisioning, which negatively effects performance. This power-performance traded-off [116] translates into controlling SLA violations in order to maintain profits and good reputation. Mindful of prior art [108], we model SLA violations (SLAV) as upper bound estimate(s) in terms of i) performance degradation due to migration (PDM) and ii) performance degradation due to contention on machine's resources (PDC). PDM is the time when containers are underperforming or unavailable due to migrations. To analyze upper bound, we assume that migrations are performed serially. Migrating in parallel surely reduces this value. PDM is defined as:

$$PDM = t_s * \mid v_s \mid + t_m * \mid v_m \mid + t_l * \mid v_l \mid \tag{7.4}$$

where t_s, t_m and t_l are maximum times needed to migrate a *small*, *medium* or *large* container while v_s, v_m and $v_l \subset \mathcal{V}$ are candidate sets of *small*, *medium* and *large* containers to be migrated. In our experiments, we use $t_s = 10$ seconds, $t_m = 20$ seconds and $t_l = 40$ seconds.

The amount of resource r used above its safety capacity is saved as $load(r)$. PDC is measured as risk tertiles, where $tertile1$ represents low contention, $tertile2$

medium contention and *tertile*3 high contention. An SLA violation only occurs when resource use exceeds 100% of total capacity but low contention (*loadCost*) is desired by balancing load among all used machines. The tertiles allow to estimate SLA violations risk if contention requirements can be relaxed by extending safety capacity to *tertile*1, 2 or 3. This increases utilization at the cost of some extra power but can reduce mean time to failure (MTTF) of a machine. PDC is defined as:

$$Cont_{r \in \mathcal{R}} = C(m, r) - SC(m, r) \tag{7.5}$$

$$PDC = \begin{cases} \text{if} \\ load(r) \leq 1/3(Cont_r) \mapsto tertile1++ \\ \text{else if} \\ load(r) \leq 2/3(Cont_r) \mapsto tertile2++ \\ \text{else if} \\ load(r) > 2/3(Cont_r) \mapsto tertile3++ \end{cases} \tag{7.6}$$

For a dataset, PDC tertiles are incremented for all machines. Finally, SLAV is defined as a product of PDM and PDC:

$$SLAV = PDM * max(1, PDC) \tag{7.7}$$

Here, PDC can be: i) sum of all tertiles, ii) sum of tertile 2 and 3 and iii) only tertile 3. This gives upto three possible SLAV values for the solved datasets.

7.4. Experimental Evaluations

Workflow

The steps that compose the workflow of experiments are shown in Fig. 7.4. First, the problem and datasets are contextualized for OpenShift PaaS cloud model as shown in Section 7.3.4. Next, the problem constraints and score calculation is implemented in a solver and its algorithms are configured (Section 7.4.2).

Next, controlled search had to be run to discover initial feasible solution. This indicated redundancy among bigger containers as the problem does not allow down scaling while scaleup is permitted. Therefore, randomly selected (mostly) *large* containers which are not part of any dependency relationship were reduced. This helped introduce some slack because OpenShift based refinements compli-

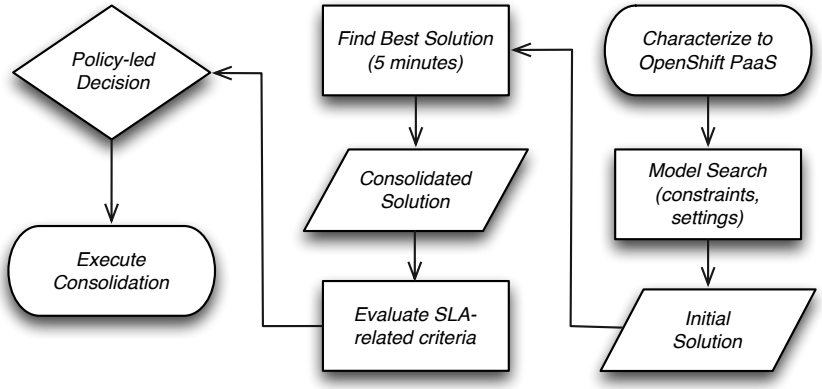


Figure 7.4.: Experiment Workflow

cate the search more than the base problem [115]. The base definition neither classify processes based on their resource requirements e.g., as *small*, *medium* or *large*, nor does it restrict machines to host only certain size of containers. Thus, the decision to consider a real cloud stack to frame and solve the service consolidation problem makes bin packing very hard.

The next step executes search where each algorithm is run for 5 minutes. This produces a consolidated (reallocation) solution by each algorithm. Now, additional evaluation criteria such as utilization, resource contention, migrations, SLA violations, machines used and energy consumption is measured from proposed solutions. This leads to the application of policy-led ranking of each solution and as a result, the most preferred solution is identified and executed. Execution may migrate and/or scaled up certain containers through OpenShift API.

7.4.1. Datasets

We used simulations to evaluate algorithms over given datasets in a reproducible manner. Table 7.1 details the unconsolidated datasets. We retain dataset names from [115] for recognition. A dataset represents a cloud configuration and workload. Datasets A.2.2 and A.2.5 represent clouds of small scale with 100 and 50 machines hosting 170 and 153 services respectively. Dataset B.1 represents a medium scale cloud with 100 machines running 15 times more services than smaller datasets. Dataset B.4 represents a large scale cloud of 500 machines running lesser services than B.1 but very larger number of containers.

The state space represents a theoretical upper bound computed by considering if all *small* containers could be placed on any machine belonging to *small* districts, *medium* containers on any machine belonging to *medium* districts, and *large* containers on any machine belonging to *large* districts. Since containers can be scaled up, the total space of possible combinations is even higher. The stated numbers reflect a reduced subset of the total space, called the base state space. The intent here is to highlight the high number of possible combinations as a relative measure to compare datasets. The task of search algorithms is to find a feasible subset of total state space, which is much smaller but cannot be practically established due to the combinatorial explosion.

Datasets A2.2, A.2.5 and B.1 have 12 while B.4 has 6 resources, which makes the search very hard. Overall, *small* containers make 60-70%, *medium* 25-32% and *large* 3-8% of total population, which also shows that *small* districts may grow faster.

7.4.2. Algorithms

Machine heterogeneity and different resource usages provide the desired variability to resemble real world dynamics. The large state spaces, high dimensionality due to multiple resources and excessive constraints all provide strong motivation to use Metaheuristic search algorithms for solving. Metaheuristics also avoid the trap of getting stuck in local optima. The open-source OptaPlanner [118] solver implements various Metaheuristics and was used in this work. A random mix of change and swap moves were used for all algorithms. Tabu Search (TS) was used with a tabu list of size 7, where one element represents a feasible assignment for a single container. TS produces numerous feasible moves, a subset of which is evaluated in each step. For TS, this evaluation size was set to 2000. Simulated Annealing (SA) requires a starting temperature value to factor score difference. This was set to 0 for hard and 400 for soft constraints. The algorithm allows some non improving moves in the start but gets elitist depending on time-gradient. For SA, the evaluation size was set to 5. Like SA, Late Acceptance (LA) makes fewer moves and improves score over certain late steps. This late size was set to 2000 and evaluation size to 500 for LA. Late Simulated Annealing (LSA) hybrids SA with LA, allowing improvement but also controlled random decrement in score. For LSA, late size was set to 100 and evaluation size to 5.

Table 7.1.: Cloud Dataset Details

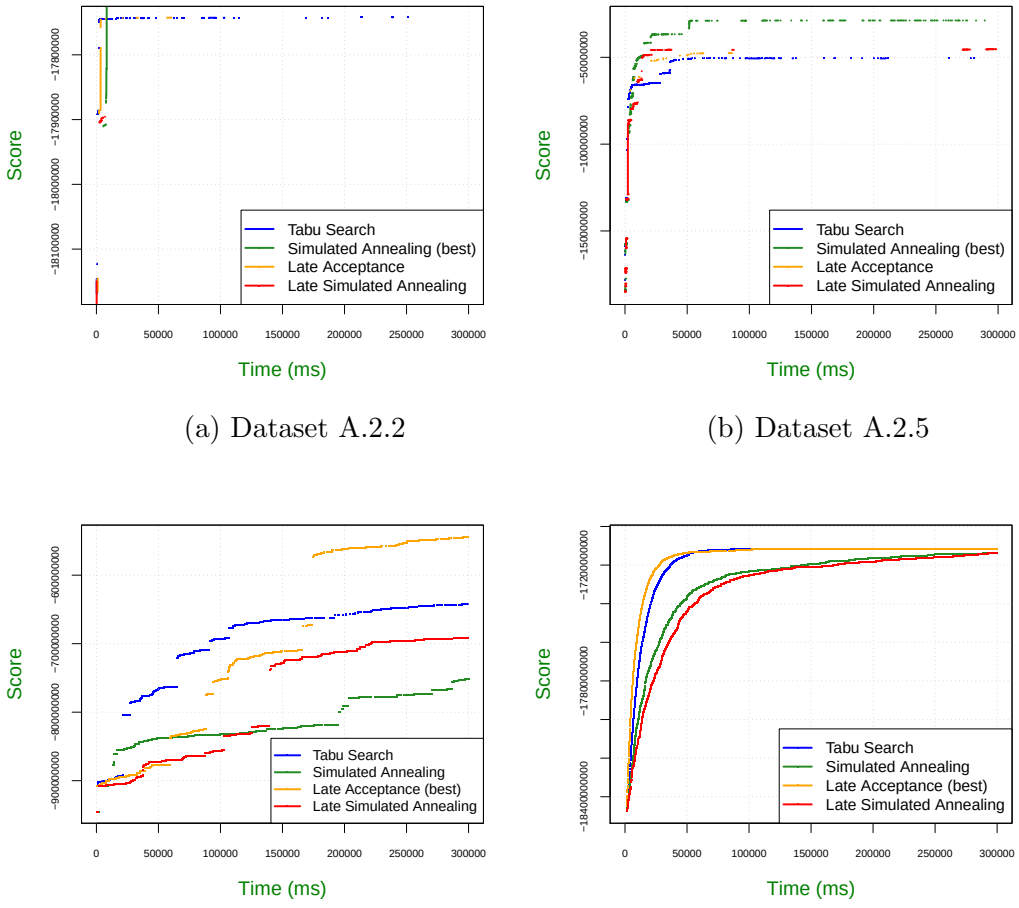
Name	Resources	Districts S, M, L	Zones	Machines S, M, L	Services	Containers S, M, L	State Space	Utilization/ Machine	Consumed Power(kWh)	Energy Efficiency	Initial Solution Score
A.2.2	12	15, 8, 2	5	60, 32, 8	170	434, 222, 48	10^{1148}	35%	19.320	45.25%	-18312546
A.2.5	12	15, 8, 2	5	30, 16, 4	153	455, 245, 63	10^{1004}	37%	10.151	45.5%	-185103629
B.1	12	6, 3, 1	5	60, 30, 10	2512	2572, 1272, 147	10^{6598}	48%	21.122	56.75%	-945528368
B.4	6	35, 13, 2	5	350, 130, 20	1732	10964, 3955, 543	10^{36959}	37%	101.199	45%	-18475614730

Table 7.2.: Consolidation using Tabu Search (TS), Simulated Annealing (SA), Late Acceptance (LA) and Late Simulated Annealing (LSA)

Dataset (Algorithm)	Solved Solution Score	Machines Used S+M+L	Utilization/ Machine	Consumed Power(kWh)	PDC Tertiles (1, 2, 3)	Reduced Load on Resources	Energy Efficiency	Migrations	PDM (min)	Container Scaleup
A.2.2 (TS)	-17742166	95	35%	19.141	(1, 1, 1)	50%	43.43%	206	66	85%
A.2.2 (SA)	-15012210	95	35%	19.126	(0, 0, 1)	83%	43.46%	220	69	80%
A.2.2 (LA)	-17741965	95	35%	19.144	(1, 1, 1)	50%	43.42%	205	66	85%
A.2.2 (LSA)	-15023979	95	35%	19.092	(0, 0, 1)	83%	43.54%	369	95	40%
A.2.5 (TS)	-50622223	47	40%	9.618	(17, 4, 4)	58%	48.87%	345	119	34%
A.2.5 (SA)	-28774923	46	41%	9.454	(3, 6, 8)	71%	49.87%	351	119	30%
A.2.5 (LA)	-47511660	47	40%	9.621	(22, 3, 4)	51%	48.85%	352	120	34%
A.2.5 (LSA)	-45311736	47	40%	9.619	(20, 2, 4)	56%	48.86%	461	136	21%
B.1 (TS)	-642113741	99	49%	20.951	(77, 10, 6)	12%	57.89%	2237	531	9%
B.1 (SA)	-751389259	98	49%	20.766	(64, 14, 4)	23%	57.81%	2362	551	10.5%
B.1 (LA)	-544596716	99	49%	20.948	(72, 14, 4)	15%	57.89%	2237	532	9%
B.1 (LSA)	-691537717	99	49%	20.947	(73, 15, 6)	11%	57.9%	2935	647	7%
B.4 (TS)	-17116530373	500	36%	101.122	(0, 0, 0)	100%	44.5%	14883	3384	17%
B.4 (SA)	-17136833433	500	36%	101.173	(18, 4, 2)	93%	44.48%	15176	3433	14%
B.4 (LA)	-17116172349	500	36%	101.153	(0, 0, 0)	100%	44.49%	14598	3333	17%
B.4 (LSA)	-17136527221	500	37%	101.21	(17, 3, 4)	93%	45.7%	15207	3437	14%

7.4.3. Performance Results and Discussion

The results of consolidated solutions are presented in Table 7.2. Figs. 7.5(a-d) reveal the search pattern for score improvement. This helps to determine how efficiently a particular algorithm finds its best solution.



(a) Dataset A.2.2

(b) Dataset A.2.5

(c) Dataset B.1

(d) Dataset B.4

Figure 7.5.: Score Improvement Pattern of Algorithms

In (a), SA finds the best score in less than 15 seconds while LSA trails closely behind. In (b), SA outshines after 20 seconds and LSA lands second but with a larger gap. Note that for SA, the curve flattens out after only 50 seconds which indicates that search is exhausted and most likely the optimal solution

has been discovered. Score improvement for other algorithms remains sparse. In (c), the score improvement pattern is quite sporadic for LA and LSA, but less sporadic for TS and SA. LA takes a wide lead after 175 seconds and is trailed by TS. However, there is no flattening of the curve for any algorithm, which means that the score would improve further in this dataset if searched for more time. An interesting pattern is seen in (d), where all curves eventually flatten out. However, this flattening is quite rapid for LA and TS . LA takes lead after 100 seconds, after which there is little competition but TS follows closely.

The solved solution score shows that SA is best suited for small state spaces (A2.2 and A2.5), LA for medium state spaces (B.1), while for large state spaces (B.4), LA is slightly better than TS. Fig. 7.6 summarizes score improvement over initial solution after consolidation. Averaging these results over all datasets places SA as the best score improving algorithm with 32.65%, LSA is second with 32.04%, LA is third with 31.79% and TS is fourth with 28.89%.

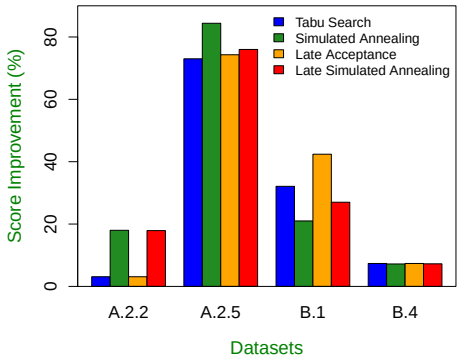
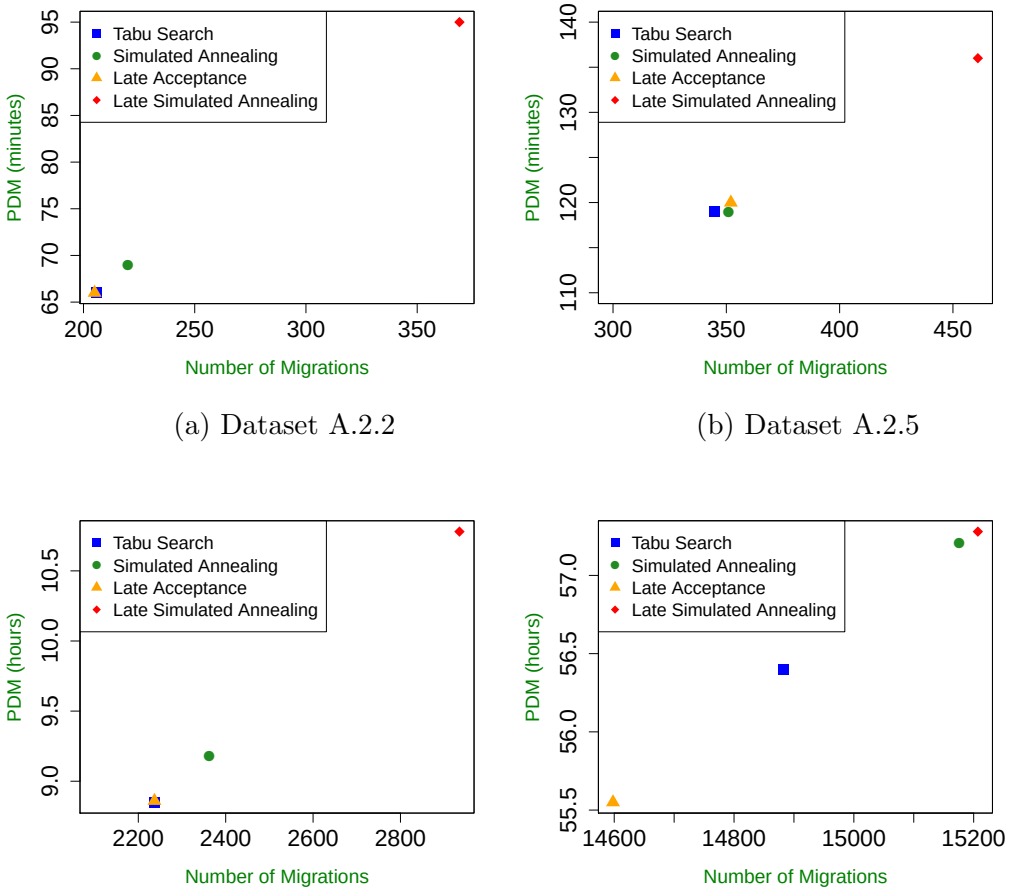


Figure 7.6.: Score improvement over initial solution

Figs. 7.7(a-d) show how many container migrations (and corresponding PDM) are proposed by the best solution of each algorithm. In plot (a), TS and LA have a slight advantage over SA whose solution score was the best. In (b), TS again proposes least migrations but the margin with SA is very low. Considering its high lead in solution score, SA is a clear winner. In (c), LA and TS propose the same number of migrations and LA can be preferred due to its wide margin on TS in terms of solution score. In (d), LA proposes least migrations and distinguishes itself over TS. The LSA is a clear outlier and proposes highest migrations. If low

migrations is a strict preference, LSA is not a good choice.



(a) Dataset A.2.2

(b) Dataset A.2.5

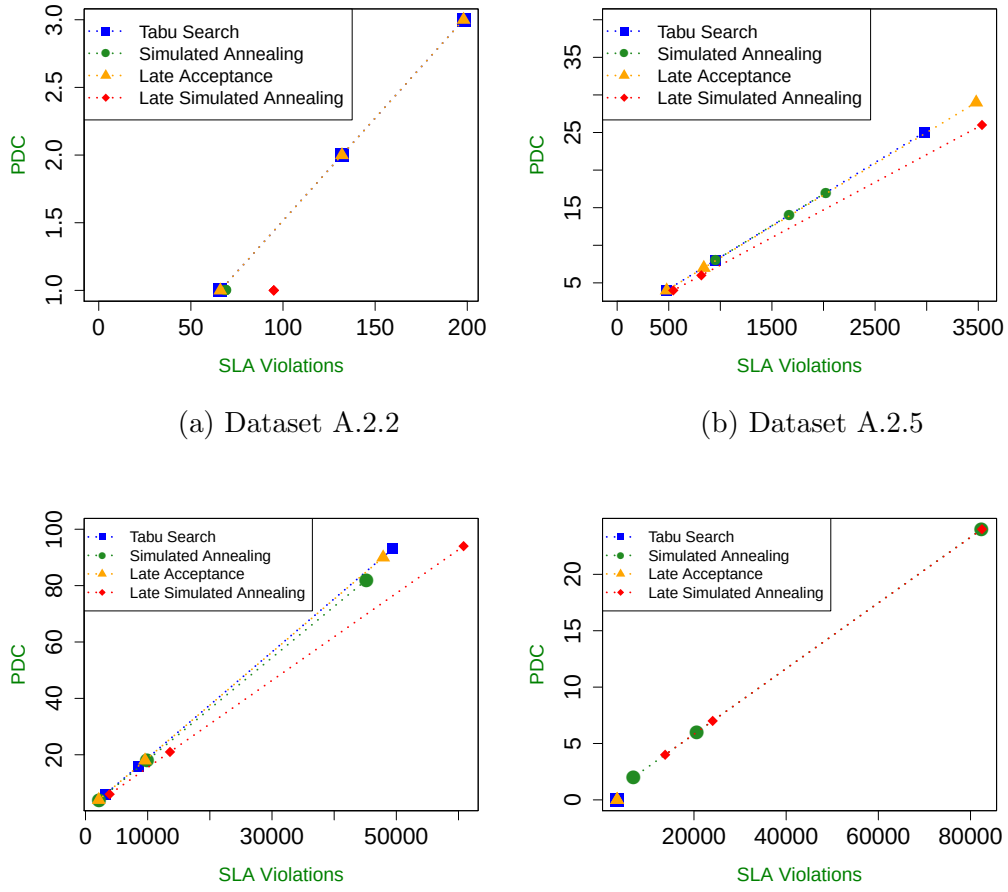
(c) Dataset B.1

(d) Dataset B.4

Figure 7.7.: Number of Migrations and PDM

Figs. 7.8(a-d) show SLA violations (SLAV) with relaxing PDC tertile values (see Table 7.2). Plot (a) shows three clear regions against the three PDC values. Using least contention (PDC=sum of all tertiles) as the most strict evaluation criteria, in (a) SA and LSA cut even but SA proposes low SLAV due to its low PDM. In (b), SA wins over TS with low SLAV as PDM is the same for both. In (c), SA again creates low contention and hence low SLAV while LA comes second. In (d), LA and TS amazingly eliminate all contention on 500 machines.

LA gives least SLAV due to lower PDM than TS.



(a) Dataset A.2.2

(b) Dataset A.2.5

(c) Dataset B.1

(d) Dataset B.4

Figure 7.8.: Drop in SLA Violations with relaxing PDC as: i) sum of all Tertiles (north-east region), ii) sum of Tertile 2 and 3 (mid-region), iii) Tertile 3 (south-west region)

Using the same evaluation criteria, Fig. 7.9 shows how consolidation significantly reduced SLA violations in comparison to the unconsolidated system state. The only exception here is the LSA which increased the SLAV by 0.02% in dataset B.1 for the used PDC criteria. It however does perform well for relaxed criteria. Taking mean value of Fig. 7.9's results over all datasets, each algorithm can be evaluated for overall reduction of SLA violations. SA takes the lead by reducing

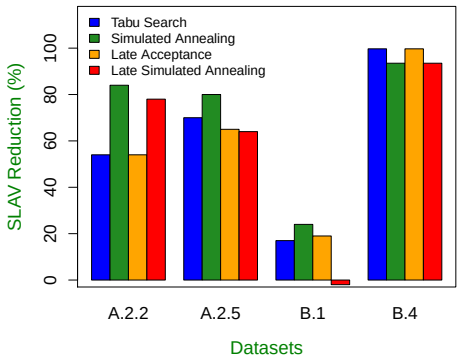


Figure 7.9.: Reduction in SLA Violations over initial solution

SLA violations upto 70.38%, TS is second with 60.18%, LA is third with 59.43% and LSA is last with 58.38%.

Overall, consolidation improved mean utilization of machines while using fewer machines and lesser energy in most cases. In A2.2, load on resources is reduced upto 83%, total energy consumption is reduced by 1%, utilization remains the same but 5% less machines are used. In A2.5, load on resources is reduced upto 71%, energy is reduced by 7%, utilization improves upto 3% while 8% less machines are used. In B.1, load on resources is reduced upto 23%, energy is reduced by 2%, utilization improves by 1% while 2% less machines are used. In B.4, load on resources is reduced upto 100%, energy saving and utilization show little improvement and same number of machines are used, hinting that longer search could be tried for larger scale. The energy consumption is considered to be mean value for hourly consumption. Note that energy savings of 1-7% observed here multiply into reasonable monthly savings as shown in Fig. 7.10. The mean value over all datasets positions SA as our most environment friendly choice with monthly savings of 229.14kWh, TS is second with 172.8kWh, LA is third with 166.68kWh and LSA is last with 166.32kWh.

These results confirm that major gains from service consolidation lie in reduced energy costs and reduced SLA violations. This is achieved through a balanced redistribution of workload on machines by performing migrations such that the resultant *loadCost* is substantially reduced. Table 7.2 also shows energy-efficiency and container scaleup values. The energy-efficiency must be considered in ref-

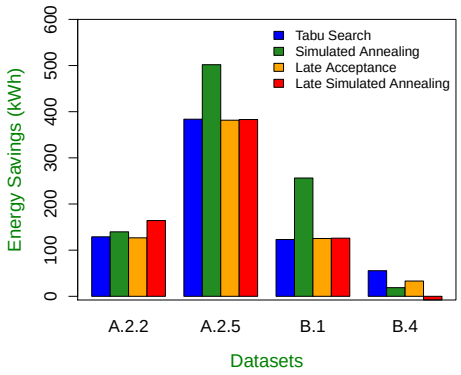


Figure 7.10.: Energy savings per month

erence to the used machines. Hence, values have been normalized to maximum energy-efficiency possible with the used machines.

7.4.4. Policy-led Ranking of Solutions

The presented analysis unfolds the decision making challenge faced by cloud service providers. The author is of the opinion that the presented metrics must be harnessed and traded-off according to a high level policy. To this goal, solutions are ranked by aggregating the weighted utility of individual metrics, where the weights reflect business preferences of the policy in use. This is achieved by the following utility function:

$$U(sol) = \sum_{i=1}^{N} w_i u_i(x_i) \tag{7.8}$$

Here, U is the utility of a solution sol, $u_i(x_i)$ gives the utility of metric x_i normalized as a real value in range $[0, 1]$, w_i is the weight assigned to u_i and $\sum_{i=1}^{N} w_i = 1$. This policy-led, utility-oriented scheme allows to obscure the complexity of individual metrics and decide for the most preferred solution.

Nine metrics were normalized to derive individual utility. For solution score, the difference between initial and solved score was normalized over the maximum difference. The same was applied to machines used and energy consumed before and after consolidation. The values for mean utilization and load reduction were

used as such since they are already normalized. Some metrics represent negative properties. These include the sum of PDC tertiles, number of migrations, PDM and container scaleups. The differences in their values were first normalized and then subtracted from 1 to obtain their positive utility. Energy-efficiency was not considered since its interpretation is subjective to side effects. Next, five business policies are presented to rank solutions and hence, algorithms on each dataset. **High Score Policy:** This policy prefers a high solution score and assigns it a weight of 0.5 while remaining metrics are weighted 0.0625 each. Results are shown in Table 7.3.

Table 7.3.: Algorithms Ranked on High Score Policy

	Datasets			
Rank	**A.2.2**	**A.2.5**	**B.1**	**B.4**
1	SA	SA	LA	TS
2	LSA	TS	TS	LA
3	TS	LSA	SA	SA
4	LA	LA	LSA	LSA

Low Migration Policy: This policy favors low migrations and weighs the PDM and the number of migrations as 0.25 each, while the remaining metrics are weighted equally with 0.071428571. Results are shown in Table 7.4.

Table 7.4.: Algorithms Ranked on Low Migration Policy

	Datasets			
Rank	**A.2.2**	**A.2.5**	**B.1**	**B.4**
1	SA	SA	SA	TS
2	TS	TS	LA	LA
3	LA	LA	TS	SA
4	LSA	LSA	LSA	LSA

Low Contention Policy: This policy favors low resource contention and weighs the PDC and load reduced on resources as 0.25 each, while the other metrics are equally weighted with 0.071428571. Results are shown in Table 7.5.

Table 7.5.: Algorithms Ranked on Low Contention Policy

	Datasets			
Rank	**A.2.2**	**A.2.5**	**B.1**	**B.4**
1	SA	SA	SA	TS
2	LSA	TS	LA	LA
3	TS	LSA	TS	SA
4	LA	LA	LSA	LSA

Low SLA Violations Policy: This policy favors low SLA violations and hence

assigns a weight of 0.25 to both PDM and PDC. The remaining metrics are equally weighted with 0.071428571. Results are shown in Table 7.6.

Table 7.6.: Algorithms Ranked on Low SLA Violations Policy

	Datasets			
Rank	**A.2.2**	**A.2.5**	**B.1**	**B.4**
1	SA	SA	SA	TS
2	LSA	TS	LA	LA
3	TS	LA	TS	SA
4	LA	LSA	LSA	LSA

Low Energy Policy: This environment friendly policy favors low energy consumption and assigns a weight of 0.5 to energy saved, while the remaining metrics are weighted equally with 0.071428571. Results are shown in Table 7.7.

Table 7.7.: Algorithms Ranked on Low Energy Policy

	Datasets			
Rank	**A.2.2**	**A.2.5**	**B.1**	**B.4**
1	LSA	SA	SA	TS
2	SA	TS	LA	LA
3	TS	LA	TS	SA
4	LA	LSA	LSA	LSA

Analyzing on a coarse-grained level, the results reveal that for four of the five policies, Simulated Annealing ranked first with a total of 13 wins on small and medium datasets (A2.2, A2.5, B.1). Tabu Search ranked first for all policies on the large dataset (B.4) and accumulated 5 wins. Late Acceptance ranked first only for the high score policy on the medium dataset (B.1) and similarly, Late Simulated Annealing ranked first on the small dataset (A2.2) for the low energy policy. In presented work, Simulated Annealing is regarded as maximum yielding algorithm to implement most policies and increase return on investment (ROI).

7.5. Summary

This chapter presented a solution for SLA-aware resource management in cloud computing by solving the service consolidation problem framed for OpenShift PaaS. To author's knowledge, this is one of the first works on the subject using multiple Metaheuristic algorithms and evaluating their performance against a variety of aspects using formally defined models. Future prospects include expanding evaluations to more metrics, constraints and datasets. The presented work strengthens the case for SLA management on modern cloud infrastructures.

Chapter 8.

Unified Management Architecture

The SLA@SOI management framework provides a comprehensive and domain-independent solution to apply SLA management on top of IT stacks. Its open source and out of box distribution contains domain-specific examples to specialize the software components to local environments. However, this may require reasonable engineering effort to adapt the framework to the underlying technology and organizational practices may need to be revised. Many contemporary cloud providers are not prepared for such advanced automated solutions, where human intervention is minimal. This is partly due to the fact that cloud technologies and deployment trends are only recently stabilizing beyond the initial hype.

Further, many organizations opt for on-premise private clouds to conform to traditional practices regarding security and data protection. Layered cloud platforms representing IaaS, PaaS and SaaS services, as shown in the cloud value chain use case (see Section 4.1), are increasingly implemented on premise. Such layered cloud systems provide benefits to customers and providers. The customer can choose from IaaS resources such as VMs, PaaS containers or managed SaaS instances. The provider can elastically manage resources by scaling individual layer against demand. However, managing a broad variety of services can get difficult with growth. For this purpose, a reference framework is needed to separate concerns belonging to the technical plane and the business plane albeit in a cohesive manner. This also paves the way to incrementally introduce selected SLA management functions on cloud stacks already in use by the provider.

8.1. Service Oriented Cloud Computing Infrastructure

The Service Oriented Cloud Computing Infrastructure (SOCCI) by OMG[1] provides such a reference framework. SOCCI synergizes SOA with cloud computing. It identifies standard service-oriented components called Elements and Management Building Blocks (MBB) to manage cloud infrastructures. The Elements represent basic infrastructure units of a cloud e.g., compute, network, storage and facilities. The MBBs build management capabilities on top of Elements. MBBs can be divided among business and operational lines. The former manage capacity, provisioning, billing and monitoring while the latter manage virtualization, metering, configuration and location related operations.

The SOCCI framework analytically addresses cloud layers and identifies extension points, which need to be adapted to the underlying technology. However, SOCCI is not an implemented framework, therefore, a prototype architecture that is able to position SOCCI Elements and MBBs with respect to popular technology stacks can fill the lack of art on unified management architectures. With this motivation, a prototype architecture named as "GWDG Platform Cloud" was implemented during this thesis work to provide proof-of-concept implementation of the SOCCI framework. This work provides three contributions:

1. The IaaS, PaaS and SaaS capabilities are classified along the business and operational planes in a unified management architecture.

2. An extensible API to provision and monitor SaaS services using PaaS.

3. A web based AppStore as front-end to expose a catalog of managed services, and lifecycle control of deployed services through a dashboard view.

8.2. Prototype Architecture

As the service provider for Max-Planck Society of Germany and the University of Göttingen, GWDG owns high performing compute and storage systems consisting of 42 physical servers with a total of 2496 CPU cores and 9.75 Terabytes of RAM. There are 38 Dell PowerEdge C6145 servers with AMD Interlagos Opteron processors with 18.55 Terabytes raw disk capacity, and 3 Petabytes of distributed data storage. On this, a compute (IaaS) cloud has been developed using the KVM

[1] http://www.opengroup.org/soa/source-book/socci/index.htm

and OpenStack[2] technologies. Using a self-service web-based interface, customers can instantly provision Virtual Machines (VMs). However, this leaves a lot of operational overhead for SaaS users, e.g., to install a software, configure it with source repositories, load balancers and databases, which usually lie on different VMs. The Platform-as-a-Service (PaaS) model solves these issues by automating these tasks, allowing SaaS users to focus on applications while the provider can elastically manage the use of IaaS resources through the PaaS layer. For this purpose, the Platform Cloud used OpenShift Enterprise [94], a state of the art PaaS framework by RedHat. For elastic management of resources, the PaaS was based on top of GWDG's OpenStack compute cloud.

PaaS Application Marketplace

OpenShift provides an ecosystem that supports multiple languages, databases and middlewares in the form of pluggable and extensible execution environments called Cartridges. It also provides a template-based approach called QuickStart that allows hooks to control life cycle of an application e.g., through start, stop, delete scripts. QuickStarts and Cartridges cloud-enable an application. The OpenShift community has made several QuickStart available for public use[3]. These have been forked to preserve as a marketplace of applications.

PaaS Containers

OpenShift broker can use a QuickStart to provision an application on resource-constrained multi-tenant Linux containers. Containers can have small, medium or large sizes based on the capacity of assigned resources, using kernel namespaces, control groups (cgroups) and SELinux technologies. To date, this combination provides a technological breakthrough for the SLA Translation problem.

SLA Translation

The SLA translation is a process which is part of the service development phase of the SLA lifecycle (see Section 3.1). This activity requires benchmarking and performance engineering techniques to develop a resource consumption model of a specific service, considering various bounds on usage load and infrastructure

[2] http://www.openstack.org
[3] https://github.com/openshift-quickstart

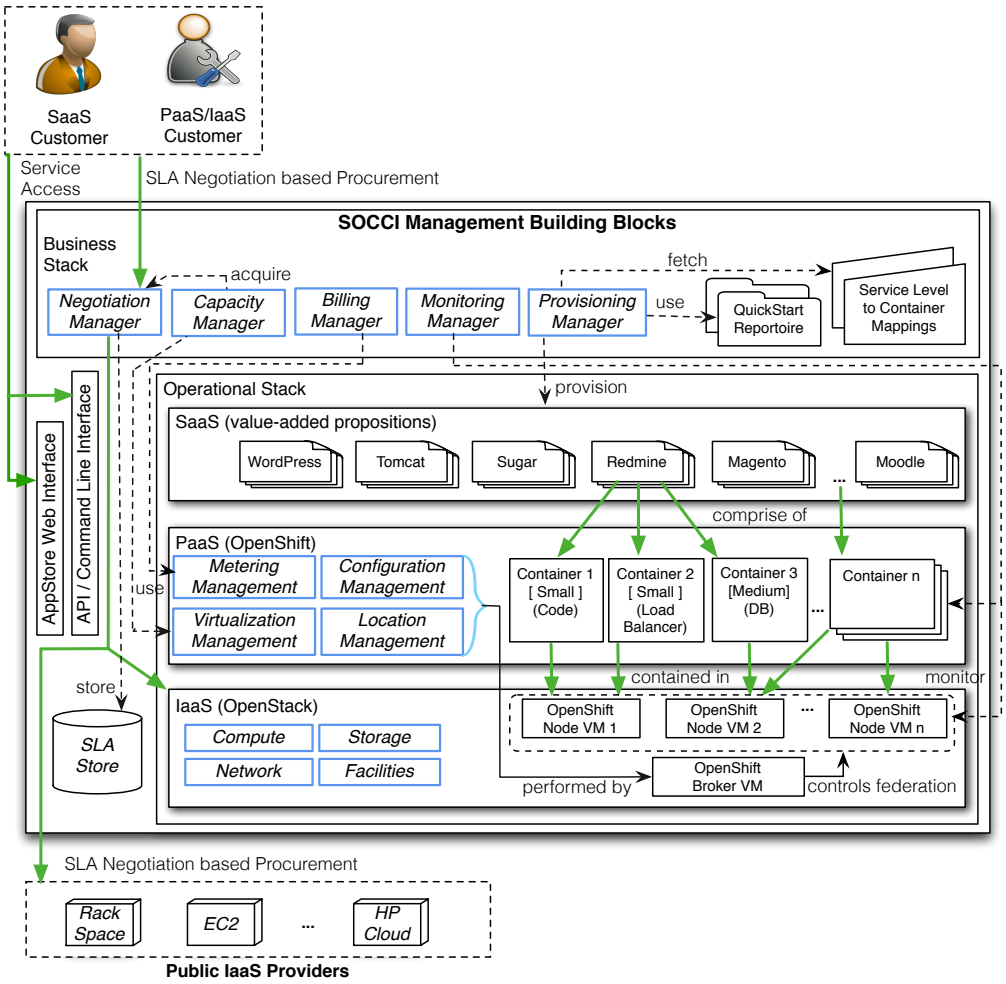


Figure 8.1.: Architecture Diagram of GWDG Platform Cloud Prototype

capacity. The SLA translation problem is out of scope for this thesis. However, basic experimentation done in this direction lead to proposing "Service Level to Container Mappings". These map high level QoS values to low level PaaS resource containers. The intention here is to summarize the basic concept as shown in Fig. 8.2. As seen in Fig. 8.2, based on the expected throughput, the quality of a container may be realized in the first step. Next, based on the desired response time, a certain quantity of containers is realized. These are then coupled with availability and backup values to identify the IaaS resources to use.

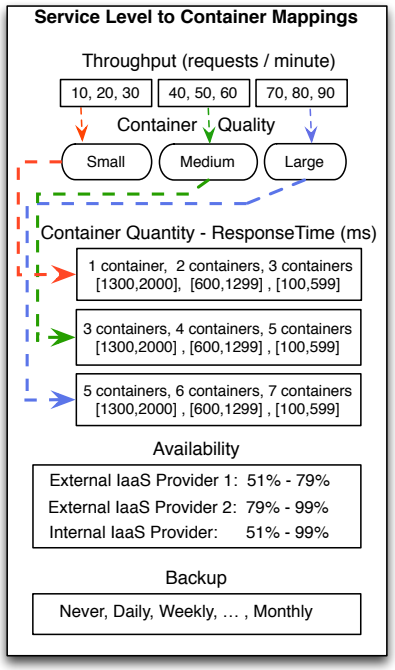


Figure 8.2.: Service Level to Container Mappings

8.2.1. Management Building Blocks

Fig. 8.1 shows the MBBs realized for the GWDG Platform Cloud. These are highlighted in blue and divided among business and operational stack.

The Provisioning Manager uses a QuickStart from its repertoire and the Service Level to Container mappings to provision an application instance with default settings. Instances can be provisioned without negotiating a custom SLA i.e. using traditional self-service means to procurement.

As an additional capability, the negotiation platform of the SLA@SOI project is added as the Negotiation Manager component. This serves as the first step to provide SLA based services leveraging the dynamic multi-round SLA (re)negotiation as presented in Chapter 5 and 6. If QoS-centered negotiations succeed for a service offering, provisioning is automatically performed using the SLA translation.

The Billing Manager uses metering management of OpenShift to bill the customer on pay per use basis. The Monitoring Manager uses an OpenTSDB based mechanism, as presented in [102,119] to collect resource usage for accounting and displaying realtime usage of CPU, memory, network and disk I/O. The Capac-

ity Manager supervises virtualization management of OpenShift, and helps the administrator decide when to acquire or release VMs from/to the IaaS layer.

In envisioned clouds, acquisition should be SLA-based and performed by the Negotiation Manager. The IaaS-agnostic PaaS layer allows to burst out to public IaaS if private IaaS lacks capacity. Cross provider infrastructure resources can be added on the fly, by running configuration scripts that setup a VM in the OpenShift federation and connect it to its Broker. The design builds upon existing capabilities of cloud stacks. However, the out of box capabilities for provisioning or monitoring are quite low level. These need to be leveraged in a customer-oriented design, which helps provider to build service offerings in a reusable manner and customers in performing lifecycle management of services.

8.3. Customer-oriented Design

8.3.1. A Unified API

One requirement for the GWDG Platform Cloud regards the front-end, using which SaaS and PaaS customers could instantiate and manage services. For this reason, a REST-based application programming interface (API) was developed to unify provisioning, billing and monitoring operations. The API provides coarse grained operations abstracting over OpenShift and OpenTSDB capabilities. The signature of the API is shown in Fig. 8.3. In addition, the negotiation interface is

```
List<Service> retrieveServices ( PaaSUser credentials )
Service createService ( PaaSUser credentials, ServiceGlossary id, String serviceName )
boolean destroyService ( PaaSUser credentials, String serviceName )
boolean startService ( PaaSUser credentials, String serviceName )
boolean stopService ( PaaSUser credentials, String serviceName )
boolean editService ( PaaSUser credentials, String serviceName )
String[] getServiceInfo ( PaaSUser credentials, Service service )
Bill getServiceBill ( PaaSUser credentials, String serviceName )
Data monitorService ( String duration, String durationunit, String aggregator, String metric, String tag, String index )
```

Figure 8.3.: API Signature

available for SLA negotiation as presented in Chapter 5 (Section 5.4). This work

could only be partially integrated due to time constraints. A higher priority was assigned to the research and engineering outcomes, while implementing a complete production system was out of scope of this thesis.

8.3.2. AppStore Frontend

The Platform Cloud prototype features an AppStore front-end to expose the marketplace of managed SaaS instances to customers. These include content management services like Redmine, Tomcat web-container, customer relationship management software like Sugar CRM, blogging tools like WordPress, DokuWiki, MediaWiki and many more. These can be conveniently instantiated on the cloud using single click. Behind the scenes, the API communicates with the OpenShift broker, passes appropriate parameters and ensures that the application instance is properly configured and remotely accessible over a pubic URL.

Fig. 8.4 shows a partial view of available services.

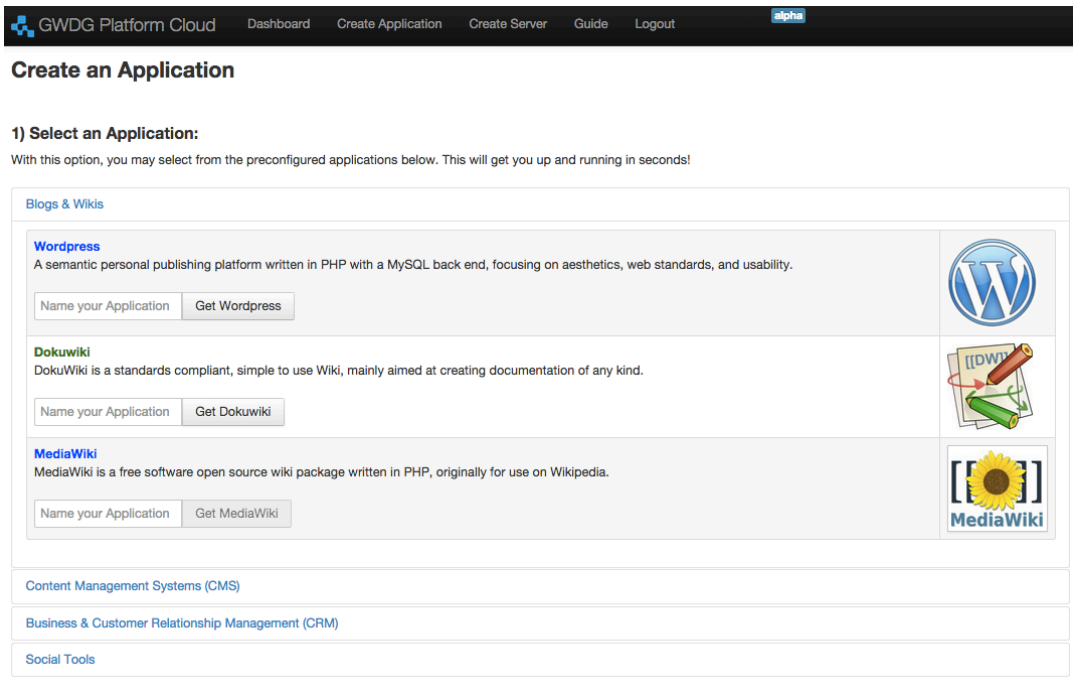


Figure 8.4.: View of Service Catalogue

Fig. 8.5 shows a view of customer's homepage. It displays all deployed services in a dashboard layout. The lifecycle controls (for start, stop, snapshot, edit, monitor, bill to date) are exposed as buttons, which invoke API operations using

the JQuery JavaScript library in AJAX fashion to avoid unnecessary request submission for the entire page.

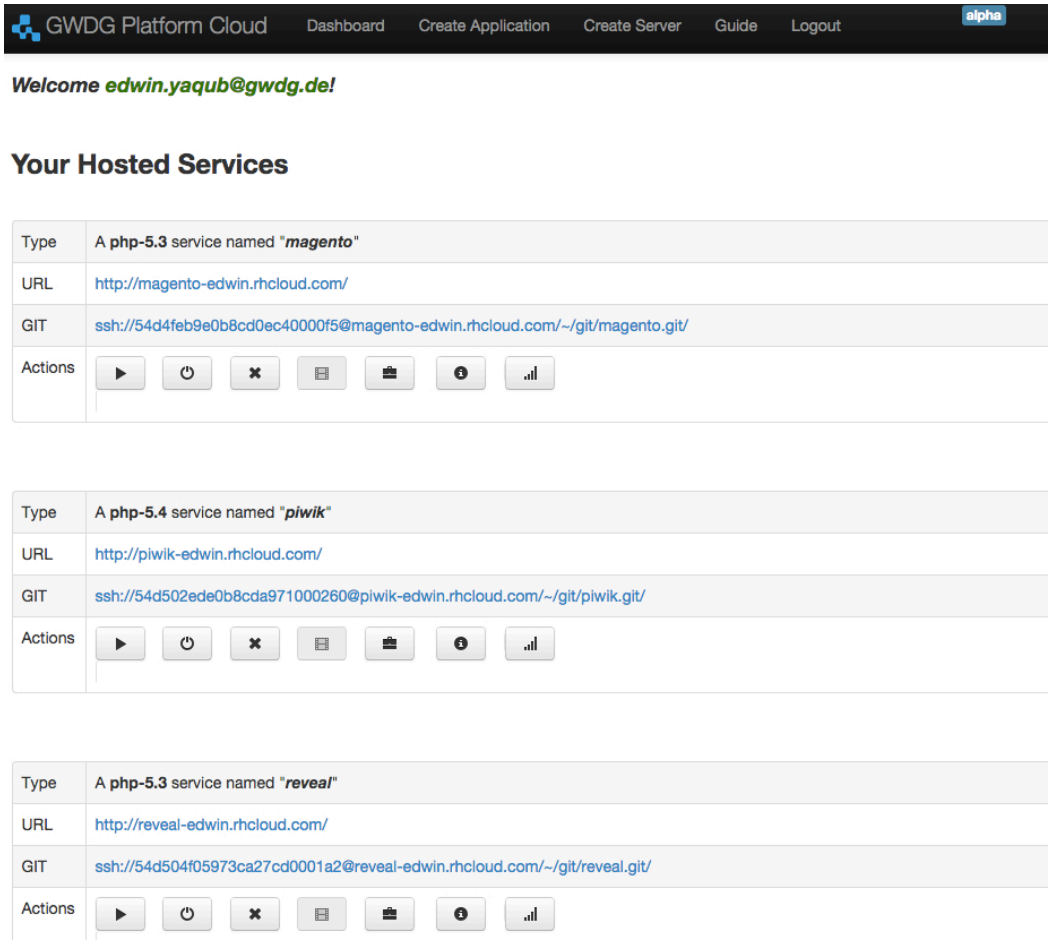


Figure 8.5.: View of Customer Homepage

8.4. Comparison with OCCI

The Open Cloud Computing Interface (OCCI) [125] is a community-led initiative established by the Open Grid Forum. OCCI comprises a set of open specifications to standardize resource management tasks on various cloud infrastructures. The basic goal is to define API based access to manipulate compute, storage and network resources. These functions can be implemented by using the OCCI Mixin extension mechanism. In practice, OCCI implementations to date support

limited cloud or infrastructure backends with focus on IaaS. Further, support for client side is rather limited to few programming or scripting languages. Many specification drafts regarding monitoring, billing, advanced reservation, negotiation and agreement features are still work in progress.

Our refinement of SOCCI has some overlap with OCCI, especially regarding the API based approach to access cloud backends. Nevertheless, there are certain subtle differences as well. First, developing the GWDG Platform Cloud as a prototypical implementation of SOCCI, we dealt mostly with OpenShift (PaaS) and OpenTSDB backends. These are currently not supported in any available OCCI implementations. Secondly, our requirements regarding service catalogue and managing lifecycle operations on service instances brings the focus on end customer. OCCI on the other hand mainly focuses on administrative concerns of cloud providers. Thirdly, we deal with layered cloud backends, while in current OCCI implementations, usually a single backed system is targeted which limits their reuse in our scenario.

Our API can indeed be converted as an OCCI extension, which could be highly differentiated due to its focus on PaaS as enabler of SaaS. However, this is kept as a future extension possibility to this work. For dissemination to broader community, the source code of the prototype system and API is opensourced on the Gitlab repository of EU project PaaSage [126].

8.5. Summary

This chapter presented a prototype architecture compatible with the SOCCI framework specification. This conceptualized Elements and Management Building Blocks of SOCCI in relation to popular cloud / cloud-related stacks that are often layered together. These included OpenStack (IaaS), OpenShift (PaaS) and OpenTSDB (for scalable monitoring). The architecture classifies IaaS, PaaS and SaaS capabilities along the business and operation planes. An API was developed to unify management actions required across multiple cloud layers. This enabled a catalogue of SaaS offerings, which could be provisioned and monitored at scale. The approach was compared with OCCI standardization initiative to view this contribution in larger perspective. This work provided proof of concept implementation to assess state of the art cloud stacks in oder to gradually incorporate SLA management features like negotiation, consolidation or SLA enforcement.

Part III.

Conclusion

Chapter 9.

Conclusion

In this chapter, the contributions made by this thesis are summarized, followed by an introspective discussion on thesis limitations. Finally, interesting future directions are identified, some of which are enabled by this work while others may extend this work, hopefully beyond the constraints imposed by this thesis.

9.1. Summary of Contributions

This work was motivated by the need to enable added-value services and their reliable delivery using the cloud computing paradigm. This thesis found inspiration in Service Level Agreement as instrument of choice to address various concerns that unfold. The main objectives were structured around developing generic methods to automate SLA management in cloud computing. The methods enable concrete application through policy based controls. Two key perspectives based on the SLA lifecycle were identified. These highlight the need for agility in adapting to changing business landscape, by means of value creation and infrastructure management.

From a high level view, the proposed methods contribute towards establishing trust, enabling customizable service propositions, diversifying selling mechanisms, maximizing business utility of procurements, and fulfilling SLA commitments by optimally operating the infrastructure. The thesis has progressed the state of art and various novelties proposed and disseminated by the author have been cited by fellow but unrelated researchers [39, 127–139].

The contributions listed in Chapter 1 towards the identified research questions have already been presented in Chapters 5-8. In order to conclude the thesis, a summary of these contributions and their impacts is presented below.

Contribution 1: On the negotiation front, this work has made an attempt to

redefine the relationship between producer and consumer of a service by leveraging SLA negotiations. To this, the role of negotiation protocols has been revisited and highlighted as selling models for services, alongside dependency resolution in value chains. A generic method to create executable negotiation protocols is considered an important contribution for these settings. Implementing protocols as declarative rules provided the solution to finely couple organizational policies with negotiation interactions. This maintained separation of concerns and promoted reusability with ease of configuration. Unlike other works, these contributions do not impose a single protocol on providers, rather a step-wise guided method is laid out which focuses on key concepts of protocol design namely modeling, verification and implementation. On the other hand, concrete application of this method led to the SBNP protocol, which is a tangible outcome and one that can be reused to replace the rigid take-it-or-leave-it offerings prevalent today. These contributions are published as a book chapter:

A Generic Platform for Conducting SLA Negotiations, Yaqub, E., Wieder, P., Kotsokalis C., Mazza V., Pasquale L., Lambea, R. J., Garcia, G. S., and Escamez, C.A., Service Level Agreements for Cloud Computing, (Eds.) Wieder P., Butler J.M., Theilmann W., Yahyapour R., 2010

and as a conference paper:

A Protocol Development Framework for SLA Negotiations in Cloud and Service Computing, Yaqub, E., Yahyapour, R., Wieder, P., and Lu, K. In: Proceedings of the 9th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON), 2012

The protocol and the negotiation platform had been demonstrated in atleast four industrial use cases of the EU project SLA@SOI. These included chained negotiation scenarios in various service sectors. The work was also deployed in the EU project Contrail, nevertheless for a negotiation/provisioning scenario beyond the original vision. This adoption testifies to the broader applicability of the proposed solution such as for cloud brokerage or other intermediary services. The cross-project adoption of this work is a good indicator for return of European Commission's investment in SLA management initiatives.

Contribution 2: The work on negotiation strategies has revealed the complex dynamics that hide behind the SLA gap between customers and providers. These are partly responsible why stakeholders are reluctant in implementing

SLA negotiations for automated service procurement. Contributions made in this area include two algorithms that are suitable for small to large SLA contract spaces. These extend the art on tit-for-tat strategies. Their performance however depends on how their behavioral variables are tuned. This work provides refinement on associating business utility to cloud-relevant SLA templates and the utility of established SLA from the perspective of customer and provider. These contributions are published as a conference paper:

Optimal Negotiation of Service Level Agreements for Cloud-based Services through Autonomous Agents, Yaqub, E., Yahyapour, R., Wieder, P., Kotsokalis C., Lu, K., and Jehangiri, A.I. In: 11th IEEE International Conference on Services Computing (IEEE SCC), Alaska, USA, 2014

and also submitted as a journal article:

Protocol-generic and Utility optimizing Negotiations for SLA based Service Procurement in Cloud Markets, Yaqub, E., Yahyapour, R., Wieder, P., Kotsokalis, C., Lu, K., and Jehangiri, A. I., Future Generation Computer Systems, (Status: in review), 2015

The tournament based evaluations used state of the art strategies and considered preference conflicts in negotiation domains designed to reflect mission critical, fault tolerant, retail services or commodities. The proposed strategy algorithms show acceptable stability levels, which is regarded positively. On the other hand, results show that no single algorithm is the most promising in all negotiation domains, against multiple notions of utility, and when faced with diverse opponent strategies. These results nevertheless advance understanding on a complex area, being pursued by an active research community.

Contribution 3: The work on cloud infrastructure management developed an efficient solution to a relatively novel service consolidation problem. To author's knowledge, contributions in this area are one of the first on SLA-aware resource management in PaaS clouds. PaaS service model, due to its focus on services in addition to machines and their geographic distribution has attracted significant attention in recent research and commercial sectors. The service consolidation problem contends dependencies among services - a vital concern of SLA based services which has been addressed in this thesis but where prior art lacked.

The SLA violation model given in this work accounts the notions of utilization, resource contention and migrations proposed by the consolidation algorithms.

Estimating potential SLA violations is one of the weak links in SLA management of clouds which is strengthened by this work. The modeling of constraints from a publicly available problem definition and accompanying datasets provided the base for framing the service consolidation problem, while avoiding the bias of self-generated datasets. The developed solution is generic by design since the applied search algorithms are domain and dataset independent. However, it was deemed necessary that the problem be characterized for a commercial PaaS cloud stack/infrastructure, to establish good relevance between theory and practice.

Furthermore, a simple but elegant use of utility function was proposed to align business and IT strategy. This harnessed various quality metrics to rank consolidated solutions according to high level business policies, so the most preferred consolidation plan can be executed. These contributions are published as a conference paper:

Metaheuristics-based Planning and Optimization for SLA-aware Resource Management in PaaS Clouds, Yaqub, E., Yahyapour, R., Wieder, P., Kotsokalis C., Lu, K., and Jehangiri, A.I. In: 7th IEEE/ACM International Conference on Utility and Cloud Computing (IEEE/ACM UCC), London, UK, 2014

Although it was not in the scope of this thesis to integrate consolidation plans in a running cloud system, the author is of the opinion that the scientific challenge has been adequately solved.

Contribution 4: Finally, a lightweight, unified and extensible architecture was developed as proof of concept prototype to understand the business and operational aspects of layered cloud systems. Here, a unified API was implemented which provided the basis for a web based AppStore user interface. This enables SaaS customers to easily instantiate and manage lifecycle of applications from a growing service catalogue. This work helped investigate how selective SLA management features can be progressively integrated with existing cloud infrastructures. These contributions are included in the paper published at the IEEE SCC conference 2014.

9.2. Discussion

The journey that led to these contributions had to confront certain challenges. Best effort was made to resolve these challenges in the most pragmatic and novel

fashion, within the time constraints faced by a doctoral thesis.

One dilemma was faced in keeping the methods generic, yet establish their applicability concretely. For instance, it became obvious that a generic protocol development method cannot be overly engineered and consummated as a set of libraries. This led to a method which coherently applied existing techniques for model checking and declarative rules to shape negotiation protocols as communicating finite state machines. Nevertheless, to reap the advantages of the given method, a reasonable understanding is expected in these areas. The thesis work has tried to simplify this task by highlighting key aspects and implementing a specific protocol, which can be followed as an example.

Although the rationale to establish agreement is easily followed, understanding negotiation dynamics requires expository analysis of economic concepts in a given domain. These concepts include utility, opposition between utility functions of customer and provider, the role of individual and social welfare, Pareto-frontier, Nash, Kalai-Smorodinsky and the irrational Utopia point. These pose some degree of challenge depending on the background knowledge of the practitioner.

The work on SLA-aware cloud resource management addressed multiple quality aspects, whose models were formally defined or reused from prior art. Other researchers may have differing opinions regarding some of these models. This does not limit the validity of this work but domain-specific application of given models can be further qualified. The application of Metaheuristic algorithms is one of the strengths of proposed solution, given that service consolidation is a combinatorial optimization problem. However, formulating the problem demands some understanding of multi-criteria optimization and constraint based search.

The multiplicity of used technologies or programming paradigms could be seen as a limitation. Nevertheless, the author believes that although proposed methods lie within expert domains, these can be effectively mastered with due regard. Adoption may be eased by the fact that open source, time tested, freely available and wherever possible, industrial strength tools were used. For instance, Spin model checker is used for protocol verification, strategies are benchmarked using GENIUS simulator, RedHat's Drools rule engine is used to implement and execute protocol rules, and RedHat's OptaPlanner solver is used for optimization.

9.3. Future Extension Possibilities

The following list offers interesting directions for future extension of this work.

- Business models targeting multi-staged service propositions can be implemented as negotiation protocols. As hinted in Chapter 5, this could use SLA negotiations to sell bundled services, representing mandatory-optional constituents from IaaS, PaaS and SaaS offerings. Further more, SBNP could be reused to offer spot-based services, where bounds regarding performance, early warning times or auto-checkpoints can be fixed in SLA.
- Cost models (in terms of financial unit) need to be attached to negotiation models. Cost can be determined dynamically based on e.g., negotiation rounds, negotiation time, service price, or as a fixed fee if a centralized negotiation marketplace (like Ebay) is used. Negotiation costs encourage negotiators to quickly converge on SLA, as these economic derivatives may get large if negotiated procurements become modus operandi for clouds. This direction is briefly addressed using discounted utilities in Chapter 6.
- In this work, evaluations were restricted to tournaments - an approach also used in the automated negotiation agents competition (ANAC). However, the tournament based approach can be extended to perform empirical game theoretic (EGT) analysis as in [80]. Although game theory is not generally applicable in this work because negotiators are unaware of opponent's utility function and conceding tactics, mean outcome based EGT analysis can discover *possible* Nash Equilibrium(ia), which can reduce the strategy space to the best few. Depending on the pool of strategies, many negotiation tournaments may need to be assessed due to strategy-profile combinations. The outcomes however remain uncertain until the very end.
- Work on the SLA-aware resource management in clouds can be extended to compare Metaheuristics with global search algorithms such as Genetic algorithms. This can examine consolidation quality and how algorithms scale against time or memory when decision variables or state space increases.
- SLA enactment is another area in automated SLA management. This allows to prevent SLA violations by learning the resource usage trends and adjusting proactively to reduce the need for frequent consolidation. In case of violations, root cause analysis (RCA) methods provide corrective actions.

List of Tables

List of Figures

Bibliography

[1] A. Cartlidge, A. Hanna, C. Rudd, I. Macfarlane, J. Windebank, and S. Rance. An Introductory Overview of ITIL V3. Best Management Practice, 2007.

[2] IDC. Predictions 2014: Battles for Dominance - and Survival - on the 3rd Platform. Technical Report, December 2013.

[3] IDC. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. Technical Report, Available: `http://idcdocserv.com/1678`. Accessed: [21/11/2015].

[4] Gartner. Press Release, Available: `http://www.gartner.com/newsroom/id/2592315`. Accessed: [21/11/2015].

[5] ProgrammableWeb. Mashups and OpenAPIs. `http://www.programmableweb.com`. Accessed: [21/11/2015].

[6] Gartner. 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business, Available: `http://www.gartner.com/newsroom/id/2819918`. Accessed: [21/11/2015].

[7] TNW. News Article, Available: `http://thenextweb.com/facebook/2014/01/29/facebook-passes-1-23-billion-monthly-active-users-945-million-mobile-users-757-million-daily-users`. Accessed: [21/11/2015].

[8] J. Rosenberg and A. Mateos. *The Cloud at Your Service*, Manning Publications, 2010.

[9] NESSI. Vision Document, Available: `http://www.nessi-europe.com/files/Docs/NESSI_VISION.pdf`. Accessed: [21/11/2015].

[10] Forrester. Service App Stores Will Reshape Corporate Tech Management. Technical Report, 2014.

[11] The TeleManagement Forum. SLA Management Handbook, Volume 2, Concepts and Principles, Release 2.5, July 2005.

[12] The TeleManagement Forum. Enabling End-to-End Cloud SLA Management, Version 0.7, 2013.

[13] C. Kotsokalis. Automated Hierarchical Service Level Agreements - Generic Management Principles and Application to Multi-Domain Infrastructure-as-a-Service. Doctoral Dissertation, 2010.

[14] NESSI. Strategic Research Agenda, Public Document, Volume 3.2 Revision 2.0, Available: `http://www.nessi-europe.com/files/Docs/NESSI_SRA_VOL_3.pdf`. Accessed: [21/11/2015].

[15] V. Kundra. Federal Cloud Computing Strategy. Technical Report, 2011.

[16] C. Cerin, C. Coti, P. Delort, F. Diaz, M. Gagnaire, Q. Gaumer, N. Guillaume, J. Lous, S. Lubiarz, J. Raffaelli, K. Shiozaki, H. Schauer, J. Smets, and L. Seguin. Downtime statistics of current cloud solutions. Technical Report, International Working Group on Cloud Computing Resiliency, 2013.

[17] P. Bianco, G. A. Lewis, and P. Merson. Service Level Agreements in Service-Oriented Architecture Environments. Technical Note, Software Engineering Institute, September 2008.

[18] Pew Research Center. The future of cloud computing. Technical Report, Available: `http://www.elon.edu/docs/e-web/predictions/expertsurveys/2010survey/PIP_Future_of_Internet_2010_cloud.pdf`. Accessed: [21/11/2015].

[19] D. Canellos. How the Internet of Things will feed Cloud Computing's next evolution. News Article, `https://blog.cloudsecurityalliance.org/2013/06/05/how-the-internet-of-things-will-feed-cloud-computings-next-evolution`. Accessed: [21/11/2015].

[20] Amazon EC2. `http://aws.amazon.com/ec2`. Accessed: [21/11/2015].

[21] IDC. Technical Report, Available: `http://www.salesforce.com/events/docs/IDC_Force_ROI_Study_Sept2009.pdf`. Accessed: [21/11/2015].

[22] Wikipedia. Force.com Platform. `http://en.wikipedia.org/wiki/Salesforce.com#Force.com`. Accessed: [21/11/2015].

[23] Object Management Group (OMG). SOCCI Framework. `http://www.opengroup.org/soa/source-book/socci/index.htm`. Accessed: [21/11/2015].

[24] Cloud Standards Customer Council. Public Cloud Service Agreements: What to Expect and What to Negotiate. Technical Report, 2013.

[25] *Harvard Business Essentials: Negotiation*. Harvard Business School Publishing Corporation, 2003.

[26] 4CaaSt Project. `http://4caast.morfeo-project.org`. Accessed: [21/11/2015].

[27] Cloud4SOA Project. `http://www.cloud4soa.eu`. Accessed: [21/11/2015].

[28] IRMOS Project. `http://www.irmosproject.eu`. Accessed: [21/11/2015].

[29] R. Kuebert, G. Georgina, O. Karsten, and O. Eduardo. Enhancing the SLA framework of a virtualized service platform by dynamic re-negotiation. In *eChallenges 2010*, IEEE, 2010.

[30] G. Kousiouris, D. Kyriazis, S. Gogouvitis, A. Menychtas, K. Konstanteli, and T. Varvarigou. Translation of application-level terms to resource-level attributes across the Cloud stack layers. In *IEEE Symposium on Computers and Communications (ISCC)*, 2011.

[31] OPTIMIS Project. `http://www.optimis-project.eu`. Accessed: [21/11/2015].

[32] OPTIMIS Service Manifest. Scientific Report, Available: `http://www.optimis-project.eu/sites/default/files/content-files/document/optimis-public-deliverable-service-manifest-scientific-report.pdf`. Accessed: [21/11/2015].

[33] CONTRAIL Project. `http://contrail-project.eu`. Accessed: [21/11/2015].

[34] I. Foster. Service-oriented science, Science, vol. 308, no. 5723, pp. 814-817, 2005.

[35] I. Marsa-Maestre, M. Klein,C. M. Jonker, and R. Aydogan. From problems to protocols: Towards a negotiation handbook. *Decision Support Systems*, vol. 60, pp. 39-54, 2014.

[36] S. Hudert and Torsten Eymann. The BabelNEG System - A prototype Infrastructure for protocol-generic SLA Negotiations. In *Wirtschaftsinformatik*, pp. 44, 2011.

[37] L. Wu and R. Buyya. Service Level Agreement (SLA) in Utility Computing Systems. *IGI Global*, 2012.

[38] S. Paurobally and J. Cunningham. Achieving common interaction protocols in open agent environments. In *Proceedings of the Workshop on Challenges in Open Agent Systems, the 2nd International Joint Conference on Autonomous Agents Multi-Agent Systems (AAMAS-03)*, Melbourne, Australia, 2003.

[39] A. Menychtas, J. Vogel, A. Giessmann, A. Gatzioura, S. G. Gomez, V. Moulos, F. Junker, M. Müller, D. Kyriazis, K. Stanoevska-Slabeva, and T. Varvarigou. 4CaaSt Marketplace: An Advanced Business Environment for Trading Cloud Services. *Future Generation Computer Systems*, vol. 41, pp.104-120, 2014.

[40] H. Li, S. Y. W. Su, and H. Lam. On Automated e-Business Negotiations : Goal, Policy, Strategy, and Plans. vol. 13, no. 1, pp. 1-29, 2006.

[41] W. L. Yeung. Behavioral modeling and verification of multi-agent systems for manufacturing control. *Expert Systems with Applications*, vol. 38, pp. 13555-13562, 2011.

[42] Drools Rule Engine. `http://www.drools.org`. Accessed: [21/11/2015].

[43] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599-616, 2009.

[44] J. Yan, R. Kowalczyk, J. Lin, M.B. Chhetri, S.K. Goh, and J. Zhang. Autonomous service level agreement negotiation for service composition provision. *Future Generation Computer Systems*, vol 23, no. 6, pp. 748-759, 2007.

[45] E. Yaqub, P. Wieder, C. Kotsokalis, V. Mazza, J. L. Rueda, and S. Garcia. A Generic Platform for Conducting SLA Negotiations. *Service Level Agreements for Cloud Computing*, pp. 187-206, Springer New York, 2011.

[46] E. Yaqub, R. Yahyapour, P. Wieder, and K. Lu. A Protocol Development Framework for SLA Negotiations in Cloud and Service Computing. In *9th International Conference on Economics of Grids, Clouds, Systems and Services (GECON 2012)*, pp. 1-15, Springer Berlin Heidelberg, 2012.

[47] E. Yaqub, R. Yahyapour, P. Wieder, C. Kotsokalis, K. Lu, and A. I. Jehangiri. Optimal Negotiation of Service Level Agreements for Cloud-based Services through Autonomous Agents. In *11th IEEE International Conference on Services Computing (SCC)*, pp. 59-66, IEEE, 2014.

[48] E. Yaqub, R. Yahyapour, P. Wieder, A. I. Jehangiri, K. Lu, and C. Kotsokalis. Metaheuristics-based Planning and Optimization for SLA-aware Resource Management in PaaS Clouds. In *7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, pp. 288-297, IEEE, ACM, 2014.

[49] E. Yaqub and A. Barroso. Distributed guidelines (DiG): a software framework for extending automated health decision support to the general population. *Perspectives in Health Information Management/AHIMA*, American Health Information Management Association, vol. 7 (Summer), no. 1b, 2010.

[50] G. J. Holzmann. The Model Checker Spin. *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279-295, 1997.

[51] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specications for Finite-State Verication. In *21st IEEE International Conference on Software Engineering (ICSE)*, pp. 411-420, 1999.

[52] SAnToS Laboratory, Kansas State University. Property Specification Patterns. Available: `http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml`. Accessed: [21/11/2015].

[53] P. Karaenke and S. Kirn. A Multi-tier Negotiation Protocol for Logistics Service Chains. In *Proceedings of the 18th European Conference on Information Systems (ECIS)*, 2010.

[54] S. Hudert, T. Eymann, H. Ludwig, and G. Wirtz. A Negotiation Protocol Description Language for Automated SLA Negotiations. In *Proceedings of the IEEE Conference on Commerce and Enterprise Computing (CEC)*, pp. 162-169, 2009.

[55] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web Service Level Agreement (WSLA) Language Specification. Available: `http://www.research.ibm.com/people/a/akeller/Data/WSLASpecV1-20030128.pdf`. Accessed: [21/11/2015].

[56] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Available: `http://www.ogf.org/documents/GFD.107.pdf`. Accessed: [21/11/2015].

[57] O. Waeldrich, D. Battré, F. Brazier, K. Clark, M. Oey, A. Papaspyrou, P. Wieder, and W. Ziegler. WS-Agreement Negotiation Version 1.0 Specification. Available: `http://ogf.org/documents/GFD.193.pdf`. Accessed: [21/11/2015].

[58] K. T. Kearney and F. Torelli. The SLA Model. *Service Level Agreements for Cloud Computing*, Part 4, Springer New York, pp. 43-67, 2011.

[59] FIPA. Iterated Contract Net Interaction Protocol Specification. Available: `http://www.fipa.org/specs/fipa00030/SC00030H.pdf`. Accessed: [21/11/2015].

[60] FIPA. ACL Message Structure Specification. Available: `http://www.fipa.org/specs/fipa00061/SC00061G.pdf`. Accessed: [21/11/2015].

[61] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1st Edition, 1990.

[62] M. Klein, P. Faratin, H. Sayama, and Y. Bar-Yam. Negotiating Complex Contracts. *Journal of Group Decision and Negotiation*, vol. 12, no. 2, pp. 111-125, 2003.

[63] W. Theilmann, J. Happe, C. Kotsokalis, A. Edmonds, K. Kearney, and J. Lambea. A Reference Architecture for Multi-Level SLA Management. *Journal of Internet Engineering*, vol. 4, no. 1, pp. 289-298, 2010.

[64] M. Kamel and S. Leue. Formalization and validation of the General Inter-ORB Protocol (GIOP) using PROMELA and SPIN. *International Journal of Software Tools for Technology Transfer (STTT)*, vol. 2, no. 4, pp. 394-409, 2000.

[65] J-P. Katoen. Lecture on Model Checking. Available: `http://www-i2.informatik.rwth-aachen.de/Teaching/Course/MC/2005/mc_lec4.pdf`. Accessed: [21/11/2015].

[66] E. Yaqub. SBNP model in PROMELA. Available: `http://sourceforge.net/p/ey-negprotocol/code`. Accessed: [21/11/2015].

[67] A. Chavez, D. Dreilinger, R. Guttman, and P. Maes. A real-life experiment in creating an agent marketplace. *Software agents and soft computing towards enhancing machine intelligence*, Springer Berlin Heidelberg, pp. 160-179, 1997.

[68] M. Comuzzi and B. Pernici. A framework for QoS-based Web service contracting. *ACM Transactions on the Web (TWEB)*, vol. 3, no. 3, 2009.

[69] WSAG4J Framework. `http://wsag4j.sourceforge.net/site/`. Accessed: [21/11/2015].

[70] I. U. Haq and E. Schikuta. Aggregation patterns of service level agreements. In *Proceedings of the 8th International Conference on Frontiers of Information Technology*. ACM, 2010.

[71] M. A. R. Gonzalez, P. Chronz, K. Lu, E. Yaqub, B. Fuentes, A. Castro, H. Foster, J. L. Rueda, and A. E. Chimeno. GSLAM - The Anatomy of the Generic SLA Manager. *Service Level Agreements for Cloud Computing*, pp. 167-186, Springer New York, 2011.

[72] P. Chronz and P. Wieder. Integrating WS-Agreement with a framework for service-oriented infrastructures. In *11th IEEE/ACM International Conference on Grid Computing (GRID)*, pp. 225-232, 2010.

[73] European Commission. Cloud Computing Service Level Agreements - Exploitation of Research Results. Technical Report, Available: `http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=2496`. Accessed: [21/11/2015].

[74] M. Chhetri, J. Lin, S. Goh, J. Yan, J. Zhang, and R. Kowalczyk. A coordinated architecture for the agent-based service level agreement negotiation of web service composition. In *Australian Software Engineering Conference*, 2006.

[75] E. D. Nitto, M. D. Penta, A. Gambi, G. Ripa, and M. L. Villani. Negotiation of service level agreements: An architecture and a search-based approach. Springer Berlin Heidelberg, pp. 295-306, 2007.

[76] F. H. Zulkernine and P. Martin. An adaptive and intelligent SLA negotiation system for web services. *IEEE Transactions on Services Computing*, vol. 4, pp. 31-43, January 2011.

[77] R. Lin, S. Kraus, T. Baarslag, D. Tykhonov, K. Hindriks, and C. M. Jonker. GENIUS : An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence*, vol. 30, no. 1, pp. 48-70, 2014.

[78] TU Delft. GENIUS platform. `http://ii.tudelft.nl/genius`. Accessed: [21/11/2015].

[79] T. Baarslag, K. Hindriks, C. Jonker, S. Kraus, and R. Lin. The first automated negotiating agents competition (ANAC 2010). *New Trends in Agent-Based Complex Automated Negotiations*, pp. 113-135, Springer, 2010.

[80] T. Baarslag, K. Fujita, E. H. Gerding, K. Hindriks, T. Ito, N. R. Jennings, C. Jonker, S. Kraus, R. Lin, V. Robu, and C. R. Williams. Evaluating practical negotiating agents: Results and analysis of the 2011 international competition. *Artificial Intelligence*, vol. 198, pp. 73-103, May 2013.

[81] T. Baarslag, K. Hindriks, and C. Jonker. Effective acceptance conditions in real-time automated negotiation. *Decision Support Systems*, 2013.

[82] H. Raiffa. *The Art and Science of Negotiation*. Harvard University Press, 1982.

[83] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, vol. 24, no. 3-4, pp. 159-182, 1998.

[84] S. Kraus. *Strategic Negotiation in Multiagent Environments*. The MIT Press, 2001.

[85] J. Li and R. Yahyapour. Learning-based negotiation strategies for grid scheduling. In *6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 06)*, 2006.

[86] R. M. Coehoorn and N. R. Jennings. Learning on opponent's preferences to make effective multi-issue negotiation trade-offs. In *Proceedings of the 6th international conference on Electronic commerce*, ACM, 2004.

[87] K. Hindriks and D. Tykhonov. Opponent modelling in automated multi-issue negotiation using bayesian learning. In *Proceedings of the 7th international joint conference on autonomous agents and multiagent systems*, International Foundation for Autonomous Agents and Multiagent Systems, vol. 1, pp. 331-338, 2008.

[88] G. C. Silaghi, L. D. Serban, and C. M. Litan. A time-constrained SLA negotiation strategy in competitive computational grids. *Future Generation Computer Systems*, vol. 28, pp. 1303-1315, 2012.

[89] S. Kawaguchi, K. Fujita, and T. Ito. AgentK: Compromising strategy based on estimated maximum utility for automated negotiating agents. *New Trends in Agent-Based Complex Automated Negotiations*, pp. 137-144, Springer Berlin Heidelberg, 2012.

[90] T. van Krimpen, D. Looije, and S. Hajizadeh. Hardheaded. *Complex Automated Negotiations: Theories, Models, and Software Competitions*, pp. 223–227, Springer Berlin Heidelberg, 2013.

[91] L. Ilany and Y. Gal. Algorithm Selection in Bilateral Negotiation. In *Workshops at 27th AAAI Conference on Artificial Intelligence*, 2013.

[92] S. Chen, H. Ammar, K. Tuyls, and G. Weiss. Optimizing complex automated negotiation using sparse pseudo-input gaussian processes. In *Proceedings of*

the 2013 international conference on autonomous agents and multi-agent systems, International Foundation for Autonomous Agents and Multiagent Systems, pp. 707-714, 2013.

[93] SLA@SOI Project. `http://sla-at-soi.eu`. Accessed: [21/11/2015].

[94] OpenShift. `https://www.openshift.com`. Accessed: [21/11/2015].

[95] Cloud Foundry. `http://www.cloudfoundry.org`. Accessed: [21/11/2015].

[96] PaaSage Project. `http://www.paasage.eu`. Accessed: [21/11/2015].

[97] IDC. Press Release, Available: `http://www.idc.com/getdoc.jsp?containerId=prUS24435913`. Accessed: [21/11/2015].

[98] European Commission. Unleashing the Potential of Cloud Computing in Europe, Technical Report, Available: `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2012:0529:FIN:EN:PDF`. Accessed: [21/11/2015].

[99] S. Paurobally, C. van Aart, V. Tamma, M. Wooldridge, and P. Van Hapert. Web services negotiation in an insurance grid. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems*, ACM, 2007.

[100] P. McKee, S. Taylor, M. Surridge, and R. Lowe, SLAs, Negotiation, and Challenges. *Market-Oriented Grid and Utility Computing*, John Wiley & Sons, Inc. 2010.

[101] S. Garcia-Gómez, M. Jimenez-Ganan, Y. Taher, C. Momm, F. Junker, J. Biro, A. Menychtas, V. Andrikopoulos, and S. Strauch. Challenges for the comprehensive management of Cloud services in a PaaS framework. *Scalable Computing: Practice and Experience*, vol. 13, no. 3, 2012.

[102] A. I. Jehangiri, E. Yaqub, and R. Yahyapour. Practical Aspects for Effective Monitoring of SLAs in Cloud Computing and Virtual Platforms. In *CLOSER*, pp. 447-454, 2013.

[103] K. Lu, R. Yahyapour, E. Yaqub, and C. Kotsokalis. Structural optimization of reduced ordered binary decision diagrams for SLA negotiation in iaas

of cloud computing. In *Service-Oriented Computing*, pp. 268-282, Springer Berlin Heidelberg, 2012.

[104] A. Dirkzwager, M. Hendrikx, and J. D. Ruiter. TheNegotiator: A Dynamic Strategy for Bilateral Negotiations with Time-Based Discounts. *Complex Automated Negotiations: Theories, Models & Software Competitions*, Springer Berlin Heidelberg, pp. 217-221, 2013.

[105] M. Peter and T. Grance. The NIST definition of cloud computing, 2011.

[106] GigaOM. Sector RoadMap: Platform as a Service in 2012. Technical Report, Available: `http://research.gigaom.com/report/sector-roadmap-platform-as-a-service-in-2012`. Accessed: [21/11/2015].

[107] Google. An update on container support on Google Cloud Platform. Available: `http://google-opensource.blogspot.de/2014/06/an-update-on-container-support-on.html`. Accessed: [21/11/2015].

[108] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397-1420, 2012.

[109] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755-768, 2012.

[110] L. Wu, S. K. Garg, and R. Buyya. SLA-based resource allocation for software as a service provider in cloud computing environments. In *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 195-204, 2011.

[111] J. Koomey. Growth in data center electricity use 2005 to 2010. Technical Report, Available: `http://www.analyticspress.com/datacenters.html`. Accessed: [21/11/2015].

[112] G. Zhang. On Variable-Sized Bin Packing. In *Proceedings of the 3rd International Workshop on ARANCE*, Rome, Italy, pp. 117-126. Carleton Scientific, 2002.

[113] K. Dhyani, S. Gualandi, and P. Cremonesi. A Constraint Programming Approach for the Service Consolidation Problem. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 97-101, Springer Berlin Heidelberg, 2010.

[114] J. Anselmi, E. Amaldi, and P. Cremonesi, Service consolidation with end-to-end response time constraints. In *34th Euromicro Conference*, Software Engineering and Advanced Applications, pp. 345-352, 2008.

[115] Google. ROADEF/EURO Challenge 2012: Machine Reassignment. Available: `http://challenge.roadef.org/2012/en/index.php`. Accessed: [21/11/2015].

[116] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, vol. 10, 2008.

[117] L. A. Barroso and U. Hölzle. The case for energy-proportional computing, *Computer*, vol. 40, no. 12, pp. 33-37, IEEE Computer Society, 2007.

[118] RedHat. OptaPlanner. `http://www.optaplanner.org`. Accessed: [21/11/2015].

[119] A.I. Jehangiri, R. Yahyapour, P. Wieder, E. Yaqub, and K. Lu. Diagnosing cloud performance anomalies using large time series dataset analysis. In *IEEE 7th International Conference on Cloud Computing (CLOUD)*, pp. 930-933, IEEE, 2014.

[120] W. Theilmann, J. Happe, C. Kotsokalis, A. Edmonds, K. Kearney, and J. Lambea. A reference architecture for multi-level SLA management. *Journal of Internet Engineering*, vol. 4, no. 1, pp. 289-298, 2010.

[121] OpenShift Community. OpenShift QuickStarts, Available: `https://github.com/openshift-quickstart`. Accessed: [21/11/2015].

[122] D. Pandit, S. Chattopadhyay, M. Chattopadhyay, and N. Chaki. Resource allocation in cloud using simulated annealing. In *Applications and Innovations in Mobile Computing (AIMoC)*, pp. 21-27, 2014.

[123] F. Larumbe and B. Sanso. A Tabu Search algorithm for the location of data centers and software components in green cloud computing networks. *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 22-35, 2013.

[124] J. Hamilton. Infrastructure Innovation Opportunities. Available: `http://mvdirona.com/jrh/talksAndPapers/JamesHamilton_FirstRoundCapital20121003.pdf`. Accessed: [21/11/2015].

[125] Open Cloud Computing Interface (OCCI). `http://occi-wg.org`. Accessed: [21/11/2015].

[126] E. Yaqub. GWDG Platform Cloud prototype and API source code. Available: `http://git.cetic.be/eyaqub/RefExp`. Accessed: [21/11/2015].

[127] H. P. Borges, J. N. De Souza, B. Schulze, and A. R. Mury. Automatic generation of platforms in cloud computing. In *Network Operations and Management Symposium (NOMS)*, pp. 1311-1318, IEEE, 2012.

[128] H. P. Borges, J. N. De Souza, B. Schulze, and A. R. Mury. Automatic services instantiation based on a process specification. *Journal of network and computer applications*, vol. 39, pp.1-16, 2014.

[129] X. Zheng. Qos representation, negotiation and assurance in cloud services, Doctoral Dissertation, 2014.

[130] X. Zheng, P. Martin, K. Brohman, and M. Zhang. Cloud service negotiation: a research report. *International Journal of Business Process Integration and Management*, vol 7, no. 2, pp. 103-113, 2014.

[131] V. Dussaux. Purchasing and offering of cloud software services by a central procurement agency: a case study. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, pp. 80-83. IEEE, 2012.

[132] S. Maity and A. Chaudhuri. Optimal negotiation of SLA in federated cloud using multiobjective genetic algorithms. In *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pp. 269-271, 2014.

[133] M. Sjouke and S. Radomirović. Generalizing Multi-party Contract Signing. *Principles of Security and Trust, Lecture Notes in Computer Science (LNCS)*, vol. 9036, Springer Berlin Heidelberg, pp. 156-175, 2015.

[134] A.F.M. Hani, I.V. Paputungan, and M.F. Hassan. Renegotiation in Service Level Agreement Management for a Cloud-Based System. *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, 2015.

[135] M. Abraham and A. J. Kuttyamma. Automatic Generation of Platforms in Cloud Computing. *International Journal of Computer Science Information Technologies*, vol. 5, no. 3, 2014.

[136] F. J. F. Goméz. Cloud Computing: caracterización de los impactos positivos obtenidos por la utilización del modelo Cloud Computing por las pymes, basado en la tipología de Modelos de Negocio de este tipo de empresas, Doctoral Dissertation, 2014.

[137] H. G. Hamilton. An Examination of Service Level Agreement Attributes that Influence Cloud Computing Adoption. Doctoral Dissertation, 2015.

[138] A. V. Dastjerdi and R. Buyya. An autonomous time-dependent SLA negotiation strategy for cloud computing environments. *The Computer Journal*, 2015.

[139] H. Jayathilaka, K. Chandra, and R. Wolski. Response time service level agreements for cloud-hosted web applications. In *Proceedings of the 6th ACM Symposium on Cloud Computing*, pp. 315-328, ACM, 2015.

Appendix A.

Negotiation Strategy Algorithms

This appendix complements the thesis with description of two negotiation strategy algorithms referred in chapter 6 but deemed too detailed for the main text.

A.1. Reactive Exploitation (RE) Algorithm

Algorithm 1 Reactive Exploitation (RE) Strategy. Let t_{max} be the maximum time, t_c is current time and B_{sort} is list of all bids sorted in descending order on utility. B_l is list of last l opponent bids. u gives the utility of a bid, u_r is the reserved utility value below which a bid is not accepted. b_r is the received bid and b_s is the bid to be sent. b_{best} is best bid received. h is the measure of opponent's hardheadedness and α is acceptable threehold for h. ρ is probability of concession and β is acceptable threshold for ρ.

Require: $t_{max}, u_r, l, B_{sort}, \alpha, \beta$
1: $b_s \leftarrow getStartingBid()$;
2: **while** $t_c \leq t_{max}$ **do**
3: $b_r \leftarrow receiveBid$;
4: $b_{best} \leftarrow updateBestBid$;
5: $h \leftarrow updateHardheadedness$;
6: $t_c \leftarrow remainingTime$;
7: $\rho \leftarrow getConcessionProbability(b_r, t_c)$;
8: **if** $h < \alpha$ **and** $\rho > \beta$ **then**
9: $temp \leftarrow nextConcessionedBid(B_{sort})$;
10: **if** $u(temp) > u(b_{best})$ **and** $u(temp) \geq u_r$ **then**
11: $b_s \leftarrow temp$;
12: **else if** $u(b_{best}) \geq u_r$ **then**
13: $b_s \leftarrow b_{best}$;
14: **end if**
15: **else**
16: $b_s \leftarrow choose(b_{best}, b_s, u_r)$;
17: **end if**
18: **if** $isAcceptable(b_r, b_s, t_c)$ **then**
19: **return** $accept(b_r)$;
20: **else**
21: **return** b_s;
22: **end if**
23: **end while**

Figure A.1.: Reactive Exploitation Negotiation Strategy

Description

Negotiations take place within a standardized time $t \in [0, 1]$. The Reactive Exploitation (RE) algorithm shown in Fig. A.1 starts by giving a minimal concession to avoid any impasse at the beginning, hoping to lure the opponent into reciprocity. Then on, it maintains a mean value of opponent's last l-many bids which it uses to model the hardheadedness $h \in \{0, l-1\}$ of its opponent. h is directly proportional to an increase or decrease in mean value. Lines 8-17 show its bidding function which generates a counter proposal b_s considering α and β thresholds, provided that the proposed bid has better than its reserved utility value u_r. Given utility u of opponent's bid b_r, the concession rate ρ is adapted to current time t_c and opponent as:

$$\rho = \frac{u - 2ut_c + 2(t_c - 1 + \sqrt{(t_c - 1)^2 + u(2t_c - 1)})}{2t_c - 1} \tag{A.1.1}$$

ρ has been suggested as acceptance function in [78], but we employ it in our bidding function instead. The value of ρ increases for low utility bids b_r as time t_c increases. RE also maintains the best bid b_{best} received from opponent in a negotiation session and resorts to it, if it cannot find a better bid. Otherwise, it chooses the next bid from b_{best} or (last) b_s, whichever has higher utility as shown in line 16, until ρ and h change in subsequent negotiation round(s). Lines 18-22 show its acceptance function A, which takes into account b_r, b_s and t_c. Practical experience showed that A increases chances of convergence when acceptable thresholds are fixed against time, rather than relying on dynamic methods. Thus, A accepts a bid when any of the following conditions are met:

$$u(b_r) > 0.98 \tag{A.1.2}$$

$$u(b_r) > u(b_s) \wedge u(b_r) > 0.85 \wedge t_c > 0.95 \tag{A.1.3}$$

$$u(b_r) \geq u_r \wedge t_c > 0.998 \tag{A.1.4}$$

This also induces a late agreement behavior i.e., RE utilizes maximum negotiation time in most cases as major concessions are usually derived in last moments.

A.2. Enhanced Reactive Exploitation (eREx) Algorithm

Algorithm 1 Enhanced Reactive Exploitation Strategy. Let t_{max} be the maximum time, t_c is current time and t_o is estimated response time of opponent. B^z_{int} is list of z randomly explored bids stored internally in descending order for utility. num is number of bids sampled internally per round and $num < z$. B^z_o is list of z opponent bids stored in descending order for utility. u gives utility of a bid at t_c, u_r is the reserved utility value below which agreement is not made. u_l is mean utility of last l opponent bids. b_o is current opponent bid and b_s is the bid to be sent. b_{best} is best bid received from opponent. h is the measure of opponent's hardheadedness and α is acceptable threshold for h. ρ is probability of concession and β is acceptable threshold for ρ. z is a matrix containing Euclidean distances between q maximum utility bids of B^z_{int} and B^z_o.

Require: $t_{max}, u_r, l, num, \alpha, \beta, z, q$
1: $b_s \leftarrow getStartingBid()$;
2: **while** $t_c \leq t_{max}$ **do**
3: $b_o \leftarrow receiveBid$;
4: $B^z_{int} \leftarrow sampleBidSpace(num)$;
5: $B^z_o \leftarrow updateOpponentBidSpace(b_o)$;
6: $b_{best} \leftarrow updateBestBid(b_o)$;
7: $u_l \leftarrow updateMeanUtility$;
8: $h \leftarrow updateHardheadedness(u_l)$;
9: $t_o \leftarrow estimateResponseTime$;
10: $t_c \leftarrow currentTime$;
11: $\rho \leftarrow getConcessionProbability(b_o, t_c)$;
12: **if** $u(b_s) < u_{best}(B^z_{int})$ **then**
13: $b_s \leftarrow u_{best}(B^z_{int})$;
14: **else if** $t_c < 0.99$ **then**
15: $tempSpace \leftarrow filterSpace(u(b_s), u_{best}(B^z_{int}))$;
16: $b_s \leftarrow tempSpace.getMedianBid()$;
17: **end if**
18: **if** $h < \alpha$ **and** $\rho > \beta$ **then**
19: $tempBid \leftarrow getConcessionBid(B^z_{int}, B^z_o)$;
20: **if** $u(tempBid) > u(b_{best})$ **and** $u(tempBid) \geq u_r$ **then**
21: $b_s \leftarrow tempBid$;
22: **else if** $u(b_{best}) \geq u_r$ **then**
23: $b_s \leftarrow b_{best}$;
24: **end if**
25: **else**
26: $b_s \leftarrow choose(b_{best}, b_s, u_r)$;
27: **end if**
28: **if** $isAcceptable(b_o, b_s, t_c, t_o)$ **then**
29: **return** $accept(b_o)$;
30: **else**
31: **return** b_s;
32: **end if**
33: **end while**

Figure A.2.: Enhanced Reactive Exploitation Negotiation Strategy

Description

Negotiations take place within a standardized time $t \in [0, 1]$. The Enhanced Reactive Exploitation (eREx) algorithm shown in Fig. A.2 starts with the best bid explored so far in its internal space B^z_{int} with sampling size $2{*}num$ in the first round. The start bid is usually not of absolute high utility value 1.0, which helps avoid a possible impasse at the very beginning and attempts to lure the opponent into reciprocity. Then on, it samples num bids per round and progressively transforms B^z_{int} into an elitist bid space. A mean value of opponent's last l-many bids is maintained as u_l which is used to model the hardheadedness $h \in \{0, l-1\}$ of opponent. h is directly proportional to u_l i.e., h is incremented if $u(b_o) < u_l$ and decremented otherwise. The strategy also stores opponent bids in a list B^z_o, removing the least utility bid if size z is exceeded. The response time of opponent is estimated regularly as t_o. Lines 12-27 show the bidding function B. Here, lines 12-17 prepare a default counter offer, resorting to the last sent offer b_s if its utility is still the best among sampled space B^z_{int}. Otherwise, a median utility bid from a temporary space filtering bids between utility $u(b_s)$ and $u_{best}(B^z_{int})$ is set as a counter offer. This step is skipped when t_c exceeds 0.99.

Lines 18-27 show the concession criteria which may overwrite the default counter offer b_s if h and ρ satisfy the α and β thresholds and has better than the reserved utility value u_r. u_r is read as defined in preference profile and varies with time for discount based profiles. Given utility u of opponent's bid b_r, the concession rate ρ is adapted to current time t_c and opponent according to A.1.1. The next concession bid is generated according to the function in Fig. A.3. Here,

Require: q, B^z_{int}, B^z_o
1: $q_{row} \leftarrow getMaxUtilityBids(q, B^z_{int})$;
2: $q_{col} \leftarrow getMaxUtilityBids(q, B^z_o)$;
3: $z \leftarrow euclideanDistanceMatrix(q_{row}, q_{col})$;
4: $leastApartBids \leftarrow getMinDistancedBidsPerRow(z)$;
5: $tempBid \leftarrow getMaxUtilityBid(temp)$;
6: **return** $tempBid$;

Figure A.3.: Method getConcessionBid

a max-min-max method is developed which i) picks q maximum utility bids from B^z_{int} and B^z_o, ii) computes a qXq matrix of Euclidean distances between these bids and picks a minimum distanced bid per row. These $leastApartBids$ pre-

serve some diversity as these may not be globally the least apart bids. iii) the bid with maximum utility is selected from $leastApartBids$ and returned as candidate counter offer. eREx also maintains the best bid b_{best} received from opponent in a negotiation session and resorts to it, if it cannot find a better bid. Otherwise, it chooses the next bid from b_{best} or b_s, whichever has higher utility as shown in line 26. This allows a rational deflection from usual tit-for-tat tactic in order to maximize payoff when the opponent's bid provide higher utility than our own.

Lines 28-32 show our acceptance function A, which takes into account b_r, b_s and t_c. Experience showed that A increases convergence when acceptable thresholds are fixed against time, rather than relying on dynamic methods. Hence, we define A to accept opponent bid b_r when any of the following conditions are met:

$$u(b_r) > 0.9 \tag{A.2.1}$$

$$u(b_r) > u(b_s) \wedge u(b_r) > 0.85 \wedge t_c > 0.95 \tag{A.2.2}$$

$$u(b_r) \geq u_r \wedge (t_c + t_o) > 1.0 \tag{A.2.3}$$

Appendix B.

Curriculum Vitae

Personal Data

Name:	Edwin Yaqub
Date of Birth:	18.10.1980
Nationality:	German
Email:	edwinyaqub@gmail.com

Education

2015	PhD student in Computer Science Chair for Practical Computer Science, University of Göttingen, Germany
2008:	Master in Computer Science RWTH Aachen University, Germany
2003:	Bachelor in Computer Science University of Central Punjab, Lahore, Pakistan

Professional Experience

Research Assistant
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG), Göttingen, Germany
October 2011 - to date

- Research and development on SLA Management, Cloud Computing, prototyping for in-house projects and work on EU Project PaaSage.

Research Assistant
Technical University of Dortmund, Germany
October 2009 - September 2011

- Research and development on SLA Management for EU Project SLA@SOI.

IT Consultant
CGI Systems, Düsseldorf, Germany
December 2008 - June 2009

- Requirements analysis, software development and testing of Vodafone online shop using the ATG eCommerce framework and SCRUM methodology.

Student Researcher
Siemens Industrial Solutions and Services, Aachen, Germany
January 2008 - November 2008

- Developed webservices and persistence layer as part of work on EU project CVIS (Cooperative Vehicle-Infrastructure System).

Master Thesis
Philips Research Laboratories, Aachen, Germany
March 2007 - October 2007

- Research and development of a model based framework to represent and interpret expert medical knowledge on patient health data.

Software Engineer
Innovative Private Ltd., Lahore, Pakistan
April 2004 - August 2005

- Design, development and testing of web based project targeting debit card based commission chains.

Software Engineer
Adamsoft International, Lahore, Pakistan
July 2003 - April 2004

- Design and development of an ERP application using model driven architecture (MDA) and code generation techniques.

Teaching Experience

University of Göttingen

- Exercises or lectures for Service Computing, Parallel Computing, Practical Course on Parallel Computing (SS 2014, WS 2013, SS 2013, WS 2011).
- Workshops (SS 2013 and Source Talk Tage 2013)

Technical University of Dortmund

- Exercises for the course Webtechnologien 1 (WS 2010).

Invited Talks

- Title: SLA Negotiation. At: ICT 2010 Event (co-hosted by the European Commission), Brussels, September, 2010.
- Title: SLA@SOI Negotiation Platform. At: European Commission's second annual review of SLA@SOI Project, Brussels, September, 2010.

Certification

- Certified Software Tester - Foundation Level (CTFL), International Software Testing Qualifications Board (ISTQB).

Honours and Awards

- Won performance award at Innovative Private Ltd. for work quality.
- Won academic merit scholarship during the Bachelors degree program.

Publications

- **Protocol-generic and Utility optimizing Negotiations for SLA based Service Procurement in Cloud Markets**. Yaqub, E., Yahyapour, R., Wieder, P., Kotsokalis, C., Lu, K., and Jehangiri, A. I. *Future Generation Computer Systems*. (Status: in review), 2015.

- **Metaheuristics-based Planning and Optimization for SLA-aware Resource Management in PaaS Clouds**. Yaqub, E., Yahyapour, R., Wieder, P., Kotsokalis C., Lu, K., and Jehangiri, A.I. In *7th IEEE/ACM International Conference on Utility and Cloud Computing (IEEE/ACM UCC)*, IEEE Computer Society, ACM Society, London, UK, 2014.

- **Optimal Negotiation of Service Level Agreements for Cloud-based Services through Autonomous Agents**. Yaqub, E., Yahyapour, R., Wieder, P., Kotsokalis C., Lu, K., and Jehangiri, A.I. In *11th IEEE International Conference on Services Computing (IEEE SCC)*, pp.59-66, DOI 10.1109/SCC.2014.17, IEEE Computer Society, Alaska, USA, 2014.

- **A Protocol Development Framework for SLA Negotiations in Cloud and Service Computing**. Yaqub, E., Yahyapour, R., Wieder, P., and Lu, K. In *9th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON 2012)*, Vanmechelen, K., Altmann J., Rana O.F. (Eds.), LNCS 7714, pp. 1-15, Springer-Verlag Berlin Heidelberg, 2012.

- **A Generic Platform for Conducting SLA Negotiations**. Yaqub, E., Wieder, P., Kotsokalis, C., Mazza, V., Pasquale, L., Lambea, R. J., Garcia, G. S., and Escamez, C.A. *Service Level Agreements for Cloud Computing*, (Eds.) Wieder, P., Butler, J.M., Theilmann, W., Yahyapour, R., pp. 187-206, Springer-Verlag, 2011.

- **Distributed Guidelines (DiG): A Software Framework for Extending Automated Health Decision Support to the General Population**. Yaqub, E. and Barroso, A. *Perspectives in Health Information Management/AHIMA*, American Health Information Management Association, vol. 7 (Summer), no. 1b, 2010.